

Predicate Refinement Heuristics in Program Verification with CEGAR

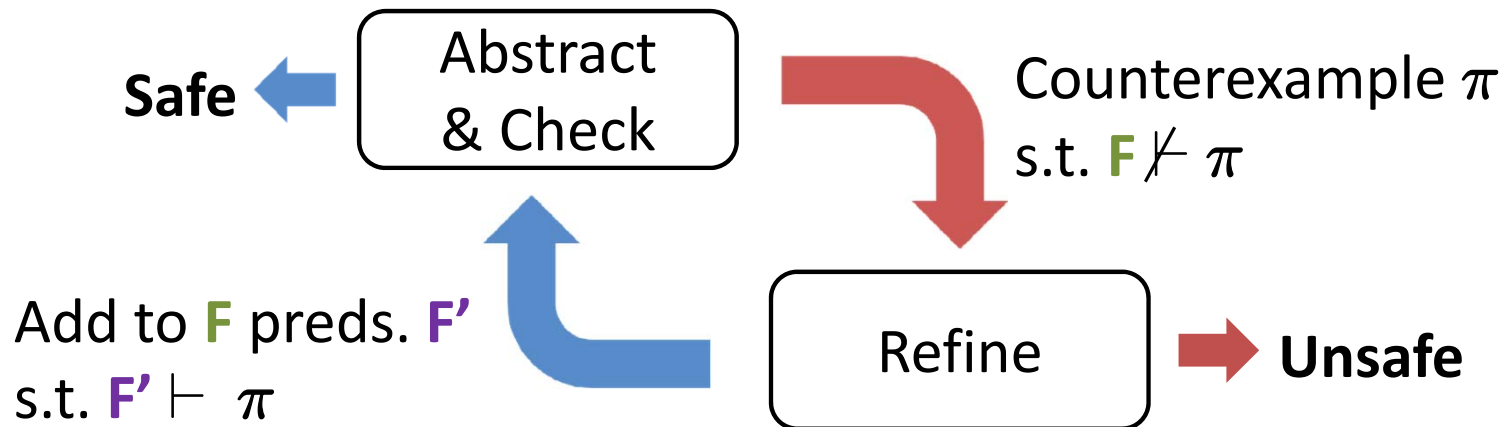
Tachio Terauchi (JAIST)

Part of this is joint work with Hiroshi Unno (U. Tsukuba)

Program Verification with CEGAR

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

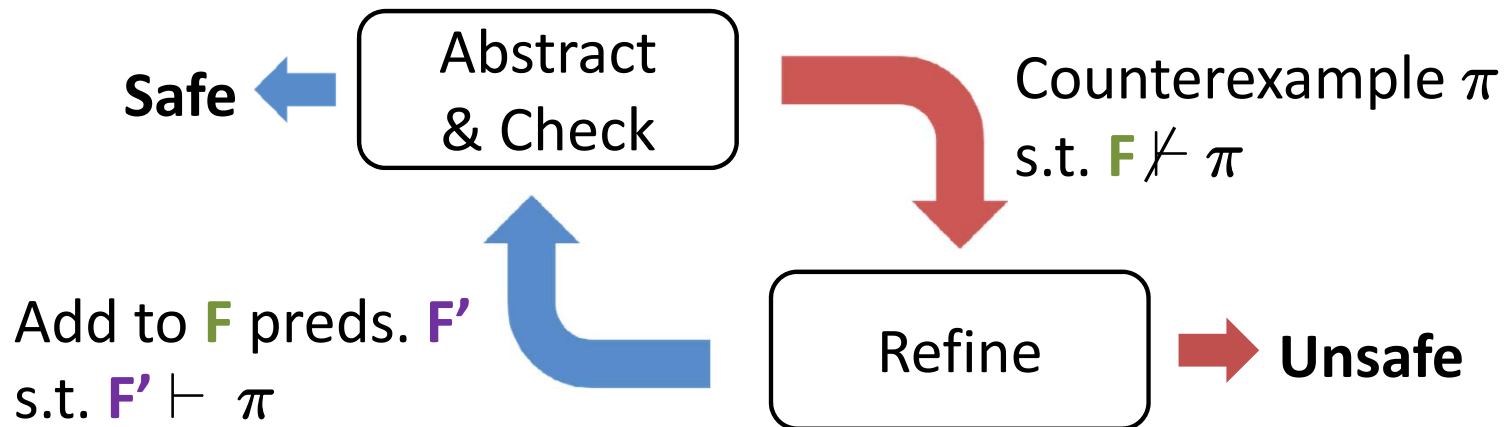
- until F forms a proof of the given program
 - T : background FOL theory (e.g., QFLRA)



Program Verification 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)

