

Using Real Relaxations during Program Specialization

F. Fioravanti¹, A. Pettorossi², M. Proietti³, V. Senni²

¹University 'd'Annunzio' of Pescara, Italy

²University of Rome 'Tor Vergata', Italy

³CNR-IASI, Rome, Italy

LOPSTR 2011, Odense, Denmark

Outline

- 1 Introduction
 - Relaxation
 - Reachability Analysis of Infinite-State Systems
- 2 Specialization
 - Specialization Rules on \mathbb{Z}
 - Specialization Strategy
 - Rules Relaxed on \mathbb{R}
- 3 Experiments

Relaxation

Integer Linear Programming

Solution in \mathbb{Z}

relaxation ↓

↑

(Real) Linear Programming

solve
⇒

Solution in \mathbb{R}

Examples of relaxations in Program Analysis:

Approximation

Abstract Interpretation

Reachability Analysis of Infinite-State Systems

Infinite-State System : (I, T, U)

Goal:

Compute the closure T^* and check that $T^*(I) \cap U = \emptyset$

Problem:

$T^*(I)$ may be non-recursive

(I, \xrightarrow{T}, U)



Reachability Analysis of Infinite-State Systems

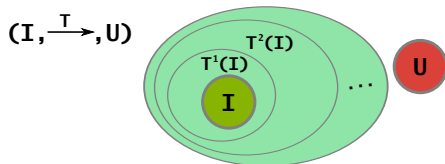
Infinite-State System : (I, T, U)

Goal:

Compute the **closure** T^* and check that $T^*(I) \cap U = \emptyset$

Problem:

$T^*(I)$ may be **non-recursive**



Reachability Analysis of Infinite-State Systems

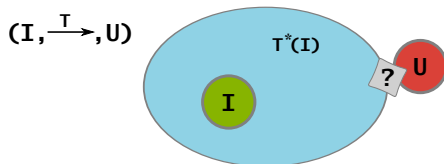
Infinite-State System : (I, T, U)

Goal:

Compute the closure T^* and check that $T^*(I) \cap U = \emptyset$

Problem:

$T^*(I)$ may be non-recursive



Reachability Analysis of Infinite-State Systems

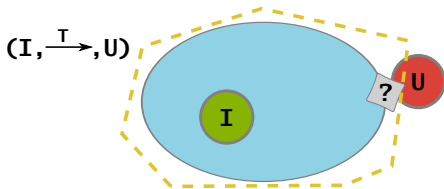
Infinite-State System : (I, T, U)

Goal:

Compute the closure T^* and check that $T^*(I) \cap U = \emptyset$

Problem:

$T^*(I)$ may be non-recursive



$\text{approximate}(T^*)(I) \cap U = \emptyset$ ☺

$\text{approximate}(T^*)(I) \cap U \neq \emptyset$??

An Example where Relaxation fails

Counter System: interpreted on \mathbb{Z}

- T:
- (1) $\langle X, Y \rangle \longrightarrow \langle X, Y-1 \rangle$ if $X \geq 1$
 - (2) $\langle X, Y \rangle \longrightarrow \langle X, Y+2 \rangle$ if $X \leq 2$
 - (3) $\langle X, Y \rangle \longrightarrow \langle X, -1 \rangle$ if $\exists V \in \mathbb{Z} \ Y = 2V+1$

An Example where Relaxation fails

Counter System: interpreted on \mathbb{Z}

- T:
- (1) $\langle X, Y \rangle \rightarrow \langle X, Y-1 \rangle$ if $X \geq 1$
 - (2) $\langle X, Y \rangle \rightarrow \langle X, Y+2 \rangle$ if $X \leq 2$
 - (3) $\langle X, Y \rangle \rightarrow \langle X, -1 \rangle$ if $\exists V \in \mathbb{Z} \ Y = 2V+1$

to show:

$$T^*(\langle 0, 0 \rangle) \cap \{\langle X, Y \rangle \mid Y < 0\} = \emptyset$$

