Constraint Solving and Machine Learning for Program Verification and Synthesis

Hiroshi Unno
University of Tsukuba / RIKEN AIP

www.cs.tsukuba.ac.jp/~uhiro
Research Interests

• Formal specification, verification, and synthesis of (mainly but not limited to) higher-order functional programs by AI techniques such as constraint solving and machine learning

• Ongoing projects
  • Synthesis of High-Level Programs from Temporal and Relational Specifications (PI: Hiroshi Unno)
  • Program Verification Techniques for the AI Era (PI: Naoki Kobayashi)
  • AI Security and Privacy (PI: Jun Sakuma)
  • Metamathematics for Systems Design Project (PI: Ichiro Hasuo)
  • ...

2020/12/18  STAIR Lab Software Technology Seminar  2
This Talk

• Tutorial of program **verification** and **synthesis** based on **constraint solving** and **machine learning**
Background

• Our society heavily relies on computer systems
• Failure or malfunction of safety-critical systems would lead to human, social, economic, and environmental damage
  • 1985-1987 – Therac-25 medical accelerator delivered lethal radiation doses to patients
  • June 4, 1996 – Ariane 5 Flight 501 exploded
  • February, 2014 – 1.9 million Prius cars recalled
  • April, 2014 – OpenSSL Heartbleed vulnerability disclosed
  • June 17, 2016 – Ethereum DAO attacked, over $55M stolen

• Reliability assurance of safety-critical systems is crucial
Program Verification

• Formally prove or disprove a mathematical proposition $Q$: “The given program satisfies its formal specification”

• Great attentions from industry and academia
  • Microsoft’s SLAM & Everest projects, Facebook’s Infer, AWS
  • Turing awards to Hoare logic, temporal logic, model checking, ...

Program & Spec.

```ocaml
let rec mc x =
  if x > 100 then
    x - 10
  else
    mc (mc (x + 11))
in
let n = randi() in
if n ≤ 101 then
  assert (mc n = 91)
```

A proof of $Q$
orA counterexample to $Q$
(e.g., a program input leading to a spec. violation)
orUnknown
Program Synthesis

• Input an **incomplete program** and its **specification** \( \phi \), and output an **executable program** \( P \) that satisfies \( \phi \)
  
  • \( \phi \) specifies extensional (what \( P \) computes) and/or intentional (how \( P \) computes) behaviors of \( P \)

  • \( \phi \) is represented as a logical formula, input/output examples (e.g., MS Excel FlashFill), a natural language sentence, ...

A program satisfying the spec.

```plaintext
let rec mc x = 
  if x > 100 then
    x - 10
  else
    mc (mc (x + 11))
```

A proof of unrealizability

or

Unknown

Incomplete Program & Specification

```plaintext
let rec mc x = ?
let n = randi() in
if n ≤ 101 then
  assert (mc n = 91)
```
Enabling Technologies

• **Program logics** for mechanizing verification & synthesis
  - Hoare logic for proving Hoare triples \( \{P\}c\{Q\} \) meaning that:
    For any initial state \( \sigma \) that satisfies the precondition \( P \), if the execution of the program \( c \) under \( \sigma \) terminates, the postcondition \( Q \) is satisfied by the resulting state
  - Separation logic
  - Dependent refinement type system
  - Graded modal type system

• **Constraint solvers** for automating verification & synthesis
  - SAT solvers: satisfiability checker for propositional formulas
  - SMT solvers: satisfiability checker for predicate formulas over first-order theories on \textit{integers, reals, lists, arrays, …}

What about \textit{functions} (that represent inductive invariants, ranking functions, recurrent sets, Skolem functions, …)?
This Talk

• Tutorial of program verification and synthesis based on constraint solving and machine learning over functions

• First part: How to reduce program verification and synthesis to constraint solving

• Second part: How to solve constraints via integrated deductive and inductive reasoning
  • Deductive reasoning by theorem proving (e.g., SAT, SMT)
  • Inductive reasoning by machine learning (e.g., decision tree learning, reinforcement learning)
This Talk

• Tutorial of program *verification* and *synthesis* based on *constraint solving* and *machine learning* over *functions*

• **First part**: How to reduce program *verification* and *synthesis* to *constraint solving*