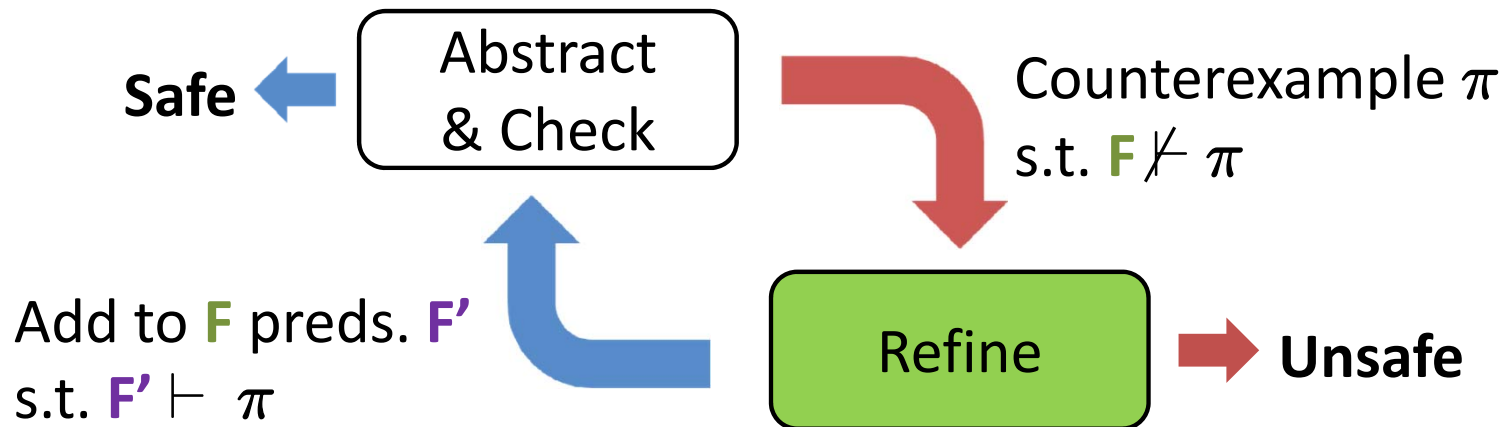


Much success for imperative programs (SLAM, BLAST, ...)
for concurrent programs (Threader, SymmPA, ...)
for functional programs (Depcegar, MoChi, ...)

Program Verification 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 π
 - i.e., $F \not\models \pi$ where F is current candidate pred. set
- Output:
 - Set of predicates F' such that $F' \models \pi$

Predicate Refinement

- Input:
 - Currently irrefutable counterexample π
 - 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, ∞ many) F' s.t. $F' \models \pi$
- Choice of F' can significantly affect CEGAR performance

Example

(How refinement choice affects CEGAR performance)

```
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; }
```

Example

(How refinement choice affects CEGAR performance)

```
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; }
```

Proof of the program:

$\phi_{\text{inv}} \equiv a=b \Rightarrow y=x+z$

Example

(How refinement choice affects CEGAR performance)

```
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; }
```



Counterexample π_1

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

Proof of the program:

$\phi_{\text{inv}} \equiv a=b \Rightarrow y=x+z$

Example

(How refinement choice affects CEGAR performance)

```
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; }
```

Proof of the program:

$\phi_{\text{inv}} \equiv a=b \Rightarrow y=x+z$

Counterexample π_1

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

Proof of π_1 : $\{ \phi_0, \phi_1, \phi_0 \vee \phi_1 \}$

- $\phi_0 \equiv x=a \wedge y=b \wedge z=0$
- $\phi_1 \equiv x=a \wedge y=b+1 \wedge z=1$

Example

(How refinement choice affects CEGAR performance)

```
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; }
```

Proof of the program:

$\phi_{\text{inv}} \equiv a=b \Rightarrow y=x+z$

Counterexample π_1

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

Proof of π_1 : $\{ \phi_0, \phi_1, \phi_0 \vee \phi_1 \}$

- $\phi_0 \equiv x=a \wedge y=b \wedge z=0$
- $\phi_1 \equiv x=a \wedge y=b+1 \wedge z=1$

Sufficient for π_1 but not for the program

Example

(How refinement choice affects CEGAR performance)

```
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; }
```

π_1 : refuted by $\{ \phi_0, \phi_1, \phi_0 \vee \phi_1 \}$

$\pi_i \equiv$ Loops unfolded i times

$\phi_i \equiv x = a \wedge y = b + i \wedge z = i$

Proof of the program:

$\phi_{\text{inv}} \equiv a=b \Rightarrow y=x+z$

Example

(How refinement choice affects CEGAR performance)

```
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; }
```

Proof of the program:

$\phi_{\text{inv}} \equiv a=b \Rightarrow y=x+z$

π_1 : refuted by $\{ \phi_0, \phi_1, \phi_0 \vee \phi_1 \}$

π_2

$\pi_i \equiv$ Loops unfolded i times

$\phi_i \equiv x = a \wedge y = b + i \wedge z = i$

Example

(How refinement choice affects CEGAR performance)

```
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; }
```

Proof of the program:

$\phi_{\text{inv}} \equiv a=b \Rightarrow y=x+z$

π_1 : refuted by $\{ \phi_0, \phi_1, \phi_0 \vee \phi_1 \}$

π_2 : refuted by $\{ \forall F \mid F \subseteq \{ \phi_0, \phi_1, \phi_2 \} \}$

$\pi_i \equiv$ Loops unfolded i times

$\phi_i \equiv x = a \wedge y = b + i \wedge z = i$

Example

(How refinement choice affects CEGAR performance)

```
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; }
```

Proof of the program:

