Integrating Induction, Deduction and Structure for Synthesis

Sanjit A. Seshia
Associate Professor
EECS Department
UC Berkeley

Collaborators: A. Tiwari, S. Gulwani, J. Deshmukh, X. Jin

NSF ExCAPE Summer School
June 12, 2013
Connections in this Talk

- Software Synthesis
- Formal Verification
- Machine Learning (Theory)
Two Messages

1. **Synthesis Everywhere**
   - Many problems can be solved effectively when viewed as synthesis

2. **Effective Approach: Induction + Deduction + Structure**
   - *Induction*: Learning from examples
   - *Deduction*: Logical inference and constraint solving
   - *Structure*: Hypothesis on syntactic form of artifact to be synthesized
Many Problems map to Synthesis

- **Formal Verification**
  - Synthesis of “verification artifacts”
  - E.g.: Inductive invariants, abstraction functions

- **Formal Specification**
  - E.g., synthesis of temporal logic formulas

- **Reverse Engineering**
  - E.g., malware analysis

- **System Modeling**
  - E.g., timing models for embedded platforms

...
Formal Verification as Synthesis

- Inductive Invariants
- Abstraction Functions
Example Verification Problem

- Transition System
  - Init: \( I \)
    \[ x = 1 \land y = 1 \]
  - Transition Relation: \( \delta \)
    \[ x' = x + y \land y' = y + x \]
- Property: \( \Psi = G (y \geq 1) \)
- Attempted Proof by Induction:
  \[ y \geq 1 \land x' = x + y \land y' = y + x \Rightarrow y' \geq 1 \]
- Fails. Need to Strengthen Invariant: Find \( \phi \) s.t.
  \[ x = 1 \land y = 1 \Rightarrow \phi \]
  \[ \phi \land y \geq 1 \land x' = x + y \land y' = y + x \Rightarrow \phi' \land y' \geq 1 \]
Example Verification Problem

- **Transition System**
  - **Init**: $I$
    \[ x = 1 \land y = 1 \]
  - **Transition Relation**: $\delta$
    \[ x' = x + y \land y' = y + x \]

- **Property**: $\Psi = G (y \geq 1)$

- **Attempted Proof by Induction**:
  \[ y \geq 1 \land x' = x + y \land y' = y + x \implies y' \geq 1 \]
  
  - **Fails. Need to Strengthen Invariant**: Find $\phi$ s.t.
    \[ x \geq 1 \land y \geq 1 \land x' = x + y \land y' = y + x \implies x' \geq 1 \land y' \geq 1 \]

- **Safety Verification $\Rightarrow$ Invariant Synthesis**
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.
$$\alpha(M) \text{ satisfies } \Psi$$
iff
$$M \text{ satisfies } \Psi$$
Common Approach for both: “Inductive” Synthesis

Synthesis of:-

- **Inductive Invariants**
  - Choose templates for invariants
  - Infer likely invariants from tests (examples)
  - Check if any are true inductive invariants, possibly iterate

- **Abstraction Functions**
  - Choose an abstract domain
  - Use Counter-Example Guided Abstraction Refinement (CEGAR)
Counterexample-Guided Abstraction Refinement is Inductive Synthesis

[Anubhav Gupta, ‘06]

Diagram:
- System + Property
- Initial Abstraction Function
- Abstract Domain
- Generate Abstraction
- New Abstraction Function
- Refine Abstraction Function
- Abstract Model + Property
- Invoke Model Checker
- Valid
- Done
- Counterexample
- Check Counterexample: Spurious?
- Spurious Counterexample
- YES
- NO
- Done

Flow:
1. System + Property
2. Initial Abstraction Function
3. Abstract Domain
4. Generate Abstraction
5. New Abstraction Function
6. Refine Abstraction Function
7. Abstract Model + Property
8. Invoke Model Checker
9. Valid
10. Done
11. Counterexample
12. Check Counterexample: Spurious?
13. Spurious Counterexample
14. YES
15. NO
16. Done
CEGAR = Inductive Synthesis

**INITIALIZE**

Structure Hypothesis, Initial Examples

**SYNTHESIZE**

Candidate Artifact

Counterexample

Synthesis Fails

**VERIFY**

Verification Succeeds
CEGAR = Inductive Synthesis = Learning from (Counter)Examples

1. **INITIALIZE**
   - “Concept Class”, Initial Examples

2. **LEARNING ALGORITHM**
   - Candidate Concept
   - Counterexample
   - Learning Fails