• **Second part**: How to solve constraints via integrated *deductive* and *inductive* reasoning
  
  • *Deductive* reasoning by *theorem proving* (e.g., SAT, SMT)
  
  • *Inductive* reasoning by *machine learning* (e.g., decision tree learning, reinforcement learning)
Program Verification via Constraint Solving

Target Program $P$ & Specification $\psi$

Constraint Generation

Constraints $C$ on Function Variables

Constraint Solving

$C$ is Sat ($P$ satisfies $\psi$), $C$ is Unsat ($P$ violates $\psi$), or Unknown

Verification Intermediary Independent of Particular Target and Method 😊
Program Verification via Constrained Horn Clauses (CHCs)

Target Program $P$ & Specification $\psi$

CHCs Constraints $C$ on Predicate Variables

Constraint Generation

- JayHorn for Java [Kahsai+ ’16]
- SeaHorn for C [Gurfinkel+ ’15]
- RCaml for OCaml [Unno+ ’09]

Constraint Solving

- SPACER [Komuravelli+ ’14]
- Hoice [Champion+ ’18]
- Eldarica [Hojjat+ ’18]

$C$ is Sat ($P$ satisfies $\psi$), $C$ is Unsat ($P$ violates $\psi$), or Unknown
**CHCs: Constrained Horn Clauses**

(see e.g., [Bjørner+ '15])

- A finite set $\mathcal{C}$ of **Horn-clauses** of either form:

  \[
  X_0(\overline{t}_0) \iff (X_1(\overline{t}_1) \land \cdots \land X_m(\overline{t}_m) \land \phi) \\
  \text{or } \bot \iff (X_1(\overline{t}_1) \land \cdots \land X_m(\overline{t}_m) \land \phi)
  \]

where $X_0, X_1, \ldots, X_m$ are predicate variables,
$\overline{t}_0, \ldots, \overline{t}_m$ are sequences of terms of a 1\textsuperscript{st}-order theory $T$,
$\phi$ is a formula of $T$ without predicate variables.

- $\mathcal{C}$ is **satisfiable** (modulo $T$) if there is an interpretation $\rho$ of predicate variables such that $\rho \models \land \mathcal{C}$
Example Program and *Partial Correctness* Specification:

**Pre-condition**

\[ \{ x = x_0 \} \]

\[ y = 0; \]

\[ \textbf{while } x \neq 0 \textbf{ do} \]

\[ y \leftarrow y + 1; \]

\[ x \leftarrow x - 1 \]

**Post-condition**

\[ \{ y = x_0 \} \]

If the initial state satisfies the **pre-condition** \( x = x_0 \) and **the loop terminates**

\[ \text{the post-condition } y = x_0 \text{ is satisfied by the resulting state} \]
Input:

\[
\begin{align*}
&\{x = x_0\} \\
y = 0; \\
&\text{while } x \neq 0 \text{ do} \\
&\quad y \leftarrow y + 1; \\
&\quad x \leftarrow x - 1 \\
&\text{done} \\
&\{y = x_0\}
\end{align*}
\]

Output \( C \): 

1. \( I(x_0, x, y) \leftarrow x = x_0 \land y = 0 \),
2. \( I(x_0, x - 1, y + 1) \leftarrow I(x_0, x, y) \land x \neq 0 \),
3. \( y = x_0 \leftarrow I(x_0, x, y) \land x = 0 \)

\( C \) is satisfiable, witnessed by a solution \( I(x_0, x, y) \equiv x_0 = x + y \)
Program Verification via Constrained Horn Clauses (CHCs)

Target Program \( P \) & Specification \( \psi \)

Limited to Linear-Time Safety Verification 😞

CHCs Constraints \( C \) on Predicate Variables

Constraint Generation

Constraint Solving

\( C \) is Sat (\( P \) satisfies \( \psi \)),
\( C \) is Unsat (\( P \) violates \( \psi \)),
or Unknown
Program Verification via Predicate Constraint Satisfaction [Satake+ '20]

Target Program $P$ & Specification $\psi$

Applicable to (Finitely-) Branching-Time Safety Verification 😊

pCSP Constraints $C$ on Predicate Variables

Constraint Generation

Constraint Solving

$C$ is Sat ($P$ satisfies $\psi$),
$C$ is Unsat ($P$ violates $\psi$),
or Unknown

PCSat [Satake+ ’20, Unno+ ’20]
Linear-Time vs. Branching-Time Verification of Non-det. Programs

• The target program $P$ may exhibit non-determinism caused by user input, network comm., scheduling, ...

