Specification Mining: New Formalisms, Algorithms and Applications

Wenchao Li

Dissertation Talk
EECS Department, UC Berkeley

Thesis Committee: Sanjit A. Seshia (Advisor), Andreas Kuehlmann, Francesco Borrelli
Acknowledgement

• Prof. Sanjit A. Seshia
• Prof. Andreas Kuehlmann, Prof. Francesco Borrelli
• Dr. Alessandro Forin (MSR, Redmond)
• Dr. Natarajan Shankar, Dr. Shalini Ghosh (SRI International, Menlo Park)
• GigaScale Research Center
• TerraSwarm Research Center
• Colleagues in the DOP center
• Friends
• Nuo Zhang
Tiny bugs can have catastrophic consequences

Ubiquitous computing results in ubiquitous bugs

Formalization of requirement helps finding bugs, but is hard
Cost of Bugs

- **Human loss**: Pacemakers, Aircraft, Nuclear reactor controllers, Car engine management system, etc.
- **Financial Loss**: 1994 Pentium FDIV costs $475 million, Mars Rover, North America Blackout, etc.

Much of the challenge in bug finding lies in finding the specification that mechanized tools can use to find bugs

**Reality Check:**
- Writing assertion is a time-consuming manual process and is perceived as “difficult”.
- “During the first formal verification runs of a new hardware design, typically 20% of the formulas are found to be trivially valid.” [IBM Haifa]
Verification and Synthesis

Model $M$

An execution should never reach an error state.

Verification

Check $M \models \psi$

Synthesis

Find $M$ s.t. $M \models \psi$

Specification is arguably the most important step for formal verification and correct-by-construction synthesis
Verification is as Good as Specification

Specification:
1. Every request should be eventually granted.
2. Never reach an error state.

Need More Specifications

Assertion-based Verification

Coverage

Pass? Yes No

Debug

Cex

Poor Good Done
Verification is as **Bad** as Specification

- **Specification:**
  1. Every request should be eventually granted.
  2. Never reach an error state.

- **Missing Assumptions**

- **“Not a bug!”**

- **Need More Specifications**

**Assertion-based Verification**

- **Pass?**
  - **No**: Debug
  - **Yes**: Coverage

- **Coverage**
  - **Poor**
  - **Good**

- **Done**
Temporal specifications can be mined systematically both from observed and counteracting behaviors, and are useful for automating difficult tasks in verification and synthesis such as localizing bugs and finding missing assumptions.

**Formalisms**
- Basis Subtrace
- Version-Space Learning

**Algorithms**
- Automata-Based
- Sparse Coding
- Counterstrategy-Guided

**Applications**
- Bug Localization
- LTL Synthesis
- Human-in-the-Loop Controller
Linear Temporal Logic

Formal specification: behavior description supported by logic-based languages

\[ \psi ::= p \mid \neg \psi \mid \psi \lor \psi \mid X \psi \mid \psi U \psi \]

**Validity Examples:**

- **G p**
  
  \[ \begin{array}{ccccccc}
  p & \rightarrow & p & \rightarrow & p & \rightarrow & p & \rightarrow & p & \rightarrow & p & \rightarrow & p & \rightarrow & p & \rightarrow & \ldots \\
  \end{array} \]

- **F p**
  
  \[ \begin{array}{ccccccc}
  p & \rightarrow & p & \rightarrow & p & \rightarrow & p & \rightarrow & p & \rightarrow & p & \rightarrow & p & \rightarrow & \ldots \\
  \end{array} \]

- **G F p**
  
  \[ \begin{array}{ccccccc}
  p & \rightarrow & p & \rightarrow & p & \rightarrow & p & \rightarrow & p & \rightarrow & p & \rightarrow & \ldots \\
  \end{array} \]

- **G (p → F q)**
  
  \[ \begin{array}{ccccccc}
  p & \rightarrow & q & \rightarrow & p & \rightarrow & p & \rightarrow & q & \rightarrow & \ldots \\
  \end{array} \]

- **G (req → F grant):**
  Every request must be followed by a grant.
Example specification $\psi(a, b)$:

1. every $a$ is followed by a $b$ within 3 cycles;
2. every two $a$s are separated by at least 7 cycles.