An Example where Relaxation fails

Counter System: interpreted on \mathbb{Z}

$$\begin{array}{l}
 \text{T: } (1) \langle X, Y \rangle \longrightarrow \langle X, Y-1 \rangle \quad \text{if } X \geq 1 \\
 (2) \langle X, Y \rangle \longrightarrow \langle X, Y+2 \rangle \quad \text{if } X \leq 2 \\
 (3) \langle X, Y \rangle \longrightarrow \langle X, -1 \rangle \quad \text{if } \exists V \in \mathbb{R} \ Y = 2V+1
 \end{array}$$

to show: $T^*(\langle 0, 0 \rangle) \cap \{\langle X, Y \rangle \mid Y < 0\} = \emptyset$

relax: $\text{relaxed}_{\mathbb{R}}(T)^*(\langle 0, 0 \rangle) \cap \{\langle X, Y \rangle \mid Y < 0\} \neq \emptyset \quad ??$

The system **cannot be proved safe** after relaxation,
no matter the technique we use

Alternative approach: Specialization

Transition System S

↓

$P_S \in \text{CLP}(\mathcal{D})$

specialize
 \implies

Specialized System $\text{Spec}S$

↑

$\text{Spec}P_S \in \text{CLP}(\mathcal{D})$

Alternative approach: Specialization

Transition System S

↓

$P_S \in \text{CLP}(\mathcal{D})$

$M_{\mathcal{D}}(P_S)$

specialize
 \implies

=

Specialized System $\text{Spec}S$

↑

$\text{Spec}P_S \in \text{CLP}(\mathcal{D})$

$M_{\mathcal{D}}(\text{Spec}P_S)$

Alternative approach: Specialization

Transition System S

Specialized System $SpecS$

$$\begin{array}{ccc}
 \downarrow & & \uparrow \\
 P_S \in \text{CLP}(\mathcal{D}) & \xRightarrow{\text{specialize}} & SpecP_S \in \text{CLP}(\mathcal{D}) \\
 M_{\mathcal{D}}(P_S) & = & M_{\mathcal{D}}(SpecP_S)
 \end{array}$$

Advantages of analyzing $SpecS$: [FPPS10]

- Simplification w.r.t. the set of the **initial/unsafe** states
- Computation of **invariants** while specializing
- **No loss of precision** (i.e., **we can apply other techniques**)

Equivalence Preservation Under Relaxation

Let us fix the domain \mathcal{D} to \mathbb{Z}

Specialization: $P_S \Rightarrow P_1 \Rightarrow \dots \Rightarrow P_n = \text{Spec}P_S$

a sequence of transformation steps whose **conditions are on \mathbb{Z}**

Idea: **Relax** (on \mathbb{R}) transformation rules **conditions**
rather than **program semantics**

Outcome: equivalence & advantages of relaxation

We can reuse efficient and scalable solvers on \mathbb{R}

Specialization

Specialization

- **Program Specialization:** a transformation technique for adapting a given program to a specific **context** of use

- **Rule-Based**

Entailment between constraints

- $c \sqsubseteq_{\mathbb{Z}} d$ iff $\mathbb{Z} \models \forall (c \rightarrow d)$

- $c \sqsubseteq_{\mathbb{R}} d$ iff $\mathbb{R} \models \forall (c \rightarrow d)$

Specialization Rules

Definition

$$\mapsto \text{newp}(X_1, \dots, X_h) \leftarrow c \wedge p(X_1, \dots, X_h)$$

Unfolding

| | | |
|---|--|---|
| $ \begin{array}{l} K \leftarrow d_1 \wedge B_1 \\ \vdots \\ K \leftarrow d_n \wedge B_n \end{array} $ | $ H \leftarrow c \wedge K \wedge R \mapsto $ | $ \begin{array}{l} H \leftarrow c \wedge d_1 \wedge B_1 \wedge R \\ \vdots \\ H \leftarrow c \wedge d_n \wedge B_n \wedge R \end{array} $ |
|---|--|---|

Folding

| | | |
|---------------------------|--|------------------------------------|
| $K \leftarrow d \wedge B$ | $H \leftarrow c \wedge B \wedge R \mapsto$ | $H \leftarrow c \wedge K \wedge R$ |
|---------------------------|--|------------------------------------|