• **Linear-time** verification concerns properties of the *execution traces* of $P$

• **Branching-time** verification concerns properties of the *computation tree* of $P$
  
  • Subsumes linear-time verification
  
  • Example: *Non-termination verification* of deciding whether there is an infinite execution of $P$ (cf. *termination verification* decides whether all execution of $P$ is finite)
**pCSP:** Predicate Constraint Satisfaction Problem [Satake+ ’20]

- A finite set $C$ of **clauses** of the form:

  $$\left( X_1(t_1) \lor \cdots \lor X_\ell(t_\ell) \right) \iff 
  \left( X_{\ell+1}(t_{\ell+1}) \land \cdots \land X_m(t_m) \land \phi \right)$$

  where $X_1, \ldots, X_m$ are predicate variables, $\tilde{t}_1, \ldots, \tilde{t}_m$ are sequences of terms of a 1\textsuperscript{st}-order theory $T$, $\phi$ is a formula of $T$ without predicate variables.

- $C$ is **satisfiable** (modulo $T$) if there is an interpretation $\rho$ of predicate variables such that $\rho \models \land C$

- $C$ is **CHCs** if $\ell \leq 1$ for all clause in $C$
If the initial state satisfies the pre-condition $x > 0$ there is an execution of the program such that the post-condition $\perp$ is satisfied when the while loop terminates.

Example Program and Specification:

Pre-condition: $\{x > 0\}$

while $x \neq 0$ do
  if read_bool() then
    $x \leftarrow x - 1$
  else
    $x \leftarrow x + 1$
  done

Post-condition: $\{\perp\}$ Contradiction

there is an execution of the program such that the post-condition $\perp$ is satisfied when the while loop terminates.
If the initial state satisfies the pre-condition \( x > 0 \), there is an execution of the program such that the while loop never terminates.
Input:
\{x > 0\}
while \(x \neq 0\) do
  if \(\text{read}_{-}\text{bool}()\) then
    \(x \leftarrow x - 1\)
  else
    \(x \leftarrow x + 1\)
done

Output \(C\):

1. \(I(x) \leftarrow x > 0\),
2. \(I(x - 1) \lor I(x + 1) \leftarrow I(x) \land x \neq 0\),
3. \(|\leftarrow I(x) \land x = 0\)

\(C\) is beyond CHCs!

\(C\) is satisfiable, witnessed by a solution \(I(x) \equiv x > 0\)
Program Verification via *Extended Predicate Constraint Satisfaction* [Unno+ ’20]

Target Program \( P \) & Specification \( \psi \)

Applicable to *(Infinitely-)*
Branching-Time Safety &
Liveness Verification 😊

\( \text{pfwCSP Constraints } C \text{ on } \text{Predicate Variables} \)

Constraint Generation

\( C \text{ is Sat } (P \text{ satisfies } \psi), \)
\( C \text{ is Unsat } (P \text{ violates } \psi), \)
or Unknown

Constraint Solving

PCSat [Satake+ ’20, Unno+ ’20]
Safety vs. Liveness Verification

• **Safety** is a class of properties of the form 
  “*something bad will never happen*”
  • Examples (absence of): assertion failure, division-by-zero, array boundary violation, ...

• **Liveness** is a class of properties of the form 
  “*something good will eventually happen*”
  • Examples: termination, deadlock freedom, ...
pfwCSP: Extension of pCSP with Functional and Well-founded Predicates [Unno+ ’20]
(cf. $\forall \exists$CHCs with dwf [Beyene+ ’13])

• A finite set $\mathcal{C}$ of pCSP clauses equipped with a map $\mathcal{K}$ from predicate variable $X$ in $\mathcal{C}$ to $\{\star, \lambda, \Downarrow\}$
  • $X$ is ordinary predicate if $\mathcal{K}(X) = \star$
  • $X$ is functional predicate if $\mathcal{K}(X) = \lambda$
  • $X$ is well-founded predicate if $\mathcal{K}(X) = \Downarrow$

