Code Analysis with Local Policy Iteration

George Karpenkov and David Monniaux

Problem

Problem: automated software verification, prove the *error* property unreachable.
Safety proof is done by *induction* — finding an *inductive invariant*, for which initial and consecution condition hold. Once we are in the invariant, we are guaranteed to stay there. We look for the *separating* inductive invariant, s.t. bad state $P(X)$ is not reachable from $I(X)$ by following the transition $\tau(X, X')$.

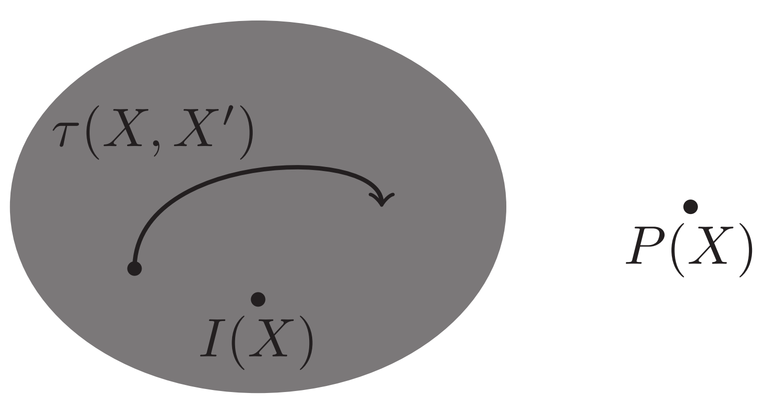


Figure 1: Separating Inductive Invariant

Abstract Interpretation

A usual tool for generating inductive invariants is *abstract interpretation — interpreting* the program in the *abstract domain* of choice. Memory locations instead of assuming *concrete* values assume *abstract* values. *Template constraints domain* fixes in advance a set of interesting linear expressions over program variables (e.g. `i, i+x, i-x`) called *templates*. Abstract value is propagated using convex maximization. Abstract values $i \leq 4$ under the increment operation `i := i + 1` becomes $i \leq 5$.

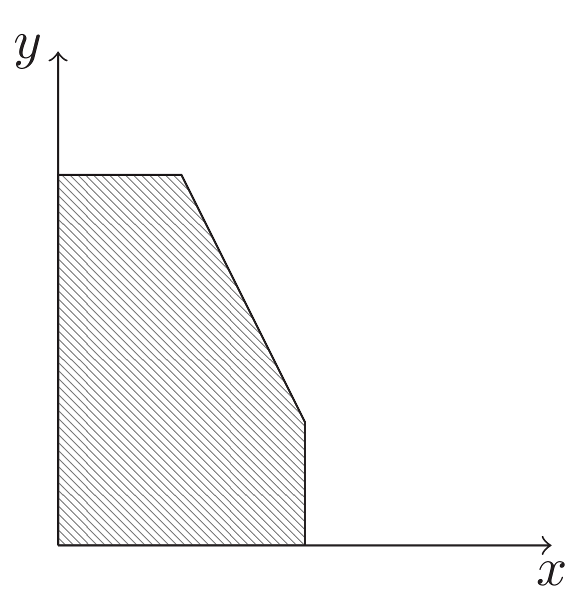


Figure 2: $x \leq 4 \wedge y \leq 4 \wedge x + y \leq 4$

References

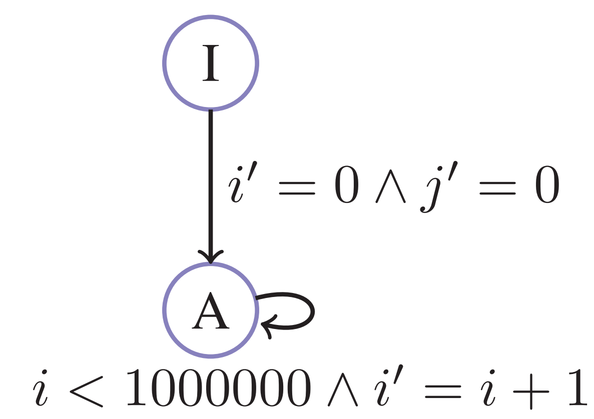
[1] T. Gawlitza, D. Monniaux Invariant Generation through Strategy Iteration in Succinctly Represented Control Flow Graphs In Logical Methods in Computer Science, 2012.

[2] T. Gawlitza, H. Seidl Precise Relational Invariants Through Strategy Iteration In CSL, 2007.

Policy Iteration Approach

Abstract interpretation relies on *widening* to enforce convergence, which is often imprecise. *Policy Iteration guarantees* to find the least *inductive* invariant in the given abstract domain. Program is represented as a set of *equations*.

```
int i=0;
while (i<1000000)
i++;
```



$$h = (\max i' \text{ s.t. } i' = 0) \\ \vee\, h = (\max i' \text{ s.t. } \\ i' = i + 1 \\ \wedge\, i < 10000000 \\ \wedge\, i \leq h)$$

This fixpoint equation system is solved for h by considering different possible arguments for disjunction:

(i) $h = (\max i' \text{ s.t. } i' = 0) = 0$, which is not inductive, since one can iterate from $i = 0$ to $i = 1$.

(ii) $h = \max i' \text{ s.t. } i' = i + 1 \wedge i < 1000000 \wedge i \leq h$, which has two solutions: $h = -\infty$ (representing unreachable state, discarded) and $h = 1000000$, which is inductive.

Contribution

Despite strong optimality claims, policy iteration remained quite obscure compared to standard Kleene (fixpoint) iteration techniques. Some reasons for these are:

- Global view of the program is required to construct the equation system, which may be prohibitive for very large programs.
- Collaboration is difficult due to heavy pre-processing.

We present new *Local-Policy-Iteration* algorithm (LPI) for computing inductive invariants using policy iteration. Open-source implementation is available at `http://metaworld.me/lpi/`. This is the first policy-iteration implementation that is capable of dealing with C code. Our solution improves on earlier max-policy approaches in the following ways:
(i) Scalability LPI constructs optimization queries that are at most of the size of the largest loop in the program. At every step we only solve the optimization problem necessary for deriving the *local* candidate invariant.
(ii) Ability to cooperate with other analyses LPI is defined within the Configurable Program Analysis framework, which is designed to allow easy inter-analysis collaboration. Expressing policy iteration as a fixpoint-propagation algorithm allows invariant exchange with other analyses.

Results

Evaluation is done on "Loops" category of SV-Comp'15. We compare LPI with: BLAST (lazy abstraction), PAGAI (abstract intepretation), and CPAchecker (ensemble of different techniques).