$\Sigma = \{a, b\}$  $\Sigma' = \{\text{req, grant, reset}\}$

Find (all) mapping $\rho : \Sigma \rightarrow \Sigma'$, s.t. $\psi(\rho(a), \rho(b))$ is true w.r.t. some evidence.

\[\begin{array}{ccccc}
\text{req} & \text{reset} & \text{grant} & \text{req} & \text{grant} \\
\hline
\text{3 cycles} & \text{2 cycles} & \hline
\text{7 cycles} & \rho(a) = \text{req}, \rho(b) = \text{grant} & \\
\end{array}\]
Part I

Requirement Generation and Error Localization
**Requirement Generation**

- **Static:** Infer specification directly from the description of the design, e.g. synthesis of interface specification for Java classes [Alur et. al., 2005]

- **Dynamic:** Infer *likely specification* from simulation /execution traces, e.g. DAIKON [Ernst et. al., 2000]

**Automata-based** [DAC’10]  **Sparse Coding** [RV’12]
Specification Mining:

An *Automata-based Monitoring* Approach
Mining Temporal Properties

With a focus on hardware traces

Traces

Library of Temporal Patterns

Specification Mining Engine

User Event Definitions

Ranking Module

Mined Assertions

Most Relevant Assertions

[Li et al., Scalable Specification Mining for Verification and Diagnosis. DAC 2010]
All possible mappings

Challenges:
- $\Sigma'$ can be very large.
- $\text{Dim(Table)} \sim |\Sigma|$.

Solutions:
- Design $\psi$ s.t. evaluating transitions are sufficient.
- Small $\Sigma$ but use inference rules to merge $\psi$. 

$\psi(\rho(a), \rho(b))$
**Requirement Generation:**

eMIPS - 278 modules and more than 20,000 signals

| Design  | $|T|$  | $|T_\Delta^m|$ | $n_m$ | $|S|$  | $|S_{\text{merged}}|$ | Runtime (s) |
|---------|------|---------------|-------|-------|----------------------|-------------|
| eMIPS   | 5 mil| 5408          | 108   | 2079  | 1028                 | 51          |
| Router  | 0.23 mil | 12420     | 28    | 120   | 74                   | 13          |
| I2C     | 1.6 mil | 20904     | 33    | 389   | 308                  | 9           |
| CAN     | 26 mil  | 36100      | 175   | 3272  | 1356                 | 71          |

**Summary:**

- Industrial-size designs;
- Traces of millions of cycles;
- Mine relevant temporal properties efficiently.
Research Question

Can we use the many simple mined specifications to *localize* complex bugs?

Post-Si Challenges:
- Limited observability
- Long error detection latency
- Transient and hard-to-reproduce bugs

Fatal Error

010101010101
011010101010
01011111010
10101

Expensive: $1 million to redesign the masks [Ying et al., 2005]; 3:1 headcount for design vs. post-Si validation [Patra et al., 2007]; post-Si validation consumes 35% of chip development time on average [Abramovici et al., 2006]
Proposed Solution

Mine *distinguishing patterns* between good and bad traces over module interfaces.

- Normal Traces
- Error Trace

Assertion Miner

Diagnosis

\[ \Psi \ominus \Phi = \{p_1, \ldots, p_k\} \]

Candidate Fault Locations

Candidate Ranking

\{f_1, \ldots, f_k\}
### Error Localization:

CMP router; localize to within 15 cycles for transient faults

<table>
<thead>
<tr>
<th>Type of Fault</th>
<th>Fault Coverage %</th>
<th>Time Localization %</th>
<th>Module Localization %</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stuck-at</td>
<td>100</td>
<td>-</td>
<td>100</td>
</tr>
<tr>
<td>Erroneous Transition</td>
<td>100</td>
<td>-</td>
<td>100</td>
</tr>
<tr>
<td>Erroneous Assignment</td>
<td>100</td>
<td>-</td>
<td>57</td>
</tr>
<tr>
<td>Transient</td>
<td>100</td>
<td>81</td>
<td>56</td>
</tr>
</tbody>
</table>

**Summary:**
- eMIPS: effectively localize different design bugs.
- CMP router: effectively localize transient bugs also.
- Mining *simple distinguishing* patterns can help to localize *complex* bugs.
Research Question

Can we learn specifications w/o assuming forms?

