Probabilistic Inference for Predicate Constraint Satisfaction

Hiroshi Unno (University of Tsukuba)
Joint Work with Yuki Satake and Hinata Yanagi
Program Verification via Predicate Constraint Satisfaction

Target Program $P$ & Specification $\psi$

Constraint Generation

Constraints $C$ on Predicate Variables

Constraint Solving

$C$ is Sat ($P$ satisfies $\psi$),
$C$ is Unsat ($P$ violates $\psi$),
or Unknown
Previous Work: Program Verification via Constrained Horn Clauses (CHCs) [Bjørner+ ’15]

Limited to Linear-Time Safety Verification 😞

CHCs Constraints $C$ on Predicate Variables

Verification Intermediary Independent of Particular Target and Method 😊

Target Program $P$ & Specification $\psi$

Constraint Generation

Constraint Solving

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

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

SPACER [Komuravelli+ ’14]
Hoice [Champion+ ’18]
Eldarica [Hojjat+ ’18]
This Work: Program Verification via Predicate Constraint Satisfaction Problem (pCSP)

Target Program $P$ & Specification $\psi$

- Support Branching-Time Safety Verification 😊
- New method for Looping & Recursive Programs

pCSP Constraints $C$ on Predicate Variables

- Verification Intermediary Independent of Particular Target and Method 😊
- New method based on Probabilistic Inference

$C$ is Sat ($P$ satisfies $\psi$), $C$ is Unsat ($P$ violates $\psi$), or Unknown
This Work: Program Verification via Predicate Constraint Satisfaction Problem (pCSP)

Target Program $P$ & Specification $\psi$

Constraint Generation

pCSP Constraints $C$ on Predicate Variables

Verification Intermediary Independent of Particular Target and Method 😊

Constraint Solving

$C$ is $\text{Sat}$ ($P$ satisfies $\psi$),
$C$ is $\text{Unsat}$ ($P$ violates $\psi$),
or Unknown
Predicate Constraint Satisfaction Problem (pCSP)

- A finite set $\mathcal{C}$ of clauses of the form:
  $$X_1(t_1) \lor \cdots \lor X_\ell(t_\ell) \lor \phi \lor$$
  $$\neg X_{\ell+1}(t_{\ell+1}) \lor \cdots \lor \neg X_m(t_m)$$

  where $X_1, \ldots, X_m$ are predicate variables,
  $t_1, \ldots, t_m$ are sequences of terms,
  $\phi$ is a first-order formula w/o predicate variables.
Predicate Constraint Satisfaction Problem (pCSP)

• A finite set $\mathcal{C}$ of clauses of the form:

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

where $X_1, \ldots, X_m$ are predicate variables,
$t_1, \ldots, t_m$ are sequences of terms,
$\phi$ is a first-order formula w/o predicate variables.

• $\mathcal{C}$ is satisfiable (modulo first-order theories) if there is an interpretation $\rho$ of predicate variables such that $\rho \models \land \mathcal{C}$

• $\mathcal{C}$ is called CHCs [Bjørner+ ’15] if $\ell \leq 1$ for each clause in $\mathcal{C}$
This Work: Program Verification via Predicate Constraint Satisfaction Problem (pCSP)

Target Program $P$ & Specification $\psi$

Support Branching-Time Safety Verification 😊

$pCSP$ Constraints $C$ on Predicate Variables

Constraint Generation

Constraint Solving

$C$ is $\text{Sat}$ ($P$ satisfies $\psi$), $C$ is $\text{Unsat}$ ($P$ violates $\psi$), or Unknown
Branching-Time Safety Verification of Finitely-Branching Programs

- **Safety** is a class of properties of the form “something bad will never happen”

- **Branching-time** verification concerns properties of the computation tree of the given program that may exhibit non-deterministic behavior (cf. linear-time verification concerns properties of execution traces)

- Subsumes **non-termination verification** of deciding whether there is a non-terminating execution
This Work: Program Verification via Predicate Constraint Satisfaction Problem (pCSP)

Target Program $P$ & Specification $\psi$

Constraint Generation

New method for Looping & Recursive Programs

pCSP Constraints $C$ on Predicate Variables

Constraint Solving

$C$ is **Sat** ($P$ satisfies $\psi$),
$C$ is **Unsat** ($P$ violates $\psi$),
or **Unknown**
Example Program and Specification:

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

while \( x \neq 0 \) do
  if \( \text{read_bool()} \) then
    \( x \leftarrow x - 1 \)
  else
    \( x \leftarrow x + 1 \)

done

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

If the initial state satisfies the pre-condition \( x > 0 \) there is an execution of the program such that the post-condition \( \bot \) 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.

