Predicate Refinement Heuristics in Program Verification with CEGAR

Tachio Terauchi (JAIST)
Part of this is joint work with Hiroshi Unno (U. Tsukuba)
Predicate Abstraction with CEGAR

Iteratively generate **candidate predicates** $F \subseteq \text{Preds}(T)$

– until $F$ forms a proof of the given program
  
  • $T$: background FOL theory (e.g., QFLRA)

Much success for imperative programs (SLAM, BLAST, ...)
for concurrent programs (Threader, SymmPA, ...)
for functional programs (Depcegar, MoCHi, ...)
Predicate Refinement

• Input:
  – Currently irrefutable counterexample $\pi$
    • i.e., $F \not\models \pi$ where $F$ is current candidate pred. set

• Output:
  – Set of predicates $F'$ such that $F' \models \pi$

Issue:
• There can be multiple (in general, $\infty$ many) $F'$ s.t. $F' \models \pi$
• Choice of $F'$ can significantly affect CEGAR performance
Example
(How refinement choice affects CEGAR performance)

Proof of the program:
• $\phi_{inv} \equiv a=b \Rightarrow y=x+z$

Counterexample $\pi_1$

Proof of $\pi_1$: \{ $\phi_0$, $\phi_1$, $\phi_0 \lor \phi_1$ \}
• $\phi_0 \equiv x=a \land y=b \land z=0$
• $\phi_1 \equiv x=a \land y=b+1 \land z=1$

Sufficient for $\pi_1$ but not for the program
Example
(How refinement choice affects CEGAR performance)

\[
a = \text{nondet}(); b = \text{nondet}();
x = a; y = b; z = 0;
\text{while (nondet())} \{
    y++; z++;
\}
\text{while (z \neq 0) } \{
    y--; z--;
\}
\text{if (a=b \&\& y!=x) } \{ \text{assert false; } \}
\]

\(\pi_1\) : refuted by \(\{ \phi_0, \phi_1, \phi_0 \lor \phi_1 \}\)
\(\pi_2\) : refuted by \(\{ \forall F \mid F \subseteq \{\phi_0, \phi_1, \phi_2 \} \}\)
\[\vdots\]
\(\pi_i\) : refuted by \(\{ \forall F \mid F \subseteq \{\phi_0, ..., \phi_i \} \}\)
\[\vdots\]
\(\pi_i \equiv \text{Loops unfolded i times}\)
\(\phi_i \equiv x = a \land y = b + i \land z = i\)

Proof of the program:
• \(\phi_{\text{inv}} \equiv a=b \Rightarrow y=x+z\)

CEGAR DIVERGES!
Outline

✓ Introduction
2. Refinement scheme with convergence guarantee
3. Fast convergence via “small refinements”
Based on [Terauchi, Unno ESOP 2015]

REFINEMENT SCHEME WITH GUARANTEED CEGAR CONVERGENCE
Example
(How refinement choice affects CEGAR performance)

Proof of the program:
• \( \phi_{inv} \equiv a=b \Rightarrow y=x+z \)

Key Observation: \( \phi_{inv} \) refutes every \( \pi_i \)

\( \therefore \) Can force convergence by restricting predicates inferred by refinement
Stratified Refinement [1,2]

• Prepare growing strata of predicate sets:

\[
\bullet \quad \text{Each } L_i \subseteq \text{Preds}(T) \text{ is finite}
\]

\[
\bullet \quad \text{Preds}(T) = \bigcup_{i=1}^{\omega} L_i
\]

• In each refinement step:

\[
\bullet \quad \text{Restrict inferred predicates to some } L_i
\]

\[
\bullet \quad \text{raise } L_i \text{ to next level when no proof of given c.e.x. is in } L_i
\]

**GUARANTEED CEGAR CONVERGENCE**

• under promise that a proof exists in $T$

---

Issue with Stratified Refinement

• Refinement step must decide if current $L_i$ has a proof of given counterexample
  • i.e., decide if $\exists F \subseteq L_i : F \models \pi$
    – Such exact finite-predicate-set-restricted proof search is hard
  • cf. ESOP’15 paper for details