$\phi_{\text{inv}} \equiv a=b \Rightarrow y=x+z$

π_1 : refuted by $\{ \phi_0, \phi_1, \phi_0 \vee \phi_1 \}$
 π_2 : refuted by $\{ \forall F \mid F \subseteq \{ \phi_0, \phi_1, \phi_2 \} \}$
 \vdots
 π_i : refuted by $\{ \forall F \mid F \subseteq \{ \phi_0, \dots, \phi_i \} \}$
 \vdots

$\pi_i \equiv$ Loops unfolded i times

$\phi_i \equiv x = a \wedge y = b + i \wedge z = i$

Example

(How refinement choice affects CEGAR performance)

```
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; }
```

Proof of the program:

$\phi_{\text{inv}} \equiv a=b \Rightarrow y=x+z$

π_1 : refuted by $\{ \phi_0, \phi_1, \phi_0 \vee \phi_1 \}$
 π_2 : refuted by $\{ \forall F \mid F \subseteq \{ \phi_0, \phi_1, \phi_2 \} \}$
 \vdots
 π_i : refuted by $\{ \forall F \mid F \subseteq \{ \phi_0, \dots, \phi_i \} \}$
 \vdots

$\pi_i \equiv$ Loops unfolded i times

$\phi_i \equiv x = a \wedge y = b + i \wedge z = i$

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)

```
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; }
```

Proof of the program:

$\phi_{\text{inv}} \equiv a=b \Rightarrow y=x+z$

π_1 : refuted by $\{ \phi_0, \phi_1, \phi_0 \vee \phi_1 \}$
 π_2 : refuted by $\{ \forall F \mid F \subseteq \{ \phi_0, \phi_1, \phi_2 \} \}$
 \vdots
 π_i : refuted by $\{ \forall F \mid F \subseteq \{ \phi_0, \dots, \phi_i \} \}$
 \vdots

$\pi_i \equiv$ Loops unfolded i times

$\phi_i \equiv x = a \wedge y = b + i \wedge z = i$

Example

(How refinement choice affects CEGAR performance)

```
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; }
```

π_1 : refuted by $\{ \phi_0, \phi_1, \phi_0 \vee \phi_1 \}$
 π_2 : refuted by $\{ \forall F \mid F \subseteq \{ \phi_0, \phi_1, \phi_2 \} \}$
 \vdots
 π_i : refuted by $\{ \forall F \mid F \subseteq \{ \phi_0, \dots, \phi_i \} \}$
 \vdots

$\pi_i \equiv$ Loops unfolded i times

$\phi_i \equiv x = a \wedge y = b + i \wedge z = i$

Proof of the program:

$\phi_{\text{inv}} \equiv a=b \Rightarrow y=x+z$

Key Observation: ϕ_{inv} refutes every π_i

Example

(How refinement choice affects CEGAR performance)

```
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; }
```

π_1 : refuted by $\{ \phi_0, \phi_1, \phi_0 \vee \phi_1 \}$
 π_2 : refuted by $\{ \forall F \mid F \subseteq \{ \phi_0, \phi_1, \phi_2 \} \}$
 \vdots
 π_i : refuted by $\{ \forall F \mid F \subseteq \{ \phi_0, \dots, \phi_i \} \}$
 \vdots

$\pi_i \equiv$ Loops unfolded i times
 $\phi_i \equiv x = a \wedge y = b + i \wedge z = i$

Proof of the program:

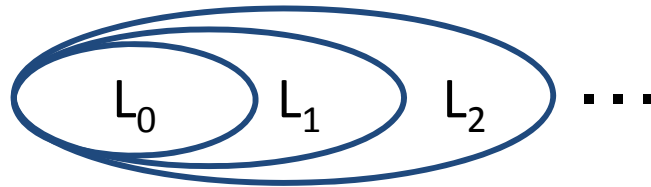
$\phi_{\text{inv}} \equiv a=b \Rightarrow y=x+z$

Key Observation: ϕ_{inv} refutes every π_i

**\therefore Can force convergence by restricting predicates
inferred by refinement**

Stratified Refinement [1,2]

- Prepare growing strata of predicate sets:



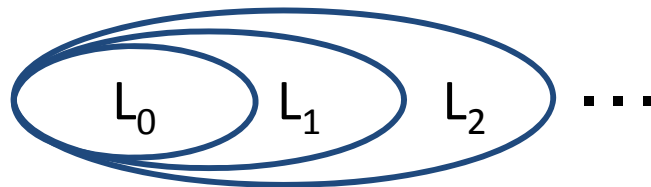
- Each $L_i \subseteq \text{Preds}(T)$ is finite
- $\text{Preds}(T) = \bigcup_{i=1}^{\omega} L_i$

[1] R. Jhala, K. McMillan. Practical and complete approach to predicate refinement. TACAS'06.

[2] K. McMillan. Quantified invariant generation using an interpolating saturation prover. TACAS'08.

Stratified Refinement [1,2]

- Prepare growing strata of predicate sets:



- Each $L_i \subseteq \text{Preds}(T)$ is finite
- $\text{Preds}(T) = \bigcup_{i=1}^{\omega} L_i$
- In each refinement step:
 - Restrict inferred predicates to some L_i
 - raise L_i 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**

[1] R. Jhala, K. McMillan. Practical and complete approach to predicate refinement. TACAS'06.

[2] K. McMillan. Quantified invariant generation using an interpolating saturation prover. TACAS'08.