Example Program and Specification:

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

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

done

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

There is an execution of the program such that the while loop never terminates.
Input:
\{x > 0\}
while \( x \neq 0 \) do
  if read_bool() then
    \( x \leftarrow x - 1 \)
  else
    \( x \leftarrow x + 1 \)
done
\{\bot\}

Output \( C \):
\[
\begin{align*}
\text{(1) } & \quad I(x) \leftarrow x > 0, \\
\text{(2) } & \quad I(x - 1) \lor I(x + 1) \leftarrow I(x) \land x \neq 0, \\
\text{(3) } & \quad \bot \leftarrow I(x) \land x = 0
\end{align*}
\]

\( C \) is beyond CHCs!
\( C \) is satisfiable, witnessed by a solution \( I(x) \equiv x > 0 \)

represents a loop invariant preserved by some execution
This Work: Program Verification via Predicate Constraint Satisfaction Problem (pCSP)

Target Program $P$ & Specification $\psi$

Constraint Generation

pCSP Constraints $C$ on Predicate Variables

Constraint Solving

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

New method based on Probabilistic Inference
Challenges in pCSP Solving

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

We address these challenges by a novel combination of probabilistic inference with 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**
    
    • Enumerate candidate solutions $\rho_1, \ldots, \rho_n$ that satisfy $\mathcal{E}$

  • **Validation Phase**
    
    • Check if there is a candidate $\rho_i$ that also satisfies $\mathcal{C}$
      
      • If yes, return $\rho_i$ as a solution of $\mathcal{C}$
      
      • If no, repeat the procedure with new example instances witnessing non-satisfaction of $\mathcal{C}$ by $\rho_1, \ldots, \rho_n$ added
Example Run of CEGIS

Synthesizer

Example Instances $\mathcal{E}$:

$\emptyset$

Starting from the empty set

Validator

pCSP 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 one of the following candidates genuine? 

$\{I(x) \iff T\}$, ...
Example Run of CEGIS

Synthesizer
Example Instances $\mathcal{E}$:

$$\bot \iff I(0) \land 0 = 0$$

Validator
pCSP 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. $\{I(x) \rightarrow T\}$ is not.
The 3rd clause is violated when $x = 0$
Example Run of CEGIS

Synthesizer
Example Instances $\mathcal{E}$:

$$\neg I(0)$$

Validator
pCSP 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 one of the following candidates genuine?

$\{I(x) \leftrightarrow x < 0\}$, ...
Example Run of CEGIS

Synthesizer
Example Instances $\mathcal{E}$:

\[-I(0)\]

$I(1) \iff 1 > 0$

Validator
pCSP Constraints $\mathcal{C}$:

\[\begin{align*}
&\cdot I(x) \iff x > 0 \\
&\cdot I(x - 1) \vee I(x + 1) \iff I(x) \land x \neq 0 \\
&\cdot \bot \iff I(x) \land x = 0
\end{align*}\]

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

Synthesizer
Example Instances $\mathcal{E}$:

\[ \neg I(0) \]

\[ I(1) \]

Validator
pCSP 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 one of the following candidates genuine?

\[ \{I(x) \iff x \geq 1\}, \ldots \]
Example Run of CEGIS

Synthesizer
Example Instances $\mathcal{E}$:

$\neg I(0)$

$I(1)$

Validator
pCSP Constraints $\mathcal{C}$:

$\cdot I(x) \iff x > 0$

$\cdot I(x - 1) \lor I(x + 1) \iff I(x) \land x \neq 0$

$\cdot \bot \iff I(x) \land x = 0$

Yes. $\{I(x) \leftrightarrow x \geq 1\}$ is a solution of $\mathcal{C}$!
Enumeration of Candidates $\rho_1, \ldots, \rho_n$

• Challenges:
  1. The number $n$ of candidates must be \textit{bounded}
  2. Candidates must satisfy $\mathcal{E}$ and should be \textit{simpler} (per Occam’s razor) and \textit{essentially different} each other for \textit{more chance to satisfy} $\mathcal{C}$

• Our Solutions:
  1. Compute \textit{predicate abstraction} \cite{Graf97} $\alpha(\mathcal{E})$ of $\mathcal{E}$ using a finite set $Q$ of predicates to reduce the search space of candidates to the \textit{finite set} of $Q$-expressible solutions
  2. Enumerate “\textit{promising}” satisfying Boolean assignments of $\alpha(\mathcal{E})$ via \textit{survey inspired decimation (SID)} \cite{Braunstein05}
Example: Predicate Abstraction

Example Instances $\mathcal{E}$:
- $\neg I(0)$
- $I(1)$

Predicates Set $\mathcal{Q}$:
$$\{ \lambda x. \bot, \lambda x. \top, \lambda x. x \geq 0, \lambda x. -x \geq 0, \lambda x. x \geq 1, \lambda x. -x \geq -1 \}$$
Example: Predicate Abstraction

Example Instances $\mathcal{E}$:
- $\neg I(0)$
- $I(1)$

Predicates Set $\mathcal{Q}$:
\[
\{ \lambda x. \bot, \lambda x. T, \\
\lambda x. x \geq 0, \lambda x. 0 \geq x, \\
\lambda x. x \geq 1, \lambda x. 1 \geq x \}
\]

