Sciduction: Combining Induction, Deduction and Structure for Verification and Synthesis

(abridged version of DAC slides)

Sanjit A. Seshia

Associate Professor
EECS Department
UC Berkeley

Design Automation Conference
June 5, 2012
A Perspective on Formal Methods

Model Checking

Theorem Proving

BDDs & Symbolic methods

SAT

SMT

What can we learn?
WHAT’S NEXT?
The Human Aspect

System Model
Environment Model
Specification

Auxiliary Inputs (abstraction, invariants, compositional lemmas, etc.)

VERIFICATION TOOL

DON’T KNOW
VALID 😊
ERROR 😞
DEBUG

No Result 😞
The End Goal

- Improve designer / programmer creativity and productivity
  - Automate tedious tasks
  - Enable user to express creative insights
  - Correct-by-construction synthesis (from high-level spec.)

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 original model checking paper)
Verification and Synthesis: Where Do We Spend Time?

Environment Model $E$

Specification $\Phi$

System $S$

VERIFIER

S is CORRECT / Buggy

SYNTHESIZER

Environment Model $E$

Specification $\Phi$

System $S$

(satisfying $\Phi$ under $E$)
Artifacts Synthesized in Verification

- Inductive / auxiliary invariants
- Auxiliary specifications (e.g., pre/post-conditions, function summaries)
- Environment assumptions / Environment model / interface specifications
- Abstraction functions / abstract models
- Interpolants
- Intermediate lemmas for compositional reasoning
- Theory lemma instances in SMT solving
- ...

EVERYTHING IS A SYNTHESIS PROBLEM!
Perspectives, so far...

- Verification “=” Synthesis
  - The hard parts of verification involve “synthesis sub-tasks”

- 3 Challenges; Human input is crucial
  - Writing specifications
  - Modeling environment
  - Guiding verification/synthesis engine

- How to help users provide creative input while automating tedious tasks?
The Lens: Examining Human-Computer Interaction in Verification

- User identifies synthesis sub-task
  - “Generate abstract model”
  - “Use localization abstraction”

and expresses creative insight

- Tool automates search
  - Counterexample-guided abstraction refinement (CEGAR) using DPLL-based SAT solving

**DEDUCTION**: General to specific
+ **INDUCTION**: Specific to general
Sciduction

Structure-Constrained Induction and Deduction

Inductive Reasoning
(Active Learning: Generalizing from Examples)

+ 

Deductive Reasoning
(“Lightweight” Logical inference & Constraint solving)

+ 

Structure Hypotheses
(on artifacts to be synthesized)
Demonstrated Applications

- Timing analysis of software
- RTL verification
- Synthesis from temporal logic
- Program synthesis
- Switching logic synthesis

Floating-point to fixed-point

Structure Hypothesis
Inductive Inference
Deductive Reasoning
Counterexample-guided Abstraction Refinement involves Synthesis

The structure hypothesis is the abstract domain

- Design Property
- Abstraction Function
- Abstract Domain
- New Abstraction Function
- Generate Abstraction
- Abstract Model + Property
- Invoke Model Checker
- Check Counterexample: Spurious?
- Spurious Counterexample

Valid

Done

YES

NO

Done
Approach

1. Identify the synthesis sub-task(s)
2. Make structure hypothesis
3. Devise top-level synthesis strategy (inductive or deductive)
4. Devise subroutines (inductive or deductive)
Synthesis Sub-Task

Find artifact satisfying specification $\Psi$

CEGAR

- $C_S = \text{All (finite-state) abstract models}$
- $\Psi = \text{Abstract model must be}$
  - sound (over-approximate)
  - complete (no spurious counterexamples)
Structure Hypothesis H

- Shrink set of artifacts from $C_S$ to $C_H$

CEGAR

- $H =$ The abstract domain (localization abstraction)
- $C_H =$ Abstract models generated using $H$
Top-Level Strategy

- Top-level search strategy for:
  \[ \exists c \in C_H \text{ s.t. } c \text{ satisfies } \Psi? \]

CEGAR

- Learn from spurious counterexamples
  - most over-approximate model satisfying \( \Psi \)
- Soundness (trivial): by construction (over-approximation)
- Completeness: the original concrete system is in \( C_H \)
Induction

- Learning algorithm
  - Active learning: choose examples to learn from

CEGAR

- Example: Spurious Counterexample
- Partially concretize abstract model to rule out spurious counterexample
  - CEGAR as Inductive Learning [Anubhav Gupta, PhD thesis 2006]
Deduction

- Lightweight decision procedure
  - Solves decision problem that is “easier” than original
  - Generates examples, labels for examples, verifies artifact, etc.

CEGAR

- Model checker
  - Generates counterexample, if one exists

- SAT solver
  - Checks if counterexample is spurious
The structure hypothesis is the abstract domain
Related Work: A Sample

- **Instances of Sciduction** (also inspiration!)
  - CEGAR [Clarke et al., ’00]
  - Compositional Reasoning, Invariant Generation based on Automata Learning (L*) [Cobleigh et al, ’03]
  - Counterexample-guided inductive synthesis (CEGIS) [Solar-Lezama et al., ’06]

- **Purely Deductive Generalization**
  - DPLL-based SAT solvers
  - Lazy SMT solvers -- DPLL(T)
  - Automata-theoretic synthesis from LTL