Issue with Stratified Refinement

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

Issue with Stratified Refinement

- Refinement step must decide if current L_i has a proof of given counterexample
 - i.e., decide if $\exists F \subseteq L_i . F \vdash \pi$
 - 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

$$B_0 \cup E_0, B_1 \cup E_1, \dots, B_i \cup E_i, \dots$$

Base & Extension

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

New Proposal: Relaxed Stratification

- Prepare strata of predicate set pairs

$$B_0 \cup E_0, B_1 \cup E_1, \dots, B_i \cup E_i, \dots$$

Base & Extension

- 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

New Proposal: Relaxed Stratification

- Prepare strata of predicate set pairs

$$B_0 \cup E_0, B_1 \cup E_1, \dots, B_i \cup E_i, \dots$$

Base & Extension

- 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 $B_i \cup E_i$

New Proposal: Relaxed Stratification

- Prepare strata of predicate set pairs

$$B_0 \cup E_0, B_1 \cup E_1, \dots, B_i \cup E_i, \dots$$

Base & Extension

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

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

- 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 $B_i \cup E_i$

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
 - a. Concrete instances of relaxed stratification
 - Modified tree interpolation algorithm
 - b. Experiments
- 3. CEGAR iteration bound via “small” refinement

Tree Interpolation

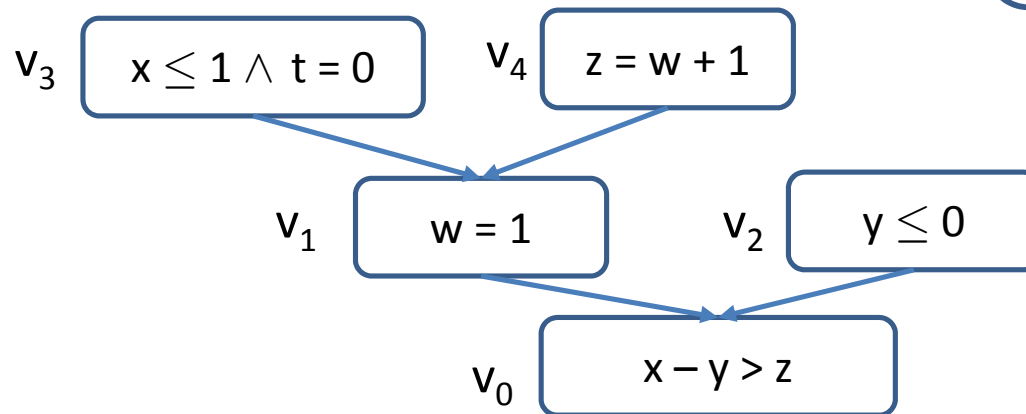
- Labeled tree (V, E, Θ) - V : nodes E : edges $\Theta: V \rightarrow \text{Preds}(T)$
- $I: V \rightarrow \text{Preds}(T)$ is **tree interpolant (ITP)** of $(V, E, \Theta) \Leftrightarrow$
 - For root $v \in V$, $I(v) = \text{false}$
 - $\forall v \in V. \Theta(v) \wedge (\bigwedge_{(v, v') \in E} I(v')) \Rightarrow I(v)$
 - $\forall v \in V. \text{vars}(I(v)) \subseteq \text{sharedvars}(v)$

vars. occurring both
in & out of subtree
rooted at v

Tree Interpolation

- Labeled tree (V, E, Θ) - V : nodes E : edges $\Theta: V \rightarrow \text{Preds}(T)$
- $I: V \rightarrow \text{Preds}(T)$ is **tree interpolant (ITP)** of $(V, E, \Theta) \Leftrightarrow$
 - For root $v \in V$, $I(v) = \text{false}$
 - $\forall v \in V. \Theta(v) \wedge (\bigwedge_{(v,v') \in E} I(v')) \Rightarrow I(v)$
 - $\forall v \in V. \text{vars}(I(v)) \subseteq \text{sharedvars}(v)$

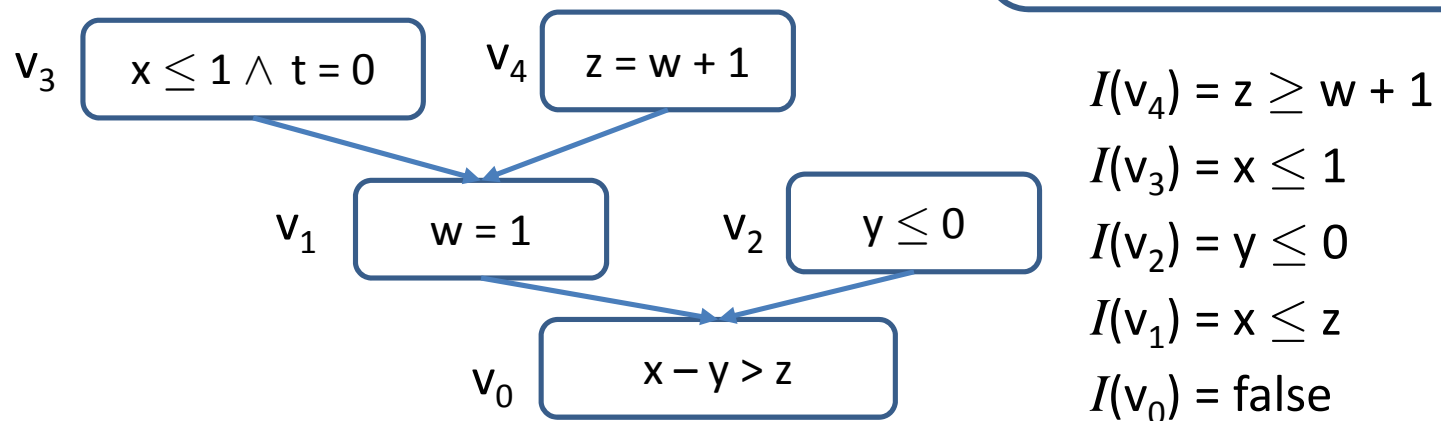
vars. occurring both
in & out of subtree
rooted at v