Solution Template with Boolean parameters $b_\bot, b_{\geq 1}, b_{0\geq}, \ldots$:
\[
\{ I(x) \rightarrow (b_\bot \Rightarrow \bot) \land (b_{\geq 1} \Rightarrow x \geq 1) \land (b_{0\geq} \Rightarrow 0 \geq x) \land \ldots \}
\]

Predicate Abstraction $\alpha(\mathcal{E}) \equiv (b_\bot \lor b_{\geq 1}) \land \neg b_\bot \land \neg b_{0\geq}$

Satisfying Assignments for $\alpha(\mathcal{E})$: $\{b_\bot \mapsto \bot, b_{0\geq} \mapsto \bot, b_{\geq 1} \mapsto T, b_T \mapsto *, b_{\geq 0} \mapsto *, b_{1\geq} \mapsto *\}$, indicating that $\lambda x. \bot$ and $\lambda x. 0 \geq x$ must not be used and $\lambda x. x \geq 1$ must be used

$\therefore$ $\mathcal{Q}$-expressible sols. are $\{I(x) \mapsto x \geq 1\}$ and $\{I(x) \mapsto x \geq 1 \land 1 \geq x\}$
SID-based Enumeration of “Promising” Satisfying Assignments for $\alpha(\mathcal{E})$

- To obtain a simpler solution (e.g., $\{I(x) \mapsto x \geq 1\}$), detect and assign $\bot$ to the don’t-care variables
- To reduce similar solutions while preserving different ones belonging to different solution clusters, use SID to iteratively assign a value $v$ to a variable of $\alpha(\mathcal{E})$ with the highest bias toward $v$ (i.e., “sufficiently” determined)
  - The biases of each variable are computed via probabilistic inference (survey propagation) in a graphical model (factor graph) obtained from $\alpha(\mathcal{E})$ (see the paper for details)
Evaluation

• Implemented the presented method as a pCSP solver **PCSat** using **Z3** as the backend SMT solver

• Tested **PCSat** on the benchmark sets from
  • SyGuS-Comp 2017 and 2018 (Invariant Synthesis Track)
  • CHC-COMP 2019 (LIA-nonlin Track)
  • New pCSP benchmarks of branching-time safety verification that *go beyond the scope of existing SyGuS and CHC solvers*
Results on 127 Benchmarks from SyGuS-Comp 2018

<table>
<thead>
<tr>
<th></th>
<th>#SAT</th>
<th>#UNSAT</th>
</tr>
</thead>
<tbody>
<tr>
<td>SID(#cand=1)</td>
<td>91</td>
<td>8</td>
</tr>
<tr>
<td>SID(#cand=2)</td>
<td>92</td>
<td>8</td>
</tr>
<tr>
<td>SID(#cand=4)</td>
<td>96</td>
<td>8</td>
</tr>
<tr>
<td>SID(#cand=8)</td>
<td>100</td>
<td>8</td>
</tr>
<tr>
<td>SID(#cand=16)</td>
<td>96</td>
<td>8</td>
</tr>
<tr>
<td>SID(#cand=32)</td>
<td>95</td>
<td>7</td>
</tr>
<tr>
<td>SID(#cand=64)</td>
<td>94</td>
<td>6</td>
</tr>
<tr>
<td>SAT(#cand=1)</td>
<td>94</td>
<td>8</td>
</tr>
<tr>
<td>SAT(#cand=2)</td>
<td>92</td>
<td>8</td>
</tr>
<tr>
<td>SAT(#cand=4)</td>
<td>91</td>
<td>8</td>
</tr>
<tr>
<td>SAT(#cand=8)</td>
<td>89</td>
<td>7</td>
</tr>
</tbody>
</table>
Summary: Program Verification via Predicate Constraint Satisfaction Problem (pCSP)

Target Program $P$ & Specification $\psi$

Constraint Generation

New method for Looping & Recursive Programs

$pCSP$ Constraints $C$ on Predicate Variables

Verification Intermediary
Independent of Particular Target and Method 😊

Constraint Solving

New method based on CEGIS, Pred. Abstraction, Survey Inspired Decimation

$C$ is Sat ($P$ satisfies $\psi$), $C$ is Unsat ($P$ violates $\psi$), or Unknown
Ongoing and Future Work

- Extend pCSP to support branching-time *liveness* verification of (possibly) *ininitely* branching programs
  - *Liveness*: “something good will eventually happen”

- Develop constraint generation tools for C, Java, OCaml, $\mu$CLP

- Extend **PCSat** to support more theories (Arrays, ADTs, heaps, ...)

- Apply *other constraint satisfaction methods* to enumerate sols.

- Apply *other probabilistic inference* like *variational inference* and *approximate model counting* by directly *modeling pCSP as factor graphs* representing joint probability distributions over random predicate variables