Sparse Coding:
Sparsity helps to uncover latent structure
e.g. finding edge detectors in an unsupervised setting

Approximation:
≈ 0.8 * + 0.3 * + 0.5 *

Specification formalism: Express each subtrace as a
Boolean combination of a few “basis subtraces” – a
(sparsity-constrained) Boolean matrix factorization problem.

[Li and Seshia. Sparse Coding for Specification Mining and Error Localization. RV 2012]
Specification Mining:

A Sparse Coding Approach
Problem Formulation

Subtrace

Multiplication as “AND”
Addition as “OR”

columns are sparse

basis

coefficient

\[ t = \begin{bmatrix} 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 \end{bmatrix} \]

\[ \begin{bmatrix} 1 & 1 \\ 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix} \]
Given a data matrix $X \in B^{m \times n}$ and a positive integer $C$, the \textit{sparsity-constrained Boolean factorization problem} is to find $k$, $B = B^{m \times k}$ and $S = B^{k \times n}$ such that

$$X = B \circ S$$

and $\|S_{:,i}\|_1 \leq C, \forall i$

(and $\sum_i \sum_j S_{i,j}$ is maximized).
Observe that the data matrix $X$ can be viewed as the adjacency matrix for a bipartite graph.

**Idea:** factorization $\rightarrow$ biclique cover (biclique $\leftrightarrow$ basis subtrace)
Error Localization

- Error localization and explanation based on reconstruction:
  
  A subtrace has an error if it cannot be reconstructed from the basis subtraces

\[
\begin{array}{ccccccccc}
0 & 1 & 0 & 1 & 1 & 0 & \ldots & \ldots \\
1 & 0 & 0 & 1 & 1 & 1 & \ldots & \ldots \\
0 & 1 & 0 & 0 & 1 & 0 & \ldots & \ldots \\
1 & 0 & 0 & 1 & 0 & 1 & \ldots & \ldots \\
\end{array}
\]

\[
\begin{array}{l}
X_{,1} \\
X_{,2}
\end{array}
\]

- A subtrace is error-free if

\[
\|X_{,i} \oplus (B \circ S_{,i})\|_1 = 0
\]

Minimize \(\|X_{,i} \oplus (B \circ S_{,i})\|_1\)

Subject to \(\|S_{,i}\| \leq C\)
Experimental Results

- **Chip Multiprocessor Router:**
  - Observe 14 control signals
  - Subtrace width: 2 cycles
  - Learn the basis from a single error-free trace of 1000 cycles: 0.243 seconds to obtain 189 basis subtraces from 93 distinct subtraces

- **Error Localization:**
  - Inject a single bit flip at a random cycle for each of 99 error traces
  - Localize the error to the subtrace (out of 999) where it was injected

- **Comparisons:**
  - Baseline approach (1): hash all distinct subtraces – report error even before an error is injected for the 99 traces
  - Baseline approach (2): use unit basis – 0% localization
  - **Sparse Coding:** 55.6% localization

[Source: Daniel Holcomb]
Part I: Contributions

Automata-based: [Li et al., 2010]
- An efficient algorithm for mining \textit{temporal properties} from traces of digital designs.
- Effective algorithm for \textit{localizing bugs} in hardware using \textit{distinguishing patterns}.

Sparse Coding: [Li et al., 2012]
- A novel formalism of specification based on the notion of \textit{basis subtraces}.
- An \textit{unsupervised} algorithm for learning basis subtraces.
- An effective way of using basis subtraces to \textit{localize bugs}. 
Part II
Assumption Mining for LTL Synthesis
Temporal Logic Synthesis

Automatically construct an implementation that is guaranteed to satisfy its behavioral description.