• $\mathcal{C}$ is satisfiable (modulo $T$) if there is an interpretation $\rho$ of predicate variables such that
  • $\rho \models \land \mathcal{C}$
  • $\forall X. \mathcal{K}(X) = \lambda \Rightarrow \rho(X)$ characterizes a total function
  • $\forall X. \mathcal{K}(X) = \Downarrow \Rightarrow \rho(X)$ represents a well-founded relation
Example Program and Specification:

Pre-condition: \( \{ x > 0 \} \)

Non-det. integer

Post-condition: \( \{ \perp \} \)

Contradiction

While loop:

```c
while x > 0 do
    x ← read_int()³ - x
done
```

If the initial state satisfies the pre-condition \( x > 0 \), there is an execution of the program such that the while loop never terminates.

2020/12/18 STAIR Lab Software Technology Seminar 25
Input:
\{x > 0\}
while x > 0 do
  x ← read_int()³ − x
done
\{⊥\}

Output \( C \):

1. \( I(x) \leftarrow x > 0, \)
2. \( (∃r. I(r − x)) \leftarrow I(x) \land x > 0, \)
   \( \perp \leftarrow I(x) \land x ≤ 0 \)

\( C \) is beyond pCSP but can be encoded in pfwCSP using a functional pred. var. that characterizes a Skolem function for \( r \).
Input:
\{x > 0\}

while \(x > 0\) do
\(x \leftarrow \text{read\_int()}\) \(\exists - x\)
done
\(\bot\)

Output \(C\):
① \(I(x) \iff x > 0\),
② \(I(r - x) \iff S_\lambda(x, r)\)
\(\land I(x) \land x > 0\),
③ \(\bot \iff I(x) \land x \leq 0\)

\(C\) is satisfiable, witnessed by a solution
\(I(x) \equiv x > 0, S_\lambda(x, r) \equiv r = x + 1\)
Example Program and Total Correctness Specification:

Pre-condition:

\[ x \neq 0 \]

while \( x \neq 0 \) do

if \( x > 0 \) then

\( x \leftarrow x - 1 \)

else

\( x \leftarrow x + 1 \)

done

Post-condition:

\[ [\top] \]

Tautology

If the initial state satisfies the pre-condition \( x \neq 0 \), the loop always terminates and the post-condition \( \top \) is satisfied by the resulting state.
Input:

\[ x \neq 0 \]

while \( x \neq 0 \) do

if \( x > 0 \) then

\( x \leftarrow x - 1 \)

else

\( x \leftarrow x + 1 \)

done

Output \( C \):

① \( I(x) \leftarrow x = 0 \),

② \( I(x - 1) \leftarrow I(x) \land x > 0 \),

③ \( I(x + 1) \leftarrow I(x) \land x < 0 \),

④ \( T_{\uparrow}(x, x - 1) \leftarrow I(x) \land x > 0 \),

⑤ \( T_{\downarrow}(x, x + 1) \leftarrow I(x) \land x < 0 \),

\( C \) is satisfiable, witnessed by a solution

\[ I(x) \equiv \top, T_{\uparrow}(x, x') \equiv |x| > |x'| \geq 0 \]
Further Applications of pfwCSP

• Refinement type inference [Unno+ ’09,’13,’18, Nanjo’18, Katsura+ ’20]
• Validity checking of fixpoint logic formulas
• LTL, CTL, CTL*, modal-mu calculus model checking
• Infinite-state infinite-duration game solving
• Bisimulation and bisimilarity verification
• Hyperproperties verification
• Program synthesis

• ...