Our Goal

More Practical Refinement with Convergence Guarantee
  – under the same promise that a proof exists in $\text{Preds}(T)$
New Proposal: Relaxed Stratification

• Prepare strata of predicate set pairs
  \[ B_0 \cup E_0, \ B_1 \cup E_1, \ldots \ B_i \cup E_i \ldots \]

  \- Each \( B_i \cup E_i \subseteq \text{Preds}(T) \) is finite
  \- \( B_i \subseteq B_{i+1} \) for each \( B_i \)
  \- \( \text{Preds}(T) = \bigcup_{i=1}^{\omega} B_i \)

• In each refinement step:
  \- Restrict inferred predicates to some \( B_i \cup E_i \)
  \- Fail to infer preds. and raise level only if no proof is in \( B_i \)

Need not to exactly decide existence of proof in \( B_i \) or in \( B_i \cup E_i \)

(Exact) stratification is special case where \( E_i = \emptyset \)
Correctness

**Theorem:** With relaxed stratification, CEGAR converges under the promise that program can be proved by Preds(T)

**Proof sketch:**

– Follows from **Key Observation:** proof of program is proof of its counterexamples. Therefore:

  • Stratum only goes up to $B_i \cup E_i$ where $B_i \supseteq$ proof of prog.
  • Stays in same stratum only for finite number of CEGAR iterations
Outline

✓ Introduction

2. Refinement scheme with convergence guarantee
   ✓ Relaxed Stratification Scheme
   b. Concrete instances of relaxed stratification
      • “Restricted” recursion-free Horn-clause solvers
   c. Experiments

3. CEGAR iteration bound via “small” refinement
Horn Clause Constraints

Predicate variables \( P, Q, P_1, P_2, \ldots \)

Body clause \( B ::= \theta | P(\vec{x}) | B \land B \)

Horn clause constraint \( hc ::= B \Rightarrow P(\vec{x}) | B \Rightarrow \bot \)

Horn clause constraint set \( H ::= \emptyset | H \cup \{hc\} \)

- \( H \) is satisfiable if \( \exists \sigma : pvars(H) \leftrightarrow \text{Preds}(\mathcal{T}). \forall hc \in H. \models \sigma(hc). \)
- \( H \) is recursion-free if \( \{(P, Q) | \ldots P(\ldots) \cdots \Rightarrow Q(\ldots) \in H\} \) is acyclic.
- \( H \) is tree-like if each \( P \in pvars(H) \) occurs at most once in left of \( \Rightarrow \) and at most once in right of \( \Rightarrow \).
Reducing Predicate Refinement to Horn-clause Solving

**Prop:** For any $\pi$, exists recursion-free $H_\pi$ such that

$\sigma$ is solution of $H_\pi \iff \{ \sigma(P) \mid P \in \text{dom}(\sigma) \} \vdash \pi$

**Standard Refinement Algorithm:**

1. Build $H_\pi$
2. Find solution $\sigma$ of $H_\pi$
3. Return $\{ \sigma(P) \mid P \in \text{dom}(\sigma) \}$ as inferred predicates

**Plan:** Modify Step 2 so that

- Finds solution from $B_i \cup E_i$ (**not from entire Preds(T)**)  
- Fails to solve only if no solution is in $B_i$
Technical Detour: Tree Interpolation

- Labeled tree \((V,E,\Theta)\) - \(V\): nodes \(E\): edges \(\Theta: V \rightarrow \text{Formula}(T)\)
- \(I: V \rightarrow \text{Formula}(T)\) is **tree interpolant (ITP)** of \((V,E,\Theta)\) \iff
  - For root \(v \in V\), \(I(v) = \text{false}\)
  - \(\forall v \in V. \Theta(v) \land (\wedge_{(v,v') \in E} I(v')) \Rightarrow I(v)\)
  - \(\forall v \in V. \text{vars}(I(v)) \subseteq \text{sharedvars}(v)\)

\[
\begin{align*}
I(v_0) &= \text{false} \\
I(v_1) &= x \leq 1 \\
I(v_2) &= y \leq 0 \\
I(v_3) &= x \leq 1 \\
I(v_4) &= z \geq w + 1
\end{align*}
\]

vars. occurring both in & out of subtree rooted at \(v\)
Reducing Horn clauses to Tree Interpolation