ψ → Synthesis → M

Behavioral Description  Implementation

[Church, 1957] [Rabin, 1972] [Piterman and Pnueli, 2006]

- 2EXPTIME Complexity
- Need Complete Spec.

[LTL] [Pnueli, 1977]

Assumption Mining
"Writing a complete formal specification for the arbiter was not trivial. Many aspects of the arbiter are not defined in ARM’s standard.” [Bloem et al., 2007]
Assumption Mining:

A Counterstrategy-Guided Approach
GR(1) Specifications

\[ \psi^e \rightarrow \psi^s \]

Require \( \psi^l \) for \( l \in \{e, s\} \) to be conjunctions in the following forms:

- \( \psi^l_i \): a Boolean formula that characterizes the initial states.
- \( \psi^l_t \): a LTL formula that describes the transition, in the form \( G f \), where \( f \) is a Boolean combination of variables in \( X \cup Y \) and expressions \( X u \) where \( u \in X \) if \( l = e \) and \( u \in X \cup Y \) if \( l = s \).
- \( \psi^l_f \): a LTL formula that describes fairness, in the form \( G F f \), where \( f \) is a Boolean formula over variables in \( X \cup Y \).

**Advantage:** Can find an implementation in \( O((2^{|X|} + |Y|)^3) \) time.

[Piterman and Pnueli, 2006]
Given $\psi = \psi^e \rightarrow \psi^s$ with input $x$ and output $y$

$\psi_f^e = G (F (\neg x));$

$\psi_t^s = G ((\neg x) \rightarrow (\neg y));$

$\psi_f^s = G (F (y));$

$x : \ 0 \ 1 \ 0 \ 1 \ 0 \ 1 \ldots \ \text{Satisfiable}$

$y : \ 0 \ 1 \ 0 \ 1 \ 0 \ 1 \ldots$

$x : \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ldots \ \text{Unrealizable}$

$y : \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ldots$

**Decide if $\exists M \text{ s.t. } M \models \psi$: Game Solving**
A game structure \( \mathcal{G} \) is a tuple 
\((X, Y, Q, \theta, \rho^e, \rho^s, Win)\), where

- \( X \): a set of **input** variables controlled by \( e \).
- \( Y \): a set of **output** variables controlled by \( s \).
- \( Q \subseteq 2^X \times 2^Y \): state space.
- \( \theta \): a Boolean formula over \( X \cup Y \) that defines the **initial states**.
- \( \rho^e \subseteq Q \times 2^X \): environment transition relation.
- \( \rho^s \subseteq Q \times 2^X \times 2^Y \): system transition relation.
- \( Win \): winning condition of the game.
Given GR(1) specifications $\psi^e_i, \psi^s_i, \psi^e_t, \psi^s_t, \psi^e_f, \psi^s_f$,

- $\theta = \psi^e_i \land \psi^s_i$
- $\rho^e = \psi^e_t$ replacing $(X u)$ by $u'$
- $\rho^s = \psi^s_t$ replacing $(X u)$ by $u'$
- $Win$ is given by $\psi^e \rightarrow \psi^s$

\[ \bigwedge_i \psi^e_{f,i} \rightarrow \bigwedge_j \psi^s_{f,j} \]

**Remark:** The mapping also works for specifications given as deterministic Büchi automata.
GR(1) Synthesis ~ Games

Compute winning regions \( W^s \subseteq Q \) using a nested fixpoint formula.

\[
W^s \xrightarrow{\text{extract}} \text{strategy } S^s = (\Gamma^s, \gamma_0^s, \eta^s) \xrightarrow{\text{compute}} \text{circuit consistent with } S^s \quad (\text{if } Q_0 \subseteq W^s)
\]

Dually, compute winning regions \( W^e = Q \setminus W^s \) using fixpoint formula.

\[
W^e \xrightarrow{\text{extract}} \text{counterstrategy } S^e = (\Gamma^e, \gamma_0^e, \eta^e) \quad [\text{Könighofer et al., 2009}]
\]