Tree Interpolation

- Labeled tree (V, E, Θ) - V : nodes E : edges $\Theta: V \rightarrow \text{Preds}(T)$
- $I: V \rightarrow \text{Preds}(T)$ is **tree interpolant (ITP)** of $(V, E, \Theta) \Leftrightarrow$
 - For root $v \in V$, $I(v) = \text{false}$
 - $\forall v \in V. \Theta(v) \wedge (\bigwedge_{(v,v') \in E} I(v')) \Rightarrow I(v)$
 - $\forall v \in V. \text{vars}(I(v)) \subseteq \text{sharedvars}(v)$

vars. occurring both in & out of subtree rooted at v



Reducing Predicate Refinement to Tree Interpolation

Prop: For any π , exists $(V_\pi, E_\pi, \Theta_\pi)$ such that

$$F \vdash \pi \iff \exists I : V_\pi \rightarrow F. I \text{ is ITP of } (V_\pi, E_\pi, \Theta_\pi)$$

Reducing Predicate Refinement to Tree Interpolation

Prop: For any π , exists $(V_\pi, E_\pi, \Theta_\pi)$ such that

$$F \vdash \pi \iff \exists I : V_\pi \rightarrow F. I \text{ is ITP of } (V_\pi, E_\pi, \Theta_\pi)$$

Standard Refinement Algorithm:

- Build $(V_\pi, E_\pi, \Theta_\pi)$
- Infer ITP I of $(V_\pi, E_\pi, \Theta_\pi)$
- Return $\{ \theta \in I(v) \mid v \in \text{dom}(V_\pi) \}$ as inferred predicates

Reducing Predicate Refinement to Tree Interpolation

Prop: For any π , exists $(V_\pi, E_\pi, \Theta_\pi)$ such that

$$F \vdash \pi \iff \exists I : V_\pi \rightarrow F. I \text{ is ITP of } (V_\pi, E_\pi, \Theta_\pi)$$

Standard Refinement Algorithm:

- Build $(V_\pi, E_\pi, \Theta_\pi)$
- Infer ITP I of $(V_\pi, E_\pi, \Theta_\pi)$
- Return $\{ \theta \in I(v) \mid v \in \text{dom}(V_\pi) \}$ as inferred predicates

Plan: Modify tree interpolation algorithm so that

- Infer ITP only from $B_i \cup E_i$
 - i.e., not from entire $\text{Preds}(T)$
- Fail to infer ITP only if there is none in B_i

Modifying Tree Interpolation

Standard tree interpolation algorithm:

- Input: (V, E, Θ)
- 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

Modifying Tree Interpolation

Standard tree interpolation algorithm:

- Input: (V, E, Θ)
- 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

Modified 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 Rules

- $p \in \text{Atoms}(T)$
- $C ::= p \mid \neg p \mid C \vee C$

$$\frac{C \wedge \dots = \Theta(v')}{(V, E, \Theta) \vdash C}$$

$$\frac{(V, E, \Theta) \vdash p \vee C_1 \quad (V, E, \Theta) \vdash \neg p \vee C_2}{(V, E, \Theta) \vdash C_1 \vee C_2}$$

$$\frac{\models_T C}{(V, E, \Theta) \vdash C}$$

Tree Interpolant Generation Rules

- $p \in \text{Atoms}(T)$
- $C ::= p \mid \neg p \mid C \vee C$

$$\frac{C \wedge \dots = \Theta(v')}{(V, E, \Theta) \vdash C : I}$$

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

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

$$I = \lambda v. \begin{cases} I_1(v) \wedge I_2(v) & \text{if } p \in \text{outs}(v) \\ I_1(v) \vee I_2(v) & \text{otherwise} \end{cases}$$

$$\frac{\models_T C}{(V, E, \Theta) \vdash C}$$

Tree Interpolant Generation Rules

- $p \in \text{Atoms}(T)$
- $C ::= p \mid \neg p \mid C \vee C$

$$\frac{C \wedge \dots = \Theta(v')}{(V, E, \Theta) \vdash C : I} \quad I = \lambda v. \begin{cases} \text{false} & \text{if } (v', v) \in E^* \\ \text{true} & \text{otherwise} \end{cases}$$

$$\frac{(V, E, \Theta) \vdash p \vee C_1 : I_1 \quad (V, E, \Theta) \vdash \neg p \vee C_2 : I_2}{(V, E, \Theta) \vdash C_1 \vee C_2 : I} \quad I = \lambda v. \begin{cases} I_1(v) \wedge I_2(v) & \text{if } p \in \text{outs}(v) \\ I_1(v) \vee I_2(v) & \text{otherwise} \end{cases}$$