**Theorem[Rümmer+’13]:** Given tree-like clause set $H$, can build tree-interpolation instance $(V_H, E_H, \Theta_H)$ s.t.

\[ \{ P \mapsto \lambda x \overrightarrow{P}.I(P) \} \models H \iff I \text{ is tree-interpolant of } (V_H, E_H, \Theta_H) \]

where $V_H = pvars(H) \cup \{ v_{rt} \}$, $v_{rt} \notin pvars(H)$, and $\overrightarrow{x_P}$’s are fresh variables.

**Can be extended to general recursion-free case (cf. [Rümmer+’13])**

- Example:

| $x \leq 1 \land t = 1 \Rightarrow P(x)$ | $x_P \leq 1 \land t = 0$ |
| $z = x + 1 \Rightarrow Q(w, z)$ | $z_Q = w_Q + 1$ |
| $y \leq 0 \Rightarrow R(y)$ | |
| $w = 1 \land P(x) \land Q(w, z) \Rightarrow S(x, z)$ | $w_Q = 1 \land x_P = x_S \land z_Q = z_S$ |
| $x - y > z \land S(x, y) \Rightarrow \bot$ | $y_R \leq 0$ |

\[ v_{rt} \]
\[ x_S - y_R > z_S \]
Review

• We want to:
  – Infer solution $\sigma$ of $H_\pi$ s.t. $\text{ran}(\sigma) \subseteq B_i \cup E_i$
  – Fail to infer only when no solution of range $B_i$ exists

• Horn-clause to tree interpolation reduction says:
  – Infer tree interpolant $I$ of $(V_H, E_H, \Theta_H)$ s.t. $\text{ran}(I) \subseteq B_i \cup E_i$
  – Fail to infer only when no interpolant of range $B_i$ exists

$\{P \leftrightarrow \lambda x P. I(P)\}$ is solution of $H \iff I$ is tree interpolant of $(V_H, E_H, \Theta_H)$

• So, it suffices to do “restricted” tree interpolation:
  – Infer tree interpolant $I$ of $(V_H, E_H, \Theta_H)$ s.t. $\text{ran}(I) \subseteq B_i \cup E_i$
  – Fail to infer only when no interpolant of range $B_i$ exists
Restricted Tree Interpolation

Standard tree interpolation algorithm:

– Input: \((V,E,\Theta)\)
– Use SMT solver to check \(\bigwedge_{v \in V} \Theta(v)\) is UNSAT
  • obtain resolution proof deriving “false”
– Compute partial ITPs at each node of resolution proof
  • ITP = partial ITP at root node

Modification: theory-level-restricted SMT solving

– Restrict leaf (i.e., theory) level reasoning to only produce partial ITPs in \(B_i\)
  • Prototype implementation using template-based technique

Only uses expensive finite preds. res. search at leaf levels
Tree Interpolant Generation

• \( p \in \text{Atoms}(T) \)

• \( C ::= p \mid \neg p \mid C \lor C \)

\[
\frac{C \land \ldots = \Theta(v')}{(V, E, \Theta) \vdash C : I}
\]

\[
\frac{(V, E, \Theta) \vdash p \lor C_1 : I_1 \quad (V, E, \Theta) \vdash \neg p \lor C_2 : I_2}{(V, E, \Theta) \vdash C_1 \lor C_2 : I}
\]

\[
I = \lambda v. \begin{cases} 
\text{false} & \text{if } (v', v) \in E^* \\
\text{true} & \text{otherwise}
\end{cases}
\]

\[
I = \lambda v. \begin{cases} 
I_1(v) \land I_2(v) & \text{if } p \in \text{outs}(v) \\
I_1(v) \lor I_2(v) & \text{otherwise}
\end{cases}
\]

\[
\frac{I \text{ is tree itp. of } (V, E, \Theta_C) \text{ where } \text{ran}(I) \subseteq B_i}{(V, E, \Theta) \vdash C : I}
\]

\[
\Theta_C = \Theta \text{ with labels restricted to } \text{Atoms}(C)
\]

