Integrating Induction, Deduction and Structure for Synthesis

Sanjit A. Seshia

Associate Professor
EECS Department
UC Berkeley

Students: S. Jha, W. Li, L. Dworkin, D. Sadigh
Collaborators: A. Tiwari, S. Gulwani

NSF ExCAPE e-Seminar
November 5, 2012
Synthesis and Its Challenges

- Specification can be incomplete or difficult to use
- Environment model might be deficient
- High Complexity of synthesis

SYNTHESIZER

Specification \( \Phi \) → Environment Model E → System S (satisfying \( \Phi \) under E) / "Unrealizable"
Obfuscated code:

Input: \( y \)  Output: modified value of \( y \)

\[
\{ \ a=1; \ b=0; \ z=1; \ c=0; \ \\
\text{while}(1) \{ \\
\quad \text{if} \ (a == 0) \{ \\
\quad\quad \text{if} \ (b == 0) \{ \ y=z+y; \ b=\neg b; \ c=\neg c; \ \text{if} \ (\neg c) \ \text{break}; \ \} \\
\quad\quad \text{else} \{ \\
\quad\quad\quad \ y=z+y; \ a=\neg a; \ b=\neg b; \ c=\neg c; \\
\quad\quad\quad \ \text{if} \ (\neg c) \ \text{break}; \ \} \\
\quad \text{else if} \ (b == 0) \{ \ z=y << 2; \ a=\neg a; \} \\
\quad \text{else} \{ \ z=y << 3; \ a=\neg a; \ b=\neg b; \} \\
\} \}
\]

What it does:

\( y = y \times 45 \)

We solve this using program synthesis.

Synthesizing Switching Logic
(Hybrid Systems with Nonlinear Dynamics)

Synthesize switch between the gears such that
• Efficiency $\eta_i$ always exceeds 0.5 when $\omega \geq 5$
• Distance to be covered takes minimum time.

$\eta_i = 0.99e^{-(\omega - \omega_i)^2/64} + 0.01$

Reactive Synthesis from LTL

Environment Assumptions

System Requirements

\[ \varphi_e \rightarrow \varphi_s \]

CHALLENGE:
ENVIRONMENT SPECIFICATION

Often due to missing environment assumptions!

Strategies to Address Challenges

- **Human Input / Insight**
  - Hypothesis on the form of artifact to be synthesized
  - “Structure Hypothesis”
  - Examples:
    - Sketch [Bodik, Solar-Lezama, et al.]
    - Component library [Jha et al., …]

- **Induction + Deduction**
  - Induction: specific examples → general rules
    - Learning from examples
  - Deduction: general rules → specific conclusions
    - Logical inference and constraint solving
  - Purely deductive approach is inefficient / inapplicable
  - Purely inductive approach gives no guarantees
Approach: Sciduction

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

Outline

- Switching Logic Synthesis for Hybrid Systems
- Reflections on the Approach
- Synthesizing Environment Assumptions for Reactive Synthesis from LTL
- Conclusions and Future Directions
Hybrid Automata

Dynamics: Location $\rightarrow$ Ordinary Differential Equations (ODEs)

Switching Logic $\subseteq$ Location $\times$ Location $\rightarrow$ Predicates

A guard is taken as soon as it is enabled.

$x$ – room temp.
$T$ – coil temp.

OFF

$\dot{x} = -0.002(x - 16)$
$\dot{T} = 0$

$18.00 \leq x \leq 18.01 \land T = 20$

HEATING

$\dot{x} = -0.002(x - T')$
$\dot{T} = 0.1$

$18.00 \leq x \leq 18.26 \land T = 22$

COOLING

$\dot{x} = -0.002(x - T)$
$\dot{T} = -0.1$

$18.00 \leq x \leq 18.26 \land T = 22$

ON

$\dot{x} = -0.002(x - T)$
$\dot{T} = 0$

$19.94 < x < 19.95 \land T = 22$
Switching Logic Synthesis Problem

- **SAFETY**: Temperature $x$ must lie between 18 and 20 C.
  - In general: safe region is a hyperbox

- **OPTIMALITY**: Minimize switching between the modes
Our Solution

Structure Hypothesis:
Guards are Hyperboxes

Inductive Inference:
Learning Hyperbox from Examples

Deductive Procedure:
Numerical Simulation within a mode
Related Work

- Synthesis through Game Solving or Reachability [Asarin et al, IEEE 2000; Koo et al., HSCC 01]
- Reduce to finite state automata synthesis [Tabuada, SCL 08]
- Constraint-based approach [Taly et al. VMCAI 09]

Our approach is different in:
- Hypothesis about guard structure
- Use of hyperbox learning from examples
- Numerical simulation as a ‘decision procedure’
Overview of Approach

Fixpoint computation connects Learning and Numerical Simulation
Overview of Approach

Fixpoint computation connects Learning and Numerical Simulation
Overview of Approach

Fixpoint computation connects Learning and Numerical Simulation
Overview of Approach

Fixpoint computation connects Learning and Numerical Simulation
Overview of Approach