**Problem:** \( \psi \) not realizable if \( Q_0 \cap W^e \neq \emptyset \)

\[
S^e \left\{ \begin{array}{l}
\Gamma^e = \mathcal{I} \times \mathcal{J} \\
\gamma_0^e \subseteq \Gamma^e \\
\eta^e \subseteq Q \times \Gamma \times 2^X \times \Gamma
\end{array} \right.
\]

**Key idea:** Mine additional assumption \( \phi \) to prohibit \( S^e \)
Given $\psi = \psi^e \rightarrow \psi^s$ with input $x$ and output $y$

$\psi^e_f = G (F (\neg x));$

$\psi^s_t = G ((\neg x) \rightarrow (\neg y));$

$\psi^s_f = G (F (y));$  

**Unrealizable**

**Counterstrategy**

$\models F (G (\neg x))$

Candidate assumption:

$\phi = \neg (F (G (\neg x))) = G (F (x))$

$\Rightarrow \psi_{new} = \phi \land \psi^e \rightarrow \psi^s$

**Realizable**
A counterstrategy graph \( G^c \) is a discrete transition system \((V, V_0 \subseteq V, T \subseteq V \times V)\), where

- \( V \subseteq Q \times \Gamma^e \): state space,
- \( V_0 = Q_0 \times \gamma_0^e \): initial states
- \( T = \eta^e \land \rho^s \): transition relation

**General Solution:**

Given a candidate assumption \( \phi \) and a counterstrategy graph \( G^c \),

\[
\psi_{new}^e := \phi \land \psi^e \enspace \text{if} \enspace \phi \land \psi^e \neq \text{false} \enspace \text{and} \enspace G^c \models \neg \phi \enspace (\text{model checking}).
\]

(a) **Consistency:** \( \phi \land \psi^e \neq \text{false} \)

(b) Done if \( \psi_{new}^e \rightarrow \psi^s \) is realizable;

(c) Otherwise, iterate with new candidate \( \phi_{new} \) and new \( G_{new}^c \).

**Question:** How to pick \( \phi \)?
Mining with Templates

What kind of assumptions?

- **Efficient**: In GR(1) to take advantage of $O(|Q|^3)$ algorithm.
- **User-friendly**: Simple formulas are easier to understand.
- **Representative**: Cover $\phi_i, \phi_t, \phi_f$.

Assumption Templates:

- $\phi_a$: $G\ F\ u$ or $G\ F\ (u \lor v)$ where $u$ and $v$ are literals over $X$.
- $\phi_b$: $G\ u$ or $G\ (u \lor v)$, where $u$ and $v$ are literals over $X$.
- $\phi_c$: $G\ (u \rightarrow (X\ v))$, where $u$ and $v$ are literals over $X$.

**Related**: Assumptions for LTL synthesis as a monolithic Büchi automaton. [Chatterjee et al., 2008]
Iterative Search

Problem 1: Redundant Checks

Problem 2: Restricted by Templates

Idea:
– Check one type of assumption in GR(1) at a time.
– Use a random determinization of $G^c$.

$$\psi = \psi^e \rightarrow \psi^s$$

$$\psi_{new} = \phi \land \psi^e \rightarrow \psi^s$$

Templates

GR(1) Synthesis

Mine Assumption $\phi$

Compute Counterstrategy $G^c$

Realizable $M_\psi$

Unrealizable
Search Optimizations

\[ \phi_1 = \neg (F \ u) \]

Terminal State: Safety Violation

\[ \phi_3 = \neg (F \ (v \wedge X \ u)) \]

SCC: Fairness Violation

\[ \phi_2 = \neg (F \ (G \ u)) \]

Boolean formula \( u \) over \( X \) and \( v \) over \( X \cup Y \)
Compute $\phi_1, \phi_2, \phi_3$ given symbolic representation of the counterstrategy graph $G^c = (V, V_0, T)$.