(see [Unno+ ’20] and upcoming papers)
Program Synthesis via Constraint Solving

Language $\mathcal{L}$ & Specification $\psi$

Constraint Generation

Constraints $\mathcal{C}$ on Function Variables

Constraint Solving

$\mathcal{C}$ is Sat \(\text{(some } P \in \mathcal{L} \text{ satisfies } \psi)\),

$\mathcal{C}$ is Unsat \(\text{(all } P \in \mathcal{L} \text{ violates } \psi)\),

or Unknown

Synthesis Intermediary Independent of Particular Target and Method 😊
Program Synthesis via Syntax-Guided Synthesis (SyGuS)

Language $\mathcal{L}$ & Specification $\psi$

Constraint Generation

SyGuS Constraints $\mathcal{C}$ on Function Variables

Constraint Solving

$\mathcal{C}$ is Sat (some $P \in \mathcal{L}$ satisfies $\psi$),
$\mathcal{C}$ is Unsat (all $P \in \mathcal{L}$ violates $\psi$),
or Unknown

CVC4 [Reynolds+ ’15,’19]
DryadSynth [Huang+ ’20]
PCSat [Satake+ ’20, Unno+ ’20]
SyGuS: Syntax-Guided Synthesis [Alur+ ’15]

• Fix a first-order background theory $T$ such as:
  • Linear integer arithmetic (LIA)
  • Strings (for FlashFill benchmarks)
  • Bit-vectors (for Hackers' Delight benchmarks)

• Given
  • Specification: $T$-formula $\phi$ over a function variable $f$
  • Language: context-free grammar $G$ characterizing the set $\mathcal{L}(G)$ of allowed $T$-terms

• Find a term $t \in \mathcal{L}(G)$ such that $\models [t/f]\phi$
Example LIA SyGuS Constraints $\mathcal{C}$:

- Language: $G$ that generates any term of LIA

- Specification: $\phi \equiv (f(x, y) \geq x \land f(x, y) \geq y \land (f(x, y) = x \lor f(x, y) = y))$

$\mathcal{C}$ is satisfied by $f(x, y) \equiv \text{if } x > y \text{ then } x \text{ else } y$

$\mathcal{C}$ can be reduced to the pfwCSP:

$r \geq x \land r \geq y \land (r = x \lor r = y) \iff F_\lambda(x, y, r)$

In general, SyGuS constraints $\mathcal{C}$ can be converted to a pfwCSP using a predicate that characterizes $\mathcal{L}(G)$
This Talk

• Tutorial of program verification and synthesis based on constraint solving and machine learning over functions

• First part: How to reduce program verification and synthesis to constraint solving

• Second part: How to solve constraints via integrated deductive and inductive reasoning
  • Deductive reasoning by theorem proving (e.g., SAT, SMT)
  • Inductive reasoning by machine learning (e.g., decision tree learning, reinforcement learning)
Program Verification and Synthesis via Predicate Constraint Satisfaction

Target Program $P$ & Specification $\psi$

Language $\mathcal{L}$ & Specification $\psi$

Constraint Generation

$pfwCSP$ Constraints $\mathcal{C}$ on Predicate Variables

$\mathcal{C}$ is Sat, $\mathcal{C}$ is Unsat, or Unknown
Challenges in Constraint Solving

• Undecidable in general even for decidable theories
• The search space of solutions is often very large (or unbounded), high-dimensional, and non-smooth

To address these challenges, researchers are integrating deductive & inductive reasoning techniques within the framework of CounterExample Guided Inductive Synthesis (CEGIS) [Solar-Lezama+ '06]
CounterExample Guided Inductive Synthesis (CEGIS)

• Iteratively accumulate example instances $\mathcal{E}$ of the given $\mathcal{C}$ through the two phases for each iteration:
  
  • *Synthesis Phase by Learner*
    • Find a candidate solution $\rho$ that satisfies $\mathcal{E}$

  • *Validation Phase by Teacher*
    • Check if the candidate $\rho$ also satisfies $\mathcal{C}$ (with an SMT solver)
      • If yes, return $\rho$ as a genuine solution of $\mathcal{C}$
      • If no, repeat the procedure with new example instances witnessing non-satisfaction of $\mathcal{C}$ by $\rho$ added
Example Run of CEGIS

Learner
Example Instances $\mathcal{E}$:

$\emptyset$

Starting from the empty set ($\mathcal{C}$ is a black box)

Teacher
Constraints $\mathcal{C}$:

- $I(x) \iff x > 0$
- $I(x - 1) \lor I(x + 1) \iff I(x) \land x \neq 0$
- $\bot \iff I(x) \land x = 0$