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

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

Tree Interpolant Generation Rules

- $p \in \text{Atoms}(T)$
- $C ::= p \mid \neg p \mid C \vee C$

$$\frac{C \wedge \dots = \Theta(v')}{(V, E, \Theta) \vdash C : I} \quad I = \lambda v. \begin{cases} \text{false} & \text{if } (v', v) \in E^* \\ \text{true} & \text{otherwise} \end{cases}$$

$$\frac{(V, E, \Theta) \vdash p \vee C_1 : I_1 \quad (V, E, \Theta) \vdash \neg p \vee C_2 : I_2}{(V, E, \Theta) \vdash C_1 \vee C_2 : I} \quad I = \lambda v. \begin{cases} I_1(v) \wedge I_2(v) & \text{if } p \in \text{outs}(v) \\ I_1(v) \vee I_2(v) & \text{otherwise} \end{cases}$$

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

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

Theorem: Inferred ITP is in $B_i^{\wedge \vee}$, and some ITP is inferred if one exists in B_i

So, $E_i = B_i^{\wedge \vee}$

Another Refinement Algorithm

Modified recursion-free Horn-clause solving

Two-phase approach

– Select a few predicate variables

1. Infer restricted solutions to the selected pred. vars.
2. Infer unrestricted solutions to the rest

Only uses expensive finite preds. res. search for few pred. vars.

Becomes relaxed stratification when selection is done in certain way

See ESOP'15 paper for details

Refinement Algorithm Schemas

- We actually propose **algorithm schemas**
 - take other refinement algorithms as modules
 - generate refinement algorithms satisfying requirements of relaxed stratification

See paper for details

Outline

- ✓ Introduction
- 2. Refinement scheme with convergence guarantee
 - ✓ Relaxed Stratification Scheme
 - a. Concrete instances of relaxed stratification
 - ✓ Modified tree interpolation algorithm
 - ✓ Modified rec.-free Horn-clause solving algorithm
 - b. Experiments
- 3. Fast convergence via “small refinements”

Prototype Implementation

new refinement algorithm

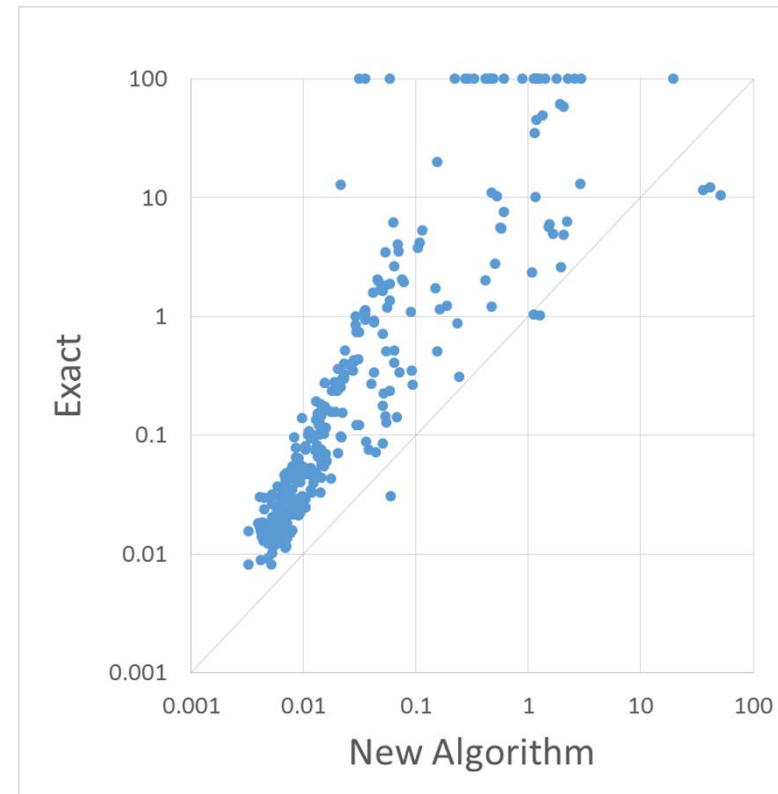
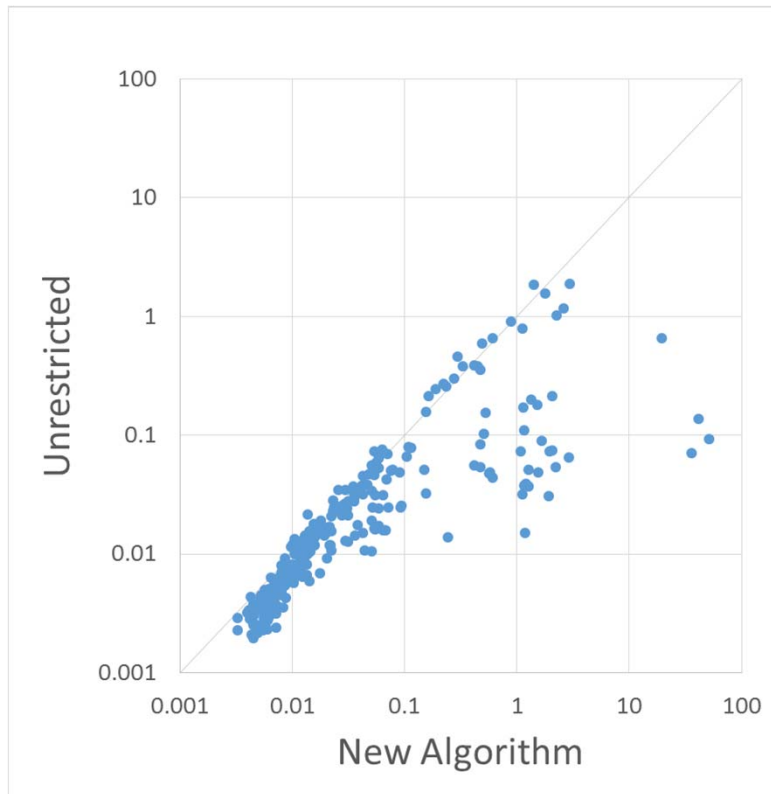
- Algorithm 1 used as module of algorithm 2
- Z3 [1] for template-based constraint solving
- MathSAT5 [2] for unrestricted refinement
- Used as refinement engine of MoChi [3]
 - Safety and termination verifier for higher-order functional programs based on CEGAR

