Integrating Induction and Deduction for Verification and Synthesis

Sanjit A. Seshia

Associate Professor
EECS Department
UC Berkeley

DATE 2013 Tutorial
March 18, 2013
Bob’s Vision: Exploit Synergies between Verification and Synthesis

- VIS: Verification Interacting with Synthesis
  - Continued in the ABC project

- Techniques useful in formal verification (e.g., SAT) help in logic synthesis, and vice-versa (logic optimization helping verification engines)
  - Our work: Beaver SMT solver uses ABC to optimize SAT encoding

- The connection between verification and synthesis goes deeper…
Artifacts synthesized in Verification

Hard part of verification is often a synthesis step

- Specifications (e.g., inductive invariants, pre/post-conditions, function summaries)
- Environment assumptions / Env model / interface specifications
- Abstraction functions / abstract models
- Interpolants
- Ranking functions
- Intermediate lemmas for compositional reasoning
- ...

Human Insight often needed to synthesize the “right” artifact
Strategy for Synthesis

- **Some Human Input / Insight**
  - Hypothesis on the form of artifact to be synthesized
  - “Structure Hypothesis”
  - Example:
    - Template for invariant (e.g., “equivalences”)
    - Simulations may suggest likely equivalences [ABC strategy]

- **Induction + Deduction**
  - Induction: specific examples $\rightarrow$ general rules
    - Learning from examples
  - Deduction: general rules $\rightarrow$ specific conclusions
    - Logical inference and constraint solving
  - Purely deductive approach is inefficient / inapplicable
  - Purely inductive approach gives no guarantees
Structure-Constrained Induction and Deduction

**Structure** Hypotheses
(on artifacts to be synthesized)

**Deductive** Procedure
“Lightweight”: solves lower complexity problem or special case of original decision problem

**Inductive** Procedure
Active Learning: selects examples to learn from

Rest of the Talk

- Synthesizing Environment Assumptions for Reactive Synthesis from Linear Temporal Logic (LTL)

- Conclusions and Future Directions
Reactive Synthesis from LTL

LTL Specification -> Synthesis Tool -> Finite-State Transducer (reactive program)

Church 1957
Rabin 1972
Pnueli-Rosner 1989

2013: many potential applications in Robotics & Embedded Systems
A Personal Story of Synthesis from LTL

- Verified Electronic Voting Machine
  - Verilog design, list of LTL properties [Sturton et al., CCS 2009]
  - http://uclid.eecs.berkeley.edu/vvm/
- Attempt 1: Synthesizer ran for more than a week, no output
- Idea: Compositional Synthesis! Synthesize individual modules (selections within contests, navigation between contests, etc.)
- Attempt 2, 3, …: Unrealizable! Too many environment assumptions needed (at interfaces between modules)
Often due to incomplete environment assumption!
Satisfiability and Realizability

- A LTL formula $\phi$ is **satisfiable** if there exists an infinite word (i.e. sequence of inputs and outputs) that satisfies $\phi$.

- A LTL specification $\phi$ is **realizable** if for all inputs, there exists a finite-state transducer $M$ (e.g. a Moore machine) which generates computations that satisfies $\phi$. 
Problem Description

- **Goal:**
  Generate additional assumptions to enable synthesis

- **Context:**
  Original specification is satisfiable but unrealizable

- **Assume:**
  - Given only a few interesting user scenarios (satisfying traces)
  - Specifications are in the GR(1) class

- **Challenge:**
  - Space of possible additional assumptions is huge
  - Want assumptions that can be understood and analyzed by a human user
Example

- Inputs: request $r$ and cancel $c$
- Outputs: grant $g$
- System specification $\varphi_s$:
  - $G (r \rightarrow X F g)$
  - $G(c \lor g \rightarrow X \neg g)$
- Environment assumption $\varphi_e$:
  - True
- No user scenarios.
- Not realizable because the environment can force $c$ to be high all the time
Our Contribution

Counterstrategy-guided synthesis of environment assumptions

- Demonstrated to generate useful/intuitive environment assumptions for digital circuits and robotic controllers

[Wenchao Li et al., MEMOCODE 2011]
Approach for Synthesizing Environment Assumptions

**Structure Hypothesis:**
Environment Assumptions are Restricted GR(1) properties

+ 

**Inductive Inference:**
Version Space Learning

+ 

**Deductive Engine:**
(Finite-state) Model Checking
GR(1) Synthesis [Piterman, Pnueli, Saar]

- Formulas in the form: $\varphi_e \rightarrow \varphi_s$
  - Input and output partitions $I$ and $O$.
  - $\varphi_\alpha^i$: initial state formulas.
  - $\varphi_\alpha^t$: transition formulas, in the form of $G B$, where $B$ is a Boolean combination of variables in $I \cup O$ and expressions $X u$, $u \in I$ if $\alpha = e$ and $u \in I \cup O$ if $\alpha = s$.
  - $\varphi_\alpha^f$: fairness formulas, in the form of $G F B$, where $B$ is a Boolean formula over $I \cup O$.
- Synthesis as a turn-based two-player game between the system and the environment
  - Realizable if the system has a winning strategy, otherwise env wins; Strategy representable as finite-state transducer
Counter-strategy and counter-trace

- Counter-strategy is a strategy for the environment to force violation of the specification.