Is the candidate $\{I(x) \mapsto T\}$ genuine?
Example Run of CEGIS

**Learner**

Example Instances $\mathcal{E}$:

\[ \perp \iff I(0) \wedge 0 = 0 \]

**Teacher**

Constraints $\mathcal{C}$:

- $I(x) \iff x > 0$
- $I(x - 1) \vee I(x + 1) \iff I(x) \wedge x \neq 0$
- $\perp \iff I(x) \wedge x = 0$

No. \{\(I(x) \mapsto T\}\} is not.
The 3\(^{rd}\) clause is violated when \(x = 0\).
Example Run of CEGIS

Learner
Example Instances $\mathcal{E}$:

$\neg I(0)$

Teacher
Constraints $\mathcal{C}$:

- $I(x) \iff x > 0$
- $I(x - 1) \lor I(x + 1) \iff I(x) \land x \neq 0$
- $\bot \iff I(x) \land x = 0$

Is the cand. $\{I(x) \leftrightarrow x < 0\}$ genuine?
Example Run of CEGIS

Learner
Example Instances $\mathcal{E}$:

- $\neg I(0)$
- $I(1) \iff 1 > 0$

Teacher
Constraints $\mathcal{C}$:

- $I(x) \iff x > 0$
- $I(x - 1) \lor I(x + 1) \iff I(x) \land x \neq 0$
- $\bot \iff I(x) \land x = 0$

No. $\{I(x) \iff x < 0\}$ is not. The 1st clause is violated when $x = 1$
Example Run of CEGIS

Learner
Example Instances $\mathcal{E}$:

- $\neg I(0)$
- $I(1)$

Teacher
Constraints $\mathcal{C}$:

- $I(x) \iff x > 0$
- $I(x - 1) \lor I(x + 1) \iff I(x) \land x \neq 0$
- $\bot \iff I(x) \land x = 0$

Is the cand. $\{I(x) \iff x = 1\}$ genuine?
Example Run of CEGIS

Learner
Example Instances $\mathcal{E}$:

- $I(0) \lor I(2) \iff I(1)$
- $I(1)$

Teacher
Constraints $\mathcal{C}$:

- $I(x) \iff x > 0$
- $I(x - 1) \lor I(x + 1) \iff I(x) \land x \neq 0$
- $\bot \iff I(x) \land x = 0$

No. $\{I(x) \iff x = 1\}$ is not.
The 2nd clause is violated when $x = 1$
Example Run of CEGIS

Learner
Example Instances $\mathcal{E}$:

- $\neg I(0)$
- $I(0) \lor I(2) \iff I(1)$
- $I(1)$

Teacher
Constraints $\mathcal{C}$:

- $I(x) \iff x > 0$
- $I(x - 1) \lor I(x + 1) \iff I(x) \land x \neq 0$
- $\bot \iff I(x) \land x = 0$

Is the cand. $\{I(x) \leftrightarrow x \geq 1\}$ genuine?
Example Run of CEGIS

**Learner**

Example Instances $\mathcal{E}$:

$\neg I(0)$

$I(0) \vee I(2) \iff I(1)$

$I(1)$

**Teacher**

Constraints $\mathcal{C}$:

- $I(x) \iff x > 0$
- $I(x - 1) \vee I(x + 1) \iff I(x) \land x \neq 0$
- $\bot \iff I(x) \land x = 0$

Yes. $\{I(x) \iff x \geq 1\}$ satisfies $\mathcal{C}$!
Is CEGIS just (Online) Supervised Learning of Classification?

• Similarities
  • Learner trains a model to fit examples $\mathcal{E}$ and obtain $\rho$
  • Teacher requires $\rho$ to generalize to $\mathcal{C}$ ($\rho$ shouldn’t overfit $\mathcal{E}$)

• Differences
  • $\mathcal{E}$ is usually assumed to have no noise & $\mathcal{C}$ is hard constraints
  • $\rho$ is required to exactly satisfy $\mathcal{E}$ (or has no chance to satisfy $\mathcal{C}$)
  • $\rho$ should be efficiently handled by Teacher (i.e., an SMT solver)
  • Sampling of $\mathcal{E}$ from $\mathcal{C}$ is not i.i.d (depends on $\rho$ and Teacher)
  • $\mathcal{E}$ may contain not only positive/negative examples but also arbitrary clause ones (cf. constrained semi-supervised learning)