Fixpoint computation connects Learning and Numerical Simulation
Overview of Approach

Fixpoint computation connects Learning and Numerical Simulation
Overview of Approach

Fixpoint computation connects Learning and Numerical Simulation
Overview of Approach

Fixpoint computation connects Learning and Numerical Simulation
Overview of Approach

Fixpoint computation connects Learning and Numerical Simulation
Overview of Approach

Fixpoint computation connects Learning and Numerical Simulation

Unsafe Exit guard
Fixpoint Computation Constraints

\[ \text{Mode}_1, I \models \phi_S \text{W} (\bigvee_{k \in M} g_{1k}) \]

\[ \text{Mode}_i, \bigvee_{j \in M} g_{ji} \models \phi_S \text{W} (\bigvee_{k \in M} g_{ik}) \text{ for } i = 1..k \]

\[
\begin{align*}
\text{OFF (F)} & : \quad \begin{cases}
\dot{x} = -0.002(x - 16) \\
\dot{T} = 0
\end{cases} \\
\text{COOLING (C)} & : \quad \begin{cases}
\dot{x} = -0.002(x - T) \\
\dot{T} = -0.1
\end{cases} \\
\text{HEATING (H)} & : \quad \begin{cases}
\dot{x} = -0.002(x - T) \\
\dot{T} = 0.1
\end{cases} \\
\text{ON (N)} & : \quad \begin{cases}
\dot{x} = -0.002(x - T)
\end{cases}
\end{align*}
\]

\[
\begin{align*}
F, I & \models \phi_S \text{W} g_{FH} \\
F, g_{CF} & \models \phi_S \text{W} g_{FH} \\
H, g_{FH} & \models \phi_S \text{W} g_{HN} \\
N, g_{HN} & \models \phi_S \text{W} g_{NC} \\
C, g_{NC} & \models \phi_S \text{W} g_{CF}
\end{align*}
\]
Fixpoint Computation

- Initialize guards to over-approximation
- Pick a mode $M_i$
- Restrict entry guard $g_{ji}$ into $M_i$ such that all states reachable in $M_i$ are safe before some exit guards $g_{ik}$ becomes true.

Check if no changes – i.e. fixpoint reached

Safety property gives an over-approximation

Guards synthesized
Structure Hypothesis

- Guard region has a restricted geometric form
- Equivalently: guard is a restricted type of logical formula

- Hyperbox:
  Interval logic: $c_1 \leq x \leq c_2 \land c_3 \leq y \leq c_4$
Learning Problem

**Given:**
- An overapproximate hyperbox for guard $g_{ij}$
- Overapproximate hyperboxes for all other guards in/out of Mode $j$
- One safe switching state from Mode $i$ to Mode $j$
- Reachability oracle for single switching pts into Mode $j$

**Find:** Safe hyperbox restriction of $g_{ij}$
Optimal Teaching Sequence

Teaching problem:
If we knew what the guard was, what is the smallest set of examples to give to a learner to uniquely identify it?
Teaching an N-dimensional Box
Learning an N-dimensional Box

Binary search to find diagonally opposite corners of the box

Terminate when optimal teaching set is found
Complexity

Number of simulations needed:

- **Box (Conjunction of intervals):** \( \text{poly} (\log R, d) \)
- **Linearly independent linear inequalities:** \( \text{poly} (\log R, d, n) \)
- **Conjunction of arbitrary linear constraints:** Exponential in \( d \) (best known)
  
  (learning convex polygons is NP-hard – COLT94,98)

\( R \) is range of variables, \( d \) is number of variables (dimension), \( n \) is the number of constraints.
Synthesized Thermostat for Safety Requirement

Room temperature stays close to the maximum of safe range: 19.80 to 20 Celsius

The room could be allowed to cool longer without violating safety.
Dwell Time Requirement

- Lower and/or upper bound on time spent in a mode.

- Approximates performance properties: minimize switching, reduce stay in “costly” modes etc.

- Key idea:
  - Restrict entry guards to decrease dwell-time
  - Restrict exit guards to increase dwell-time
  - Recompute fixed point to enforce safety properties
Synthesized Thermostat
300 s min dwell time in off and on mode
Theorems

- **Soundness:** Synthesized Design satisfies given safety objectives.

- **Completeness:** No Design Synthesized if and only if No Design Feasible with given safety objectives.

Assuming:
(1) Ideal Reachability Oracle for ODEs
   - Necessary assumption
(2) Structure Hypothesis
   - if ANY switching logic exists, there exists one with hyperbox guards (on a known grid reflecting precision of sensing)
   - Assumption of “convenience”
### Case Studies