- Counter-trace is a fixed input sequence such that the specification is violated regardless of the outputs generated by the system.
Example

- System $\varphi_s$:
  - $G (r \rightarrow X F \neg g)$
  - $G(c \lor g \rightarrow X \neg g)$
- A counter-trace:
  - $r$: 1 1 (1)
  - $c$: 1 1 (1)
Assumption Mining to Assist Synthesis

1. Formal Specification
2. Synthesis Tool
3. Unrealizable
   - Compute Counterstrategy
4. Realizable
5. Done

Start

Specification Templates

A Few User Scenarios

Mine Assumptions

Add

18
Mining Algorithm

Specification Templates
User Scenarios
Counter-strategy

Candidate Spec $\varphi$, e.g. $G F_p$
New Candidate $\varphi'$

Model Checker

Trace Verifier

Pass? Yes
Add $\varphi$

$\neg \varphi$

Pass?

No

Yes

No

Yes
Assumption Templates follow GR(1)

- $\gamma^1$: $G F b$, where $b \in I$
- $\gamma^2$: $G (b_1 \rightarrow X b_2)$, where $b_1 \in I \cup O$ and $b_2 \in I$
- $\gamma^3$: $G (b_1 \lor b_2)$, where $b_1, b_2 \in I$

- The specification remains in GR(1) after the addition of new assumptions.
- How to pick candidate assumptions: (next slide)
  - Weakest to Strongest, with heuristics
  - $G F b$ weaker than $G X b$ weaker than $G b$
  - IMPORTANT: Check each assumption for consistency with existing set
Theoretical Results

- **Theorem 1:** A redundant environment assumption (implied by the existing assumptions) is never added.
  - Proof sketch: guaranteed by the counter-strategy guided approach.
- **Corollary:** The set of environment assumptions is minimal (but not minimum, in terms of number of properties).
  - Example: We may find $G\ p$ and $G\ q$ rather than $G\ (p \land q)$

- **Theorem 2:** [Completeness] If there exist environment assumptions under our structure hypothesis that make the spec realizable, then the procedure finds them (terminates successfully).
  - “conditional completeness” guarantee

- **Theorem 3:** [Soundness] The procedure never adds inconsistent environment assumptions.
Example

- **System $\phi_s$:**
  - $\mathbf{G} (r \rightarrow X \mathbf{F} \neg g)$
  - $\mathbf{G}(c \lor g \rightarrow X \neg g)$

- **A counter-trace:**
  - $r$: 1 1 (1)
  - $c$: 1 1 (1)

- **Test assumption candidates by checking its negation:**
  - $\mathbf{G} (\neg c)$
  - $\mathbf{G} (F \neg c)$

- **Environment $\phi_e$:**
  - $\mathbf{G} (\neg c)$

- **System $\phi_s$:**
  - $\mathbf{G} (r \rightarrow X \mathbf{F} \neg g)$
  - $\mathbf{G}(c \lor g \rightarrow X \neg g)$

- **Environment $\phi_e$:**
  - $\mathbf{G} (F \neg c)$

*Realizable!*
Experimental Results Summary

- **Experiment Setup:**
  - Remove assumptions from an originally realizable specification
  - Use a few (often a single) satisfying traces of the original specification as representative user scenarios
  - Mine additional assumptions until the specification is realizable

- Use Cadence SMV to generate the satisfying traces and model check the counter-strategies

- Use RATSY [Bloem et al. 2010] to check realizability of the specifications and compute the counter-strategies and counter-traces in case of unrealizability
Result Summary:

- Case studies in existing literature: AMBA AHB, IBM Gen Buffer, robotic vehicle controller, etc.
- Recovered the missing assumption in most cases
- AMBA AHB Example:

Original assumption: \( G (\text{HLOCK}[0] = 1 \rightarrow \text{HBUSREQ}[0] = 1) \)

Mined assumption: \( G (F \ \text{HLOCK}[0] = 0) \)

Master 0 requests locked access to the bus

Master 0 requests access to the bus
Related Work

• Constructing the “weakest” assumption needed for realizability from the game graph [Chatterjee et al. 2008]
  o Computes a safety assumption that removes a minimal set of environment edges from the game graph
  o Computes a liveness assumption that puts fairness on the remaining environment edges
  o The additional environment assumption is a single Büchi automaton which can be difficult for a normal human user to analyze or even understand.

• Computing counter-strategy for GR(1) synthesis [Konighofer et al. 2009]
  o Counter-strategies and counter-traces as explanations for unrealizability
Discussion

- Our approach can generate useful environment assumptions to cope with unrealizability
  - More experimentation underway \(\rightarrow\) compositional synthesis
- Limitation: choice of templates
  - Can extend same approach with broader set of templates
- Can the same idea of learning from counter-strategies and counter-traces be applied to other synthesis tasks?
  - i.e., not just synthesis from LTL
Conclusion: Induction + Deduction + Structure

- Verification “=” Synthesis
- Structure Hypothesis encodes human insight about form of artifact
- Synthesis procedure combines inductive inference with deductive reasoning
- Several demonstrations
  - Abstraction-based verification of RTL designs [Brady et al, ’11]
  - Synthesis of loop-free programs [Jha et al, ’10]
  - Switching logic synthesis for safety and optimality [Jha et al ’10,’11]
  - Environment assumptions for LTL synthesis [Li et al ’11]
  - Fixed-point code from floating-point [Jha, Seshia, ’11]
  - …