$c \sqsubseteq_{\mathbb{Z}} d$

Clause Removal

1.
$$\begin{array}{l}
 H \leftarrow c \wedge B \\
 H \leftarrow d
 \end{array}
 \mapsto H \leftarrow d \quad \text{if } c \sqsubseteq_{\mathbb{Z}} d$$
2.
$$H \leftarrow c \wedge B \mapsto \emptyset \quad \text{if } c \text{ is unsatisfiable in } \mathbb{Z}$$

Specialization Strategy

Input: P and a clause $\delta_0: p_{sp}(X_1, \dots, X_h) \leftarrow c \wedge p(X_1, \dots, X_h)$

Output: $SpecP$ s.t. $\forall z \in \mathbb{Z}^h (p_{sp}(z) \in M_{\mathbb{Z}}(P) \text{ iff } p_{sp}(z) \in M_{\mathbb{Z}}(SpecP))$

$SpecP := \emptyset;$

$Defs := \{\delta_0\};$

while $\exists \delta \in Defs$ **do**

$Unfold \delta \quad \mapsto \Gamma$

$ClauseRemoval \Gamma \mapsto \Delta$

$Generalize\&Fold \Delta \mapsto (\Phi, NewDefs)$

$Defs := (Defs - \{\delta\}) \cup NewDefs$

$SpecP := SpecP \cup \Phi$

od

Specialization Strategy

Input: P and a clause $\delta_0: p_{sp}(X_1, \dots, X_h) \leftarrow \underline{c \wedge p(X_1, \dots, X_h)}$

Output: $SpecP$ s.t. $\forall z \in \mathbb{Z}^h (p_{sp}(z) \in M_{\mathbb{Z}}(P) \text{ iff } p_{sp}(z) \in M_{\mathbb{Z}}(SpecP))$

$SpecP := \emptyset;$

$Defs := \{\delta_0\};$

while $\exists \delta \in Defs$ **do**

$Unfold \delta \quad \mapsto \Gamma$

$ClauseRemoval \Gamma \mapsto \Delta$

$Generalize\&Fold \Delta \mapsto (\Phi, NewDefs)$

$Defs := (Defs - \{\delta\}) \cup NewDefs$

$SpecP := SpecP \cup \Phi$

od

Specialization Strategy

Input: P and a clause $\delta_0: p_{sp}(X_1, \dots, X_h) \leftarrow \underline{c \wedge p(X_1, \dots, X_h)}$

Output: $SpecP$ s.t. $\forall z \in \mathbb{Z}^h (p_{sp}(z) \in M_{\mathbb{Z}}(P) \text{ iff } p_{sp}(z) \in M_{\mathbb{Z}}(SpecP))$

$SpecP := \emptyset;$

$Defs := \{\delta_0\};$

while $\exists \delta \in Defs$ **do**

$Unfold \delta \quad \mapsto \Gamma$ (Propagate Context)

$ClauseRemoval \Gamma \mapsto \Delta$ (Simplify)

$Generalize\&Fold \Delta \mapsto (\Phi, NewDefs)$ (Apply Induction)

$Defs := (Defs - \{\delta\}) \cup NewDefs$

$SpecP := SpecP \cup \Phi$

od

Specialization Strategy

Input: P and a clause $\delta_0: p_{sp}(X_1, \dots, X_h) \leftarrow \underline{c \wedge p(X_1, \dots, X_h)}$

Output: $SpecP$ s.t. $\forall z \in \mathbb{Z}^h (p_{sp}(z) \in M_{\mathbb{Z}}(P) \text{ iff } p_{sp}(z) \in M_{\mathbb{Z}}(SpecP))$

$SpecP := \emptyset;$

$Defs := \{\delta_0\};$

while $\exists \delta \in Defs$ **do**

$Unfold \delta \quad \mapsto \Gamma$ (Initial/Unsafe States)

$ClauseRemoval \Gamma \mapsto \Delta$

$Generalize\&Fold \Delta \mapsto (\Phi, NewDefs)$ (Invariants)