<table>
<thead>
<tr>
<th>Example</th>
<th># of Iterations</th>
<th>Runtime (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thermostat Controller</td>
<td></td>
<td></td>
</tr>
<tr>
<td>v1</td>
<td>5</td>
<td>21.6</td>
</tr>
<tr>
<td>v2 Case A</td>
<td>6</td>
<td>26.2</td>
</tr>
<tr>
<td>v2 Case B</td>
<td>6</td>
<td>25.7</td>
</tr>
<tr>
<td>TCAS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Case A</td>
<td>4</td>
<td>55.3</td>
</tr>
<tr>
<td>Case B</td>
<td>5</td>
<td>59.1</td>
</tr>
<tr>
<td>Automatic Transmission</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>83.6</td>
</tr>
<tr>
<td>Train Gate Controller</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Case A</td>
<td>3</td>
<td>22.5</td>
</tr>
<tr>
<td>Case B</td>
<td>4</td>
<td>28.3</td>
</tr>
</tbody>
</table>
Synthesis for Optimality

- Minimize long-run cost
  \[ \lim_{t \to \infty} \sum_{i=1}^{n} \frac{P_i}{R_i}(t) \]

**STRUCTURE HYPOTHESIS**
- Form of guard

**DEDUCTIVE ENGINE**
- Synthesizes for special case

**INDUCTIVE INFERENCE (LEARNING)**
- Generalizes from special case results

- Halfspace
- Numerical Simulation/Optimization
- PAC-learning of Halfspaces
Outline

- Switching Logic Synthesis for Hybrid Systems
- Reflections on the Approach
- Synthesizing Environment Assumptions for Reactive Synthesis from LTL
- Conclusions and Future Directions
Structure Hypothesis gives only Conditional Soundness/Completeness

- Synthesis procedure is **sound (complete)** assuming the structure hypothesis holds.
  - Switching logic synthesis: If ANY switching logic exists, there exists one with hyperbox guards (on a known grid)
  - In practice: how do we know if hyperbox guards exist?

- Can we restate this assumption in terms of the inputs to the synthesis process?
  
  YES. Sufficient conditions:
  - monotonicity of continuous dynamics in each mode
  - safe region is hyperbox on the known grid
Structure Hypothesis gives only Conditional Soundness/Completeness

- Another approach: Use a verifier to check that the synthesized system satisfies its specification
  - Verifier invoked in the loop or at the end
    - E.g. CEGIS loop used with Sketching
  - Requires efficient verification
Structure Hypothesis defines Concept Class for Learning

**INDUCTIVE ENGINE**

Hyperbox Learning (Active: selects examples)

**What is the label (+/-) for this example?**

**DEDUCTIVE ENGINE**

ODE Reachability Oracle (uses numerical simulation; special case of synthesis problem)

Labels (safe/unsafe)
Inductive Strategy at the Top-Level

Structure Hypothesis defines Concept (Program) Class

INDUCTIVE ENGINE

Learning from Examples

Queries

Examples

Labels for Examples

Candidate Program satisfying spec

DEDUCTIVE ENGINE

Logical Inference & Constraint Solving
Outline

- Switching Logic Synthesis for Hybrid Systems
- Reflections on the Approach
- Synthesizing Environment Assumptions for Reactive Synthesis from LTL
- Conclusions and Future Directions
A Personal Story of Synthesis from LTL

- Long, long ago…
- … we embarked on a “synthesis from LTL” project.
- 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 forever (~a week), no output
- Idea: synthesize individual modules (selections within contests, navigation between contests, etc.)
- Attempt 2, 3, …: Unrealizable! Too many environment assumptions needed (at interfaces between modules)
Problem

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
Our Contribution

Counterstrategy-guided synthesis of environment assumptions

- Demonstrated to generate useful/intuitive environment assumptions for digital circuits and robotic controllers
Sciduction 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.
Assumption Mining to Assist Synthesis

Start → Formal Specification

Add → Mine Assumptions

Synthesis Tool → Unrealizable

Unrealizable → Compute Counterstrategy

Realizable → Done

Specification Templates

A Few User Scenarios
Mining Algorithm

Specification Templates

User Scenarios

Counter-strategy

New Candidate $\phi'$

Candidate Spec $\phi$, e.g. $GFp$

Trace Verifier

$\neg \phi$

Model Checker

Pass?

Yes

No

Pass?

Yes

Add $\phi$

No
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
Version Space Learning

STRONGEST (most specific)

WEAKEST (most general)
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 $Gp$ and $Gq$ 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

- 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
Example

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

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

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

Realizable!
Result 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: \( G (HLOCK[0] = 1 \rightarrow HBUSREQ[0] = 1) \)
Mined assumption: \( G (\neg HLOCK[0] = 0) \)

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]
  - Computes a safety assumption that removes a minimal set of environment edges form the game graph
  - Computes a liveness assumption that puts fairness on the remaining environment edges
  - 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]
  - Counter-strategies and counter-traces as explanations for unrealizability
Discussion

- Specification mining can generate useful environment assumptions to cope with unrealizability.
- 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
Outline

- 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
- Synthesis procedure combines inductive inference with deductive reasoning
- Several demonstrations
  - 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]
Future Directions

- Exploring Structure Hypotheses
  - Need for flexibility in hypotheses
  - Requires flexible (extensible) learning algorithms that can accommodate changes to hypotheses

- Providing unconditional guarantees
  - Techniques to verify if structure hypothesis holds for a given input

- Investigate strategies to combine induction & deduction
  - Interplay between Teaching & Learning