3. **VERIFICATION ORACLE**
   - Learning Succeeds
Active Learning: Key Elements

1. **Search Strategy**: How to search the space of candidate concepts?
2. **Example Selection**: Which examples to learn from?
Counterexample-Guidance: A Successful Paradigm for Synthesis and Learning

- **Active Learning** from Queries and Counterexamples [Angluin ’87a,’87b]
- Counterexample-Guided Inductive Synthesis (CEGIS) [Solar-Lezama et al., ’06]
  - See upcoming lectures
- Both rely heavily on Verification Oracle
- Other effective active learning methods?
  - Especially when Verification Oracle is expensive
Induction + Deduction + Structure

**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

Reverse Engineering Malware

Obfuscated code:
Input: y  Output: modified value of y

```c
{ a=1; b=0; z=1; c=0;
  while(1) {
    if (a == 0) {
      if (b == 0) { y=z+y; a =~a; b=~b; c=~c; if (~c) break; }
      else { z=z+y; a=~a; b=~b; c=~c; if (~c) break; }
    } else if (b == 0) {z=y << 2; a=~a;}
    else { z=y << 3; a=~a; b=~b;}
  }
}
```

What it does:

\[ y = y \times 45 \]

We solve this using program synthesis.

FROM
CONFICKER WORM

Reverse Engineering Malware

Obfuscated code:
Input: y      Output: modified value of y

{ a=1; b=0; z=1; c=0;
  while
    if (a == 0) {
      if (b == 0) { y=z+y; a =~a; b=~b; c=~c; if (~c) break; }
      else {
        z=z+y; a=~a; b=~b; c=~c; if (~c) break; }
    }
    else if (b == 0) {z=y << 2; a=~a;}
    else { z=y << 3; a=~a; b=~b; }
} }

What it does:
FROM CONFICKER WORM

Loop-free compositions of “components”
(+, -, <<, >>, *, if-then-else,...)
+ Learning from Distinguishing I/O Examples
+ SMT Solving (bit-vector arithmetic)

What it does:
FROM CONFICKER WORM

Learning from Distinguishing I/O Examples
+ SMT Solving (bit-vector arithmetic)

Synthesizing Switching Logic for Hybrid Systems

- SAFETY: Room Temperature $x$ must lie between 20 and 22 C.
- OPTIMALITY: Minimize switching between modes to save energy

Papers: S. Jha et al., ICCPS 2010 and EMSOFT 2011.
Synthesizing Switching Logic for Hybrid Systems

Guards are Hyperboxes

Hyperbox Learning from +/- Examples (safe/unsafe switching states)

Numerical Simulation (constraint solving)

• SAFETY: Room Temperature $x$ must lie between 20 and 22 C.
• OPTIMALITY: Minimize switching between modes to save energy

Papers: S. Jha et al., ICCPS 2010 and EMSOFT 2011.
Reactive Synthesis from LTL

- Environment Assumptions
- System Requirements

Synthesis Tool

$\phi_e \rightarrow \phi_s$

Realizable

Unrealizable

Often due to missing environment assumptions!

Reactive Synthesis from LTL

Environment Assumptions

Env Assumptions are Restricted GR(1)  
+  
Version Space Learning  
(from Counterstrategies)  
+  
(finite-state) Model Checking

Often due to missing environment assumptions!

Outline

- Program Synthesis from Input-Output Examples
- Synthesizing Environment Assumptions for Reactive Synthesis from LTL
- Conclusions and Future Directions
Obfuscated code:

Input: y  Output: modified value of y

\{ 
  a=1; b=0; z=1; c=0;
  \textbf{while}(1) \{
    \text{if } (a == 0) \{
      \text{if } (b == 0) \{ 
        y=z+y; a =\neg a;
        b=\neg b; c=\neg c; \text{if } (\neg c) \text{ break}; \}
      \text{else} \{
        z=z+y; a=\neg a; b=\neg b; c=\neg c;
        \text{if } (\neg c) \text{ break}; \}
    \text{else if } (b == 0) \{ z=y \ll 2; a=\neg a; \}
    \text{else } \{ z=y \ll 3; a=\neg a; b=\neg b; \}
  \}\}
\}

What it does:

\texttt{y = y * 45}

We solve this using program synthesis.