**Lemma 1:** $\phi_1 \land \phi_2$ is a minimal assumption in GR(1) syntax that removes the counterstrategy.

**Lemma 2:** The optimized algorithm produces a nontrivial $\phi$, i.e. $\phi \land \psi^e \neq \text{false}$.

**Theorem:** Given a satisfiable GR(1) specification $\psi = \psi^e \rightarrow \psi^s$ and a $G^c$ that represents all moves by the environment to force a violation of $\psi$, the optimized algorithm computes a nontrivial and minimal environment assumption $\phi$ in GR(1) such that $\phi \land \psi^e \rightarrow \psi^s$ is realizable.
Experimental Evaluation

Benchmarks:
- IBM Gen. Buffer, AMBA AHB Bus. [Bloem et al., 2007]
- Simple robotic controller.

Setup:
- Remove a single assumption from a realizable specification.
- Mine $\phi$ s.t. $\psi$ is realizable.

Result Highlight:
- Recover the missing assumption in most cases.
- Reasonable replacement?

AMBA AHB Example:

$\phi_{\text{original}} = G (HLOCK[0] \rightarrow HBUSREQ[0])$

$\phi_{\text{mined}} = G (F \neg HBUSREQ[0])$

HLOCK[0]: locked access
HBUSREQ[0]: bus request

[Li et al., Mining Assumptions for Synthesis. MEMOCODE 2011]
Summary of Contributions

• **First counterstrategy-guided synthesis framework**

\[ \psi = \psi^e \rightarrow \psi^s \]

\[ \psi_{new} = \phi \land \psi^e \rightarrow \psi^s \]

• An efficient algorithm with theoretical guarantees for assumption generation – a key problem in correct-by-construction synthesis from temporal logic.
Assumption Mining:

Synthesizing *Human-in-the-Loop* Controllers
Many *safety-critical* systems interact with humans. The correctness of such systems depend on both the correctness of *autonomous controller, actions of the human* and their interaction.
“Vehicles at this level of automation enable the driver to cede full control of all safety-critical functions under certain traffic or environmental conditions and in those conditions to rely heavily on the vehicle to monitor for changes in those conditions requiring transition back to driver control. The driver is expected to be available for occasional control, but with sufficiently comfortable transition time.”

Level 0: No Automation: Driver is in complete control

Level 1: Function Specific Automation
- Pre-charged Brakes

Level 2: Combined Function Automation
- Cruise Control
- Lane Keeping

Level 3: Limited Self Driving Automation

Level 4: Full Self Driving Automation

Research Question

When autonomous controller fails, can human safely take over control?

MIT Cornell Crash during DARPA Urban Challenge, 2007
**Human-in-the-Loop Controllers**

**Criteria:**

- **Monitoring**
  *Determine control switch based on monitored information*

- **Minimally Intervening**
  *Low probability of human control needed*

- **Prescient**
  *Notify danger ahead of time*

- **Conditionally Correct**
  *Safe until human takes over control*

---

**Composition of Auto-Controller, Human Operator and Advisory Controller**

Controller Synthesis

- System Specification $\psi^s$
- Environment Assumption $\psi^e$

Temporal Logic Synthesis

Realizable

Unrealizable?

- Compute Counterstrategy

Human-in-the-Loop Controller

Autonomous Controller

Approach:
- Mine transition assumptions $\phi_3$ to monitor
- Modify $G^c$ to account for human response time
- Assign probability and early intervention penalty to $G^c$
- Find s-t cut in the weighted $G^c$
Theoretical Guarantees

**Theorem:** Given a GR(1) specification $\psi$, and a response time parameter $T$, the algorithm is guaranteed to either produce a fully autonomous controller satisfying $\psi$, or a HuIL controller, modeled as a composition of an auto-controller, a human operator and an advisory controller that is monitoring, prescient (with parameter $T$), minimally intervening and conditionally correct.

**Assumptions:** System cannot fail within $T$ steps.

**Remark:** The human operator can be replaced by a controller that maintains critical functionalities.

A Car Following Example

Autonomous car: A  
Environment cars: B & C

Objective: A follows B, and when this is not achievable, switches control to the human driver with sufficient time for her to respond.