[1] <http://z3.codeplex.com/>

[2] <http://mathsat.fbk.eu/>

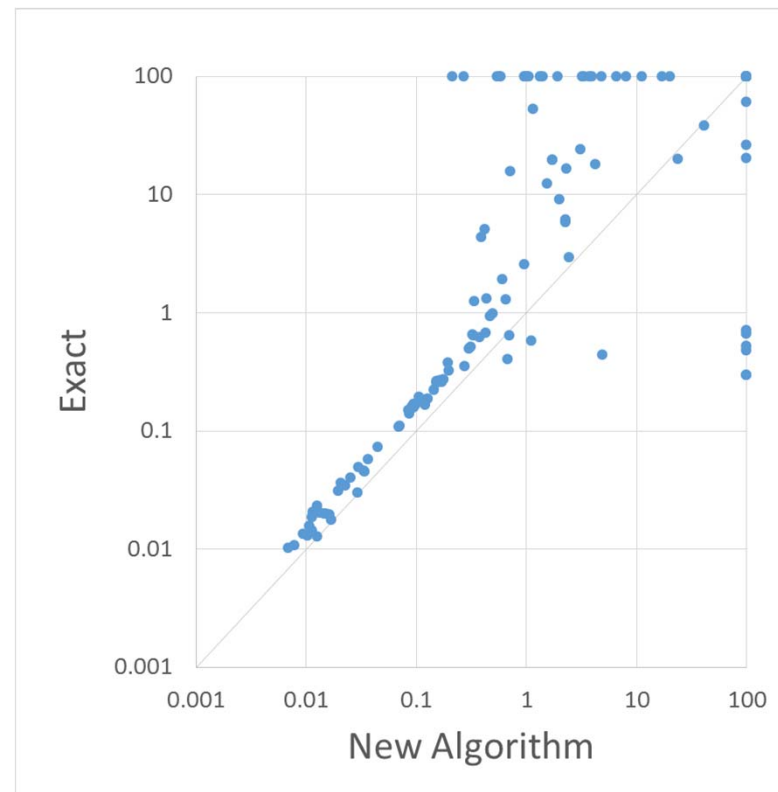
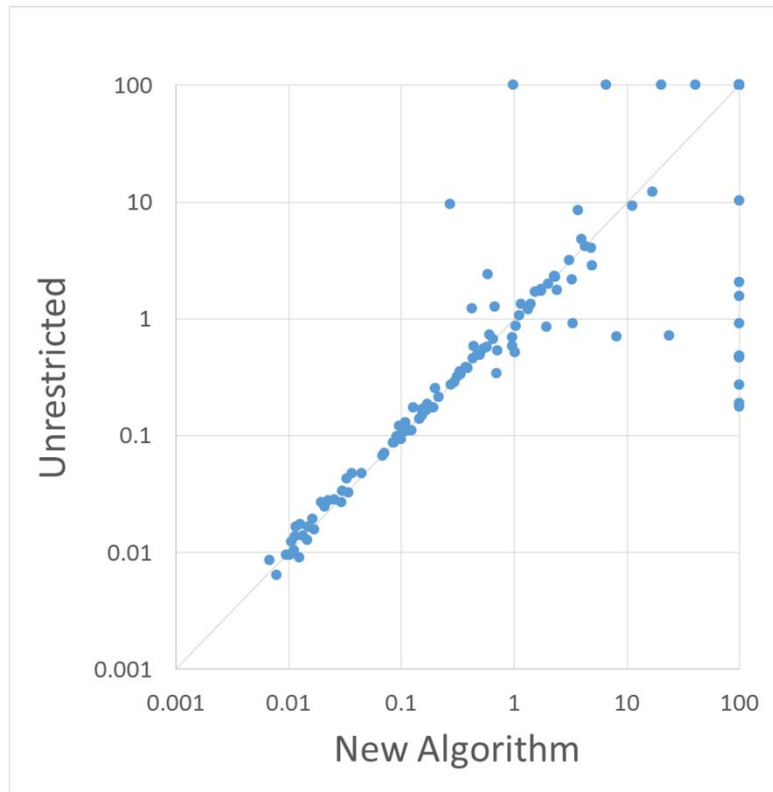
[3] <http://www.kb.is.s.u-tokyo.ac.jp/~ryosuke/mochi/>