Class of Programs: “Loop-Free”

- Programs implementing functions: $I \rightarrow O$

$$P(I):$$

\[
\begin{align*}
O_1 &= f_1 (V_1) \\
O_2 &= f_2 (V_2) \\
&\vdots \\
O_n &= f_n (V_n)
\end{align*}
\]

where

$f_1, f_2, \ldots, f_n$ are functions from a given component library

Functions could be if-then-else definitions and hence, the above represents any loop-free code.
Problem

Space of all possible programs
Each dot represents semantically unique program

Specification Oracle

I/O Oracle

I/O Examples that identify the correct program?
Program Learning as Set Cover

[Goldman & Kearns, 1995]

Space of all possible programs
Each dot represents
semantically unique program
Program Learning as Set Cover

Space of all possible programs
Each dot represents
semantically unique program
Program Learning as Set Cover

Space of all possible programs
Each dot represents semantically unique program

\((i_1, o_1)\)

Example \(\equiv\) Set of programs \textit{ruled out} by that example
Program Learning as Set Cover

Space of all possible programs
Each dot represents semantically unique program

\[(i_1, o_1) - E_1\]
\[(i_2, o_2) - E_2\]

\[
\ldots\ldots\ldots
\]

\[(i_n, o_n) - E_n\]

Theorem: [Goldman & Kearns, '95]
Smallest set of I/O examples to learn correct program

\[IS\]

Minimum size subset of \[
\{E_1, E_2, \ldots, E_n\}\] that covers all the incorrect programs
Program Learning as Set Cover

Space of all possible programs
Each dot represents semantically unique program

Practical challenge: can’t enumerate all inputs and find set $E_i$ for each

Smallest set of I/O examples to learn correct design

IS

Minimum size subset of $\{E_1, E_2, \ldots, E_n\}$ that cover all the incorrect programs
Program Learning as Set Cover

(i_1, o_1) - E_1
(i_2, o_2) - E_2

........

(i_n, o_n) - E_n

Space of all possible programs
Each dot represents semantically unique program

ONLINE set-cover:
In each step,
• choose some (i_j, o_j) pair
• eliminated incorrect programs E_j disclosed
Program Learning as Set Cover

Space of all possible programs
Each dot represents semantically unique program

\[(i_1, o_1) - E_1\]
\[(i_2, o_2) - E_2\]
\[\ldots\ldots\]
\[(i_n, o_n) - E_n\]

Our heuristic:

\[|E_j| \geq 1: \text{atleast one incorrect program identified}\]

Efficient model counting could yield better algorithm

ONLINE set-cover:

In each step,
- choose some \((i_j, o_j)\) pair
- eliminated incorrect programs \(E_j\) disclosed

In each step,
Our Approach: Learning based on Distinguishing Inputs

Space of all possible programs
Each dot represents semantically unique program

[Jha et al., ICSE’10]
Our Approach

Example I/O set $E := \{(i_1, o_1)\}$

Space of all possible programs
Example I/O set $E := \{(i_1, o_1)\}$

Space of all possible programs
Our Approach

Example I/O set $E := \{(i_1, o_1)\}$

Space of all possible programs
Our Approach

Example I/O set $E := \{(i_1, o_1)\}$

Space of all possible programs
Our Approach

Example I/O set $E := E \cup \{(i_2, o_2)\}$

Space of all possible programs
Example I/O set $E := E \cup \{(i_j, o_j)\}$

Space of all possible programs
Example I/O set $E := E \cup \{(i_k, o_k)\}$

Space of all possible programs
Example I/O set $E := E \cup \{(i_n, o_n)\}$

Space of all possible programs
Soundness

*Conditional on Validity of Structure Hypothesis*

- Library of components is sufficient?
  - YES: Correct design
  - NO: Infeasibility reported
    - YES: Infeasibility reported
    - NO: Incorrect design

Can invoke a Verification Oracle at the end and iterate
Other Important Details

- Novel encoding of the space of possible programs as an SMT formula
- Obtain a feasible program for given set of input/output pairs using SMT solving
- Obtain second feasible program and a distinguishing input using SMT solving

[Jha et al., ICSE ’10, PLDI ’11]
Malware Deobfuscation

- Conficker worm
- MyDoom and
- survey paper on obfuscations by Collberg et al*

Synthesized over 35 bit-manipulation programs from Hacker’s delight (the “Bible of bit-manipulation”).

Program length: 3-15

Number of input/output examples: 2 to 13.

Total runtime: < 1 second to 5 minutes.