**Theorem:** Inferred ITP is in \( B_i^{\land \lor} \), and some ITP is inferred if one exists in \( B_i \)

\[
\text{So, } E_i = B_i^{\land \lor}
\]
Outline

✓ Introduction

2. Refinement scheme with convergence guarantee
   ✓ Relaxed Stratification Scheme
   
b. Concrete instances of relaxed stratification
      • “Restricted” recursion-free Horn-clause solvers
         ✓ Theory-level-restricted tree interpolation
            ii. Another concrete instance

c. Experiments

3. CEGAR iteration bound via “small” refinement
Another Concrete Instance of Relaxed Stratification Scheme

• “Restricted” Horn clause constraint solver (again)

**Two-phase approach**

1. Partition clauses into bounded-size trees
2. Infer restricted solutions to predicate variables at partition boundaries
3. Infer unrestricted solutions to the rest

Only use expensive finite predicate-restricted search for few predicate variables

Becomes relaxed stratification under some mild assumptions on generated counterexample patterns
Algorithm Explained Pictorially

1. Partition into bounded-size sub-trees
2. Infer restricted solutions to boundary predicate variables
   • E.g., via template-based method
     ➢ Could also use another relaxed stratification refinement alg. itself
3. Infer unrestricted solutions to the rest
   • via standard approaches [Unno+’09, Terauchi’10, Rümmer+’13, etc.]

Theorem: This satisfies the requirements of relaxed stratification, assuming there is a “generator” clause set $H_{\text{gen}}$ s.t. any c.ex. is an unfolding of $H_{\text{gen}}$
Refinement Algorithm Schemas

• These are actually **algorithm schemas**
  – take other refinement algorithms as modules
  – generate refinement algorithms satisfying requirements of relaxed stratification

See ESOP’15 paper for details
25

Outline

✓ Introduction

2. Refinement scheme with convergence guarantee
   ✓ Relaxed Stratification Scheme
   ✓ Concrete instances of relaxed stratification
     ✓ “Restricted” Horn clause solvers algorithms
       ✓ Theory-level-restricted tree interpolation
       ✓ 2-phase inference approach via bounded partitioning

c. Experiments

3. Fast convergence via “small refinements”
Prototype Implementation

New refinement algorithm

– Algorithm 1 used as module of algorithm 2
– Z3 [1] for template-based constraint solving
– MathSAT5 [2] for unrestricted refinement

• Used as refinement engine of MoCHi [3]
  – Software model checker for higher-order functional programs based on CEGAR

Experiment Results: Individual Refinement Runs

- 318 counterexamples generated from 139 benchmark programs
- Three refinement algorithms:
  - New algorithm
  - Unrestricted refinement
  - Exact stratification
Experiment Results:
Overall Verification Performance

• 139 benchmark programs
• MoCHi with each refinement algorithm:
  – New algorithm
  – Unrestricted refinement
  – Exact stratification
Outline

✓ Introduction
✓ Refinement scheme with convergence guarantee
  ✓ Relaxed Stratification Scheme
  ✓ Concrete instances of relaxed stratification
    ✓ “Restricted” Horn clause solvers algorithms
  ✓ Experiments

3. Fast convergence via “small refinements”
FAST CONVERGENCE VIA “SMALL REFINEMENTS”

Based on [Terauchi SAS 2015]
Talk so far

Predicate refinement in CEGAR

– Return predicates that refutes given counterexample
– Clever choice of predicates can make CEGAR converge

Q: Can we say anything about convergence speed?