$Defs := (Defs - \{\delta\}) \cup NewDefs$

$SpecP := SpecP \cup \Phi$

od

Weakening Rules Conditions

Theorem (1)

$\forall z \in \mathbb{Z}^h \quad \rho_{sp}(z) \in M_{\mathbb{Z}}(P) \text{ iff } \rho_{sp}(z) \in M_{\mathbb{Z}}(\text{Spec}P)$

Lemma

if c is \mathbb{Z} -satisfiable then c is \mathbb{R} -satisfiable

if $c \sqsubseteq_{\mathbb{R}} d$ then $c \sqsubseteq_{\mathbb{Z}} d$

Theorem (2)

if we replace \mathbb{Z} -satisfiability by \mathbb{R} -satisfiability and $c \sqsubseteq_{\mathbb{Z}} d$ by $c \sqsubseteq_{\mathbb{R}} d$ then Theorem (1) still holds

Example - Continued

Counter System:

- | | | |
|-----|---|--|
| (1) | $\langle X, Y \rangle \longrightarrow \langle X, Y-1 \rangle$ | if $X \geq 1$ |
| (2) | $\langle X, Y \rangle \longrightarrow \langle X, Y+2 \rangle$ | if $X \leq 2$ |
| (3) | $\langle X, Y \rangle \longrightarrow \langle X, -1 \rangle$ | if $\exists V \in \mathbb{Z} \ Y = 2V+1$ |

Context: initial state $\langle 0, 0 \rangle$

$$ts_{sp}(X, Y) \leftarrow X=0 \wedge Y=0 \wedge ts(X', Y')$$

By specialization

$$ts_{sp}(X, Y) \leftarrow X=0 \wedge Y=0 \wedge X'=0 \wedge Y'=2 \wedge new1(X', Y')$$

$$ts_{sp}(X, Y) \leftarrow X=0 \wedge Y=0 \wedge Y=2Z+1 \wedge X'=0 \wedge Y'=-1 \wedge new1(X', Y')$$

SpecP:

$$new1(X, Y) \leftarrow X=0 \wedge X'=0 \wedge Y'=Y+2 \wedge new1(X', Y')$$

$$new1(X, Y) \leftarrow X=0 \wedge Y=2Z+1 \wedge X'=0 \wedge Y'=-1 \wedge new1(X', Y')$$

$$new1(X, Y) \leftarrow X=0 \wedge Y < 0$$

The computation of the least \mathbb{Z} -model $M_{\mathbb{Z}}(\text{SpecP})$ terminates

Since $M_{\mathbb{Z}}(\text{SpecP})$ does not contain $ts_{sp}(0, 0)$, the system is proved safe

On Generalizations and Other Operators

Generalization, widening, and approximation operators on \mathbb{R} play an important role in Program Analysis [CH78]

When we move from \mathbb{R} to \mathbb{Z} . . .

- We need to develop new operators
- Complexity is an issue

In the specialization-based approach we can **reuse the operators on \mathbb{R}**

Experiments

| EXAMPLES | default | | A | | F | |
|-----------------------------------|----------|-------|----------|---------|----------|----------|
| | Sys | SpSys | Sys | SpSys | Sys | SpSys |
| Bakery2 | 0.03 | 0.05 | 0.03 | 0.05 | 0.06 | 0.04 |
| Bakery3 | 0.70 | 0.25 | 0.69 | 0.25 | ∞ | 3.68 |
| MutAst | 1.46 | 0.37 | 1.00 | 0.37 | 0.22 | 0.59 |
| Peterson | 56.49 | 0.10 | ∞ | 0.10 | ∞ | 13.48 |
| Ticket | ∞ | 0.03 | 0.10 | 0.03 | 0.02 | 0.19 |
| Berkeley RISC | 0.01 | 0.04 | \perp | 0.04 | 0.01 | 0.02 |
| DEC Firefly | 0.01 | 0.02 | \perp | 0.03 | 0.01 | 0.07 |
| IEEE Futurebus | 0.26 | 0.68 | \perp | \perp | ∞ | ∞ |
| Illinois University | 0.01 | 0.03 | \perp | 0.03 | ∞ | 0.07 |
| Barber | 0.62 | 0.21 | \perp | 0.21 | ∞ | 0.08 |
| CSM | 56.39 | 7.69 | \perp | 7.69 | ∞ | 125.32 |
| Consistency | ∞ | 0.11 | \perp | 0.11 | ∞ | 324.14 |
| Insertion Sort | 0.03 | 0.06 | 0.04 | 0.06 | 0.18 | 0.02 |
| Selection Sort | ∞ | 0.21 | \perp | 0.21 | ∞ | 0.33 |
| Reset Petri Net | ∞ | 0.02 | \perp | \perp | ∞ | 0.01 |
| Train | 42.24 | 59.21 | \perp | \perp | ∞ | 0.46 |
| <i>No. of verified properties</i> | 12 | 16 | 5 | 13 | 6 | 15 |