Outline

- Program Synthesis from Input-Output Examples
- Synthesizing Environment Assumptions for Reactive Synthesis from LTL
- Conclusions and Future Directions
Synthesis from LTL

Environment Assumptions $\varphi_e$ Synthesis Tool $\varphi_s$ Realizable

System Requirements $\varphi_e \to \varphi_s$

Unrealizable

Often due to incomplete environment assumptions!
Satisfiability and Realizability

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

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

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

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

- **Assume:**
  - Given 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 from Counterstrategies

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 $\varphi_s$:
  - $G (r \rightarrow X F g)$
  - $G(c \lor g \rightarrow X \neg g)$

- A counter-trace:
  - $r$: 1 1 (1)
  - $c$: 1 1 (1)
CounterStrategy-Guided Environment Assumption Synthesis

Start

Formal Specification

Add

Mine Assumptions

Synthesis Tool

Compute Counterstrategy

Realizable

Done

Specification Templates

User Scenarios
Assumption Synthesis Algorithm

- **Specification Templates**: e.g. $G F$?
- **User Scenarios**:
- **Counter-strategy**

**Generate Candidate $\varphi$**
- e.g. $G F_p$

**Eliminate or Accept Candidate $\varphi$**
- Eliminate $\varphi$
- Accept $\varphi$

**Done** (check consistency with existing assumptions)
Assumption Synthesis Algorithm

1. Generate Candidate $\phi$
   - e.g. $GF$?

2. Invoke Model Checker: Does $E$ satisfy $\neg \phi$?
   - Yes: Accept $\phi$
   - No: Eliminate $\phi$

VERSION SPACES:
- Retain candidates consistent with examples
- Traverse weakest to strongest

User Scenarios
Specification Templates
Counter-strategy $E$
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.
- Occam’s Razor: Simple propositional structure $\rightarrow$ generalizable
- Selecting candidate assumptions: (next slide)
  - Weakest to Strongest, with heuristics
  - $G \ F \ b$ weaker than $G \ X \ b$ weaker than $G \ b$
Version Space Learning

Originally due to [Mitchell, 1978]

STRONGEST (most specific)

WEAKEST (most general)

\( p_1 \lor p_2 \)
\( p_1 \rightarrow X p_2 \)
\( p_{n-1} \lor p_n \)
Theoretical Results

- **Theorem 1: [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 2: [Soundness]** The procedure never adds inconsistent environment assumptions.
Example

- System $\varphi_s$:
  - $G(r \rightarrow X F g)$
  - $G(c \lor g \rightarrow X \neg g)$

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

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

- System $\varphi_s$:
  - $G(r \rightarrow X F g)$
  - $G(c \lor g \rightarrow X \neg g)$

- Environment $\varphi_e$:
  - $G(F \neg c)$

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
Experimental Results Summary

• 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:**
  \[ G (HLOCK[0] = 1 \rightarrow HBUSREQ[0] = 1) \]

  **Mined assumption:**
  \[ G (F \ HLOCK[0] = 0) \]

  Master 0 requests locked access to the bus
  Master 0 requests access to the bus
Switching Logic Synthesis for Hybrid Systems
Reflections on the Approach
Synthesizing Environment Assumptions for Reactive Synthesis from LTL
Conclusions and Future Directions
Induction + Deduction + Structure

- Structure Hypothesis encodes human insight about form of artifact to be synthesized
- Synthesis procedure combines inductive inference with deductive reasoning
- Many instances of this approach exist already

Key Points of Difference between instances:
1. **Search Strategy**: How to search the space of candidate concepts?
2. **Example Selection**: Which examples to learn from?
3. **Deductive Oracles**
## Comparison of Three Approaches

<table>
<thead>
<tr>
<th>Approach</th>
<th>Search Strategy for Candidate Programs</th>
<th>Example Selection Strategy</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Traditional” Counterexample-Guided Synthesis</td>
<td>Determined by Constraint Encoding &amp; Solver Heuristics</td>
<td>Counter-Example guided</td>
</tr>
<tr>
<td>Program Synthesis from I/O Examples</td>
<td>Determined by Constraint Encoding &amp; Solver Heuristics</td>
<td>Distinguishing Input guided</td>
</tr>
<tr>
<td>Env Assumption Generation for LTL Synthesis</td>
<td>Version Spaces</td>
<td>Counter-Strategy (counter-example) guided</td>
</tr>
</tbody>
</table>
Ongoing and Future Work

- Many Other Applications
  - Switching logic synthesis for safety and optimality [Jha et al '10,'11]
  - Fixed-point code from floating-point [Jha & S, ’11, ’13]
  - Synthesizing Abstractions for Hardware Verification [Brady et al., ’11]
  - Synthesizing Requirements for Industrial Control Models [Jin et al, ’13]
  - …

- Need better understanding of Inductive Synthesis
  - Explore range of learning models
  - Role of structure hypothesis
  - Control over deductive oracles (e.g., via problem encoding / search strategy in SAT/SMT)