Follow := move to a square where A can still sense B

Given specs encoding movement rules and $T = 1$. 
Sening Regions:

Assume given a finite-state abstraction.

[Kloetzer and Belta, 2008][Bhatia, 2011] [Wolff et al., 2013]
Failure Scenario:

Step 1

Step 2
Mined Assumptions:

\[ \varphi_{env} = G \left( \left( p_A = 4 \right) \land \left( p_B = 6 \right) \land \left( p_c = 1 \right) \right) \rightarrow \]
\[ X \left( \left( p_B \neq 8 \right) \land \left( p_C \neq 5 \right) \right) \land \]
\[ G \left( \left( p_A = 4 \right) \land \left( p_B = 6 \right) \land \left( p_c = 1 \right) \right) \rightarrow \]
\[ X \left( \left( p_B \neq 6 \right) \land \left( p_C \neq 3 \right) \right) \land \]
\[ G \left( \left( p_A = 4 \right) \land \left( p_B = 6 \right) \land \left( p_c = 1 \right) \right) \rightarrow \]
\[ X \left( \left( p_B \neq 6 \right) \land \left( p_C \neq 5 \right) \right) \land \]
Part 2: Contributions

Assumption Mining: [Li et al., 2011]
• First counterstrategy-guided approach for synthesis from temporal logic.
• An efficient algorithm with theoretical guarantees for mining assumptions for GR(1) synthesis.

Human-in-the-Loop Controllers: [Li et al., 2013]
• A novel formalism of human-in-the-loop controllers.
• Identify criteria with application to driving automation.
• An algorithm for synthesizing human-in-the-loop controllers that automatically satisfy these criteria, from temporal logic specifications.
Specification Mining

Crowdsourced Game

Requirement for Verification
Error Localization

Mapping from Natural Language

Synthesis from Temporal Logic
Human-in-the-Loop Controller
Human Inputs:

CrowdMine: Gamification and Crowdsourcing
CrowdMine

Web-based Game Prototype

Two Sampled Subtraces

Selected Patterns → LTL Formulas → Model Checker

Counterexample

No Cex

Spec. Found

[URL: http://verifun.eecs.berkeley.edu/crowdmine2/]
Preliminary Results

- Circuit: I/O traces from a 2-input 2-output arbiter.

  ![Arbiter Diagram]

- Top ranked patterns:

  ![Top Ranked Patterns]

  “When $r_1$ is high and there is no competing $r_0$, $g_1$ is high at the same cycle.”
**Discussion**

Remark: w/o model checker in the loop.

- What are humans good at?
  - Visual recognition?
    Most frequently identified common patterns correspond to desired behaviors of the circuit.
  - Randomness?
    165 different patterns out of 283 hits (mostly EECS students)
    Top rank patterns have counts 31, 16 and 7.

- What problems do we crowdsource?
  Problems that require human input and insight, or ones that are hard to formally define.
  ➢ E.g. specification, diagnosis, repair.
  ➢ Not purely computationally intractable problems.

Related Work: FunSAT/Human EDA [DeOrio and Bertacco, DAC 2009]
Human Inputs:

Mapping *Natural Language* to Temporal Logic
Natural Language → LTL Specification

Result highlights:

• FAA-Isolette requirements from NL to LTL.
• Assumption mining discovered a missing assumption.

Conclusion

Formal specifications can be mined in a systematic way to improve the effectiveness of verification and synthesis.

**Formalisms**
- Basis Subtrace
- Version-Space Learning

**Algorithms**
- Automata-Based
- Sparse Coding
- Counterstrategy-Guided

**Applications**
- Bug Localization
- LTL Synthesis
- Human-in-the-Loop Controller
Future Work

• Combine automata-based and sparse coding-based approaches for mining specifications.
• Improve the scalability of the sparse coding-based approach.
• Mining assumptions in contract-based synthesis.
• Evaluate human-in-the-loop controller synthesis in real setting.
• Human studies of CrowdMine for large designs.
• More robust NL→LTL techniques.

Thank you!