Verification times using the ALV [YKB09] system, based on the Omega library.

' \perp ' = 'Unable to verify' and ' ∞ ' = 'Timeout' (10 minutes).

Advertisement

An implementation in SICStus Prolog as a module of our MAP system.

Give it a try!

MAP - Specialization-Based Reachability Analysis of Infinite-State Transition Systems

| 1. Program Uploading | 2. Options Selection | 3. Specialization | 4. Perfect Model |
|--|----------------------|--|------------------|
| <pre> % Bakery Protocol 2 processes - safety [Delzanno-Pedolski, 2001] % % Transitions t((s(A,S,B),s(w,D,S,B)) :- D=>B!, A=>0, B=>0. t((w,A,S,B),s(s,A,S,B)) :- B!, A=>0. t((w,A,S,B),s(s,A,S,B)) :- B=>0, A=>0. t((s(A,S,B),s(t,D,S,B)) :- D=>0, A=>0, B=>0. t((s,A,t,B),s(s,A,w,D)) :- D=>0, B=>0. t((s,A,w,B),s(s,A,u,B)) :- B!, B=>0. t((s,A,w,B),s(s,A,u,B)) :- A=>0, B=>0. t((s,A,u,B),s(s,A,t,D)) :- D=>0, B=>0, A=>0. % Elementary Properties e1ee((s,u,A,u,B),unsafe) :- A=>0, B=>0. e1ee((t,A,t,C),initial) :- A=>0, C=>0. e1ee((w,A,w,A),initial) :- A=>0. % Temporal Properties lev1 :- unreachable(backward,initial,unsafe). </pre> | | <p>Specialization Options:</p> <p>Invariant: <input type="text" value="true"/></p> <p>Timeout: <input type="text" value="10 s"/></p> <p><input type="radio"/> Default <input checked="" type="radio"/> Custom</p> <p>Generalization Parameters:</p> <p>MaxCoeff: <input type="text" value="off"/></p> <p>Firing Relation: <input type="text" value="variant"/></p> <p>Gen. Oper.: <input type="text" value="eiden"/></p> <p>Gen. Param.: <input type="text" value="e_req_maximum"/></p> <p>Polyvariance Parameters:</p> <p>Partitioning: <input type="text" value="single"/></p> <p>Include Foldable: <input type="text" value="include"/></p> <p>Candidate: <input type="text" value="w.r.t. ancestor"/></p> <p>Post-Folding: <input type="text" value="most general"/></p> <p>Specialize Help</p> | |

<http://map.uniroma2.it/mapweb/>

Thanks to J. Gallagher for giving us the CHA interface

References



Patrick Cousot and Nicolas Halbwachs.

Automatic discovery of linear restraints among variables of a program.

In *POPL*, pages 84–96, 1978.



Fabio Fioravanti, Alberto Pettorossi, Maurizio Proietti, and Valerio Senni.

Program specialization for verifying infinite state systems: An experimental evaluation.

In María Alpuente, editor, *LOPSTR*, volume 6564 of *Lecture Notes in Computer Science*, pages 164–183. Springer, 2010.



Tuba Yavuz-Kahveci and Tefvik Bultan.

Action language verifier: an infinite-state model checker for reactive software specifications.

Formal Methods in System Design, 35(3):325–367, 2009.