Integrating Induction and Deduction in Model Checking

Sanjit A. Seshia

EECS Department
UC Berkeley

Clarke Symposium
September 19, 2014
E. M. Clarke and E. A. Emerson, 1981:

“We propose a method of constructing concurrent programs in which the *synchronization skeleton of the program is automatically synthesized* from a high-level (branching time) Temporal Logic specification.”

(1st sentence of their seminal model checking paper)
Three Messages in this Talk
[Seshia DAC’12; Jha & Seshia, SYNT’14]

1. Verification by Reduction to Synthesis
   - Many (verification) tasks involve synthesis

2. Induction + Deduction + Structure: An Effective Approach to Synthesis:
   - *Induction*: Learning from examples
   - *Deduction*: Logical inference and constraint solving
   - *Structure*: Hypothesis on syntactic form of artifact to be synthesized
   - “Syntax-Guided Synthesis” [Alur et al., FMCAD’13]
     - Inspired by Counterexample-guided abstraction refinement (CEGAR) [Clarke et al., CAV’00]

   - Analysis of Counterexample-Guided Synthesis
     - Sample Complexity, Convergence, Search Strategies
Artifacts Synthesized in Verification

- Inductive invariants
- Abstraction functions / abstract models
- Auxiliary specifications (e.g., pre/post-conditions, function summaries)
- Environment assumptions / Interface specifications
- Interpolants
- Ranking functions
- Intermediate lemmas for compositional proofs
- Theory lemma instances in SMT solving
- Patterns for Quantifier Instantiation
- …
Formal Verification as Synthesis

- Inductive Invariants
- Abstraction Functions
One Reduction from Verification to Synthesis

NOTATION
Transition system $M = (I, \delta)$
Safety property $\Psi = G(\psi)$

VERIFICATION PROBLEM
Does $M$ satisfy $\Psi$?

SYNTHESIS PROBLEM
Synthesize $\phi$ s.t.

\[
I \Rightarrow \phi \land \psi \\
\phi \land \psi \land \delta \Rightarrow \phi' \land \psi'
\]
Two Reductions from Verification to Synthesis

NOTATION
Transition system $M = (I, \delta)$, $S = \text{set of states}$
Safety property $\Psi = G(\psi)$

VERIFICATION PROBLEM
Does $M$ satisfy $\Psi$?

SYNTHESIS PROBLEM #1
Synthesize $\phi$ s.t.
$I \Rightarrow \phi \land \psi$
$\phi \land \psi \land \delta \Rightarrow \phi' \land \psi'$

SYNTHESIS PROBLEM #2
Synthesize $\alpha : S \rightarrow \hat{S}$ where $\alpha(M) = (\hat{I}, \hat{\delta})$
s.t.
$\alpha(M)$ satisfies $\Psi$
iff
$M$ satisfies $\Psi$
Counterexample-Guided Abstraction Refinement is “Inductive” Synthesis

SYNTHESIS

- System + Property
- Initial Abstraction Function
- Abstract Domain

Generate Abstraction

- New Abstraction Function
- Refine Abstraction Function

Refine Abstraction Function

Abstract Model + Property

VERIFICATION

- Invoke Model Checker
- Counterexample

Check Counterexample: Spurious?

- Spurious Counterexample
- YES

Spurious Counterexample

Valid

- Done

No

Done
Lazy SMT Solving performs “Inductive” Synthesis (of Lemmas)

SYNTHESIS

Initial Boolean Abstraction

Generate SAT Formula

Blocking Clause/Lemma

SAT Formula

Proof Analysis

“Spurious Model”

VERIFICATION

Invoke SAT Solver

Invoke Theory Solver

SAT (model) ("Counter-example")

SAT

Done

UNSAT

Done
CEGAR & Lazy SMT perform Active Learning from (Counter)Examples

Difference from standard learning theory:
Learning Algorithm and Verification Oracle are typically “General” Solvers, independent of concept class

- INITIALIZE
  - “Concept Class”, Initial Examples

- LEARNING ALGORITHM
  - Learning Fails

- VERIFICATION ORACLE
  - Learning Succeeds

- Candidate Concept
  - Counterexample
Machine Learning Theory ↔ Formal Methods: 2 Sample Results

- **Lower Bounds on Convergence of Counterexample-guided loop**
  - **Teaching Dimension (TD):** *Minimum* number of (labeled) examples a teacher must reveal to *uniquely* identify any concept from a class
  - **Thm:** TD is a lower bound on # iterations for counterexample-guided synthesis

- **Impact of “Quality” of Counterexamples**
  - Does the type of counterexample affect convergence for *infinite-size* concept classes?
  - **Thm:** Minimum counterexamples are no better than arbitrary counterexamples

[Jha & Seshia, SYNT’14]
Conclusion

- Model Checking and Synthesis: many connections

- Verification by Reduction to Synthesis

- Approach for Synthesis: Induction + Deduction + Structure
  - “Syntax-Guided Synthesis” [Alur et al., FMCAD’13]
  - Inspired by Counterexample-guided abstraction refinement (CEGAR) [Clarke et al., CAV’00]

  - Sample Complexity, Convergence, Search Strategies