Despite the differences, machine learning techniques turned out to be quite useful!
Machine Learning for CEGIS

• Adapt ML models and algorithms to implement Learner
  • Piecewise linear classifiers [Sharma+ ’13a, Garg+ ’14, Unno+ ’20]
  • Decision trees [Krishna+ ’15, Garg+ ’16, Champion+ ’18, Ezudheen+ ’18, Zhu+ ’18]
  • Neural networks [Chang+ ’19, Zhao+ ’20, Abate+ ’21]
  • Greedy set covering w/ logic minimization [Padhi+ ’16, Sharma+ ’13b]
  • Metropolis Hastings MCMC sampler [Sharma+ ’14]
  • Probabilistic inference, survey propagation [Satake+ ’20]
  • Ensemble learning [Padhi+ ’20]

• Learning to learn
  • Reinforcement learning of NNs to generate candidates [Si ’18]
  • Reinforcement learning of strategy to adjust classification models used by Learner (joint work w/ Tsukada, Sekiyama, Suenaga)
SMT-based Piecewise Linear Classification (aka Template-based Synthesis)

1. Prepare a solution template with unknown coefficients,
2. Generate constraints on them, and
3. Solve them using an SMT solver

Examples: \( \mathcal{E} \equiv \{ I(0), I(0) \Rightarrow I(1), \neg I(-1) \} \)

Solution Template: \( I(x) \Leftrightarrow c_1 \cdot x + c_2 \geq 0 \)

Coeff. Constraints: \( \{ c_2 \geq 0, c_2 \geq 0 \Rightarrow c_1 + c_2 \geq 0, -c_1 + c_2 < 0 \} \)

Satisfying Assignment: \( \{ c_1 \leftrightarrow 1, c_2 \leftrightarrow 0 \} \)

A Candidate Solution: \( \rho \equiv \{ I(x) \leftrightarrow x \geq 0 \} \)
Decision Tree Learning

1. Consistently label atoms in $\mathcal{E}$ with $+/−$ using a SAT solver
2. Generate a set $Q$ of predicates used in classification
3. Classify atoms in $\mathcal{E}$ with $Q$ using a decision tree learner

Examples: $\mathcal{E} \equiv \{I(0), I(0) \Rightarrow I(1), \neg I(-1)\}$

Labeling: $\{I(0) \mapsto +, I(1) \mapsto +, I(-1) \mapsto -\}$

Predicates: $Q \equiv \{x \geq 0, x \leq 0, x \geq 1, x \geq -1, x \leq 1, x \leq -1\}$

Classifier: $\rho \equiv \{I(x) \mapsto x \geq 0\}$
Template-based Synthesis vs Decision Tree Learning

- Template-based Synthesis (TB)
  - 😞 Fixes the *shape* of solution (updated upon failure)
  - 😊 Flexibly find necessary *predicates* via SMT solving
  - 😊 Atoms in $\mathcal{E}$ are consistently *labeled* using $\mathcal{E}$ as an SMT formula

- Decision Tree Learning (DT)
  - 😞 Fixes the *predicates* of solution (updated upon failure)
  - 😊 Flexibly adjust the *shape* based on information gain
  - 😞 Atoms are consistently *labeled* using $\mathcal{E}$ as a SAT formula

- Evaluation on SyGuS-Comp’19 Inv track XC benchmarks
  - TB solved 228 instances (out of 276) and DT solved 180 instances
Future Research Directions

• Efficient synthesis of *complex and large functions* from *complex and large constraints*
  • (Co)Inductive functions
  • Functions over (linked, (co)algebraic, array) data structures
  • Improve labeling, sampling and filtering of examples, and generation and ranking of predicates

• Convergence theory of CEGIS

• More applications
Summary

• Various program verification and synthesis problems can be reduced to constraint solving problems
  • The separation of constraint generation and solving facilitate tool development

• CEGIS-based constraint solving integrates deductive and inductive reasoning to address challenges
  • Deductive reasoning by theorem proving (e.g., SAT, SMT)
  • Inductive reasoning by machine learning (e.g., decision tree learning, reinforcement learning)