Short Answer: YES
Our Result

Small Refinement Heuristic (SRH)

- Refinement phase returns “small” proof of counterexample’s safety
  [Hoder+’12][Scholl+’14][Albarghouthi,McMillan’14][Unno,Terauchi’15]

• We will show:
  - CEGAR with SRH converges in number of CEGAR iterations bounded in the size of the proof for the input program
Example (from before)

```
a = nondet(); b = nondet();
x = a; y = b; z = 0;
while (nondet()) {
    y++; z++;
}
while (z != 0) {
    y--; z--;
}
if (a=b && y!=x) { assert false; }
```

\(\pi_1:\) refuted by \(\{\phi_0, \phi_1, \phi_0 \lor \phi_1\}\)
\(\pi_2:\) refuted by \(\{\forall F \mid F \subseteq \{\phi_0, \phi_1, \phi_2\}\}\)
\(\vdots\)
\(\pi_i:\) refuted by \(\{\forall F \mid F \subseteq \{\phi_0, \ldots, \phi_i\}\}\)
\(\vdots\)

\(\pi_i \equiv \text{Loops unfolded i times}\)
\(\phi_i \equiv x = a \land y = b + i \land z = i\)

Key Observation: \(\phi_{\text{inv}}\) refutes every \(\pi_i\)

\[\therefore\] Inferring small refinements should hasten convergence

Refinement will infer \(\phi_{\text{inv}}\) or some other small proof of program before inferring large \(\phi_k\)'s

How to define “small”?
Proof Size Metric and SRH

**Def:** \( \text{size} : \mathcal{P}_{\text{fin}}(\text{Preds}(T)) \rightarrow \text{Nat} \) is **generic proof size metric** if \( \exists c > 0. \forall n \geq 0. \left| \{ F \mid \text{size}(F) \leq n \} \right| \leq c^n \)

**Def:** \( \text{minprfsize}(\gamma) = \min_{F \in \{ F \mid F \vdash \gamma \}} \text{size}(F) \)

\[ \gamma : \text{cex or program} \]

**Def:** CEGAR with **Small Refinement Heuristic (SRH)** is CEGAR with \( \text{Refine}() \) satisfying:

\[ \exists \text{poly } f. \forall \pi. \text{ if } \text{Refine}(\pi) = F \text{ then } \text{size}(F) \leq f(\text{minprfsize}(\pi)) \]
Convergence Bound

**Theorem:** Suppose proof size metric is generic. Then CEGAR with SRH converges in number of iterations bounded exponentially in $\min\text{prfsize}(P)$.

- **Proof:** By KEY OBSERVATION and simple counting argument

**Corollary:** CEGAR with SRH converges in exponential number of iterations under the promise that program has polynomial size proof

Can a tighter bound be obtained with a more concrete setting?
Bound for CFG-represented Programs

• Assumptions
  – Program represented by reducible Control Flow Graph
  – Counterexamples are loop-unfoldings of the CFG
    • every loop unfolded the same number of times
  – Proof is Floyd-style node-wise inductive invariant
  – Predicate abstraction is Cartesian predicate abstraction
  – \( \text{size}(F) = \sum_{\phi \in F} \text{syntactic_size}(\phi) \)

• Theorem: CEGAR with SRH converges in number of iterations bounded polynomially in \( \text{minprfs}(P) \) for CFG programs.

See SAS’15 paper for details
Outline

✓ Introduction
✓ Refinement scheme with convergence guarantee
  ✓ “Relaxed” Stratification
3. Fast convergence via “small refinements”
  ✓ Generic Setting
    ✓ $\exp(\text{minprfsize}(P))$ CEGAR iteration bound
  ✓ CFG-represented Programs
    ✓ $\text{poly}(\text{minprfsize}(P))$ CEGAR iteration bound (under various assumptions)
Some Thoughts and Future Work

Results on CEGAR iteration bound can be taken as negative results?

– Inferring small refinements must be hard because otherwise verification would be easy?
  • Substantiates the experience with stratified refinement

Need further investigation

– Hardness of verification w.r.t. “predicate refinement oracles” and under “small proof promise” seems to be underexplored