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
- 2. Refinement scheme with convergence guarantee
 - ✓ Relaxed Stratification Scheme
 - ✓ Concrete instances of relaxed stratification
 - ✓ Modified tree interpolation algorithm
 - ✓ Modified rec.-free Horn-clause solving algorithm
 - ✓ Experiments
- 3. Fast convergence via “small refinements”

Based on [Terauchi SAS 2015]

FAST CONVERGENCE VIA “SMALL REFINEMENTS”

Talk so far

Predicate refinement in CEGAR

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

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?

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; }
```

Proof of the program:

$\phi_{\text{inv}} \equiv a=b \Rightarrow y=x+z$

π_1 : refuted by $\{ \phi_0, \phi_1, \phi_0 \vee \phi_1 \}$
 π_2 : refuted by $\{ \forall F \mid F \subseteq \{ \phi_0, \phi_1, \phi_2 \} \}$
 \vdots
 π_i : refuted by $\{ \forall F \mid F \subseteq \{ \phi_0, \dots, \phi_i \} \}$
 \vdots

$\pi_i \equiv$ Loops unfolded i times

$\phi_i \equiv x = a \wedge y = b + i \wedge z = i$

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; }
```

π_1 : refuted by $\{ \phi_0, \phi_1, \phi_0 \vee \phi_1 \}$
 π_2 : refuted by $\{ \forall F \mid F \subseteq \{ \phi_0, \phi_1, \phi_2 \} \}$
 \vdots
 π_i : refuted by $\{ \forall F \mid F \subseteq \{ \phi_0, \dots, \phi_i \} \}$
 \vdots

$\pi_i \equiv$ Loops unfolded i times
 $\phi_i \equiv x = a \wedge y = b + i \wedge z = i$

Proof of the program:

$\phi_{\text{inv}} \equiv a=b \Rightarrow y=x+z$

Key Observation: ϕ_{inv} refutes every π_i

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; }
```

π_1 : refuted by $\{ \phi_0, \phi_1, \phi_0 \vee \phi_1 \}$
 π_2 : refuted by $\{ \forall F \mid F \subseteq \{ \phi_0, \phi_1, \phi_2 \} \}$
 \vdots
 π_i : refuted by $\{ \forall F \mid F \subseteq \{ \phi_0, \dots, \phi_i \} \}$
 \vdots

$\pi_i \equiv$ Loops unfolded i times
 $\phi_i \equiv x = a \wedge y = b + i \wedge z = i$

Proof of the program:

$\phi_{inv} \equiv a=b \Rightarrow y=x+z$

Key Observation: ϕ_{inv} refutes every π_i

\therefore Inferring small refinements should hasten convergence
Refinement will infer ϕ_{inv} or some other small proof of program before inferring large ϕ_k 's

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; }
```

π_1 : refuted by $\{ \phi_0, \phi_1, \phi_0 \vee \phi_1 \}$
 π_2 : refuted by $\{ \forall F \mid F \subseteq \{ \phi_0, \phi_1, \phi_2 \} \}$
 \vdots
 π_i : refuted by $\{ \forall F \mid F \subseteq \{ \phi_0, \dots, \phi_i \} \}$
 \vdots

$\pi_i \equiv$ Loops unfolded i times
 $\phi_i \equiv x = a \wedge y = b + i \wedge z = i$

Proof of the program:

$\phi_{\text{inv}} \equiv a=b \Rightarrow y=x+z$

Key Observation: ϕ_{inv} refutes every π_i

How to define “small”?

∴ Inferring small refinements should hasten convergence
Refinement will infer ϕ_{inv} or some other small proof of program before inferring large ϕ_k 's

Proof Size Metric and SRH

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

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

γ : proof 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 Result (Generic)

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

– **Proof: By a simple counting argument**

Convergence Bound Result (Generic)

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

– **Proof: By a simple counting argument**

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

Convergence Bound Result (Generic)

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

– **Proof:** By a 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 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
 - Abstract&Check process uses Cartesian pred. abs.
 - $\text{size}(F) = \sum_{\phi \in F} \text{syntactic_size}(\phi)$

Bound for CFG-represented Programs

- Assumptions
 - Program represented by 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
 - Abstract&Check process uses Cartesian pred. abs.
 - Proof size metric is syntactic
- **Theorem: CEGAR with SRH converges in number of iterations bounded polynomially in $\text{minprfsize}(P)$ for CFG programs.**

See SAS'15 paper for details

Outline

- ✓ Introduction
- 2. Refinement scheme with convergence guarantee
 - ✓ Relaxed Stratification Scheme
 - ✓ Concrete instances of relaxed stratification
 - ✓ Modified tree interpolation algorithm
 - ✓ Modified rec.-free Horn-clause solving algorithm
 - ✓ Experiments
- 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

CEGAR iteration bound results can be taken as negative result?

- 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