PostHat and All That: Attaining Most-Precise Inductive Invariants*

Aditya Thakur[1], Akash Lal[2], Junghee Lim[1], and Thomas Reps[1,3]

[1] University of Wisconsin; Madison, WI, USA {adi,junghee,reps}@cs.wisc.edu
[2] Microsoft Research India; Bangalore; India. akashl@microsoft.com
[3] GrammaTech, Inc.; Ithaca, NY, USA

Abstract. In abstract interpretation, the choice of an abstract domain fixes a limit on the precision of the inductive invariants that one can express; however, for a given abstract domain $\mathcal{A}$, there is a *most-precise* ("strongest", "best") inductive $\mathcal{A}$-invariant for each program. Many techniques have been developed in abstract interpretation for finding *over-approximate* solutions, but only a few algorithms have been given that can achieve the fundamental limits that abstract-interpretation theory establishes. In this paper, we present an algorithm that solves the following problem:

Given program P, an abstract domain $\mathcal{A}$, and access to an SMT solver, find the most-precise inductive $\mathcal{A}$-invariant for P.

1 Introduction

This paper continues our investigation of ways to harness the power of SMT solvers to obtain algorithms for fundamental operations in abstract interpretation. In particular, we present an algorithm that solves the following problem:

Given program P, an abstract domain $\mathcal{A}$, and access to an SMT solver, find the most-precise inductive $\mathcal{A}$-invariant for P.

The method provides a systematic method for obtaining *most-precise loop invariants* and *procedure summaries*. In addition to providing insight on fundamental limits in abstract interpretation, the method also performs well.

The Role of Best Abstract Transformers. The choice of an abstract domain fixes a limit on how precisely the statements of a program can be over-approximated: for a given abstract domain, there is a single most-precise answer. For instance, suppose that one has a Galois connection $\mathcal{C} \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \mathcal{A}$ between concrete domain $\mathcal{C}$ and abstract domain $\mathcal{A}$. Then the best abstract transformer

* Supported, in part, by NSF under grants CCF-{0810053, 0904371}; by ONR under grants N00014-{09-1-0510, 11-C-0447}; by ARL under grant W911NF-09-1-0413; by AFRL under grants FA9550-09-1-0279 and FA8650-10-C-7088; and by DARPA under cooperative agreement HR0011-12-2-0012. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors, and do not necessarily reflect the views of the sponsoring agencies.

(BAT) for a concrete transformer τ, denoted by $\widehat{\text{Post}}[\tau] : \mathcal{A} \to \mathcal{A}$, is the most-precise abstract operator possible, given $\mathcal{A}$, that over-approximates the concrete operator $\text{Post}[\tau] : \mathcal{C} \to \mathcal{C}$. $\widehat{\text{Post}}$ can be expressed as follows [5]:

$$\widehat{\text{Post}}[\tau] = \alpha \circ \text{Post}[\tau] \circ \gamma. \tag{1}$$

Eqn. (1) defines the limit of precision obtainable using abstraction $\mathcal{A}$.

We will distinguish between two different goals suggested by Eqn. (1):

1. Given a concrete transformer τ and an abstract value $a \in \mathcal{A}$, *apply* the best abstract transformer $\widehat{\text{Post}}[\tau]$ to a (i.e., compute $\widehat{\text{Post}}[\tau](a)$).
2. Given a concrete transformer τ, *obtain an explicit representation* of the best abstract transformer $\widehat{\text{Post}}[\tau]$.

Unfortunately, Eqn. (1) is non-constructive; it does not provide an *algorithm*, either for applying $\widehat{\text{Post}}[\tau]$ or for finding a representation of the function $\widehat{\text{Post}}[\tau]$. In particular, in many cases, the application of γ to an abstract value would yield an intermediate result—a set of concrete states—that is either infinite or too large to fit in computer memory.

Nevertheless, several algorithms have been developed for problems (1) and (2) [10, 16, 17, 24, 18], and the two problems have received increasing attention in recent years [12, 4, 6, 15, 22, 20]. A common feature in much of this work is the interplay between logic and abstract interpretation, which is formalized as the operation of *symbolic abstraction* [17]: the symbolic abstraction of a formula φ in logic $\mathcal{L}$, denoted by $\widehat{\alpha}(\varphi)$, returns the best value in $\mathcal{A}$ that over-approximates the meaning of φ. Note that $\widehat{\alpha}(\varphi)$ is thus the strongest consequence of φ that can be expressed in $\mathcal{A}$. In particular, to solve problem (2), encode the semantics of τ as a logical formula φ_τ, and return $\widehat{\alpha}(\varphi_\tau)$.

The fundamental importance of the BAT problem is underscored by the following observation:

Observation 1. *Let program P consist of (i) nodes $N = \{n_i\}$, (ii) edges $E_P = \{n_i \to n_j\}$, and (iii) a concrete-state transformer $\tau_{i,j}$ associated with each edge $n_i \to n_j$. Let $\mathcal{A}$ be an abstract domain. The **best inductive invariant** (BII) for P that is expressible in $\mathcal{A}$ is the least fixed-point of the equation system*

$$V_1 = a_1 \qquad V_j = \bigsqcup_{n_i \to n_j \in E_P} \widehat{\text{Post}}[\tau_{i,j}](V_i), \tag{2}$$

where a_1 is the best value in $\mathcal{A}$ that over-approximates the set of allowed input states at the enter node n_1.

The Best-Inductive-Invariant Problem. Putting together Obs. 1 with the papers cited above, the current state of the art is that in some situations it is possible to obtain best inductive $\mathcal{A}$-invariants using known techniques for the BAT problem. However, as we know from the huge literature on program-analysis methods, just because we have one method in hand for solving a problem, that is typically not the end of the story. On the contrary, it is generally important to explore the design space of alternatives, e.g., for efficient special cases or for general-purpose improvements. We believe that this situation holds true for the BII problem, and thus have been investigating methods for solving Eqn. (2).

In addition, there has also been some work on the BII problem directly. For instance, the Houdini algorithm [8] solves the BII problem for a restricted family of predicate-abstraction domains. Yorsh et al. [23] created a framework for BII, which they instantiated for two different abstract-interpretation problems.

In some sense, such work on BII came too early—i.e., before the advent of modern SMT solvers, which are (i) fast and (ii) able to create a satisfying model for a satisfiable formula. Moreover, our experience in talking with others about BII and related problems is that the community under-appreciates the power and utility that a solution to BII provides. One of the goals of this paper is re-examine the BII problem and proselytize for it being given greater attention. A second goal is to bring some new ideas to bear on the problem.

Contributions. Our work makes the following contributions:

– We re-examine the BII problem, taking advantage of (i) modern SMT solvers, and (ii) recently developed "anytime algorithms" for symbolic abstraction that combine under-approximations with over-approximations, and can return a nontrivial (non-$\top$) value in case of a timeout [22, 20]. We develop such algorithms for BII: they over-approximate BII in general, but achieve BII when no timeout occurs (§2 and §3).
– We present a solution for the BII problem for multi-procedure programs (§3).
– In addition to BII *per se*, our algorithms have applications in creating loop summaries and procedure summaries.
– We created two concrete implementations of the technique (§4). One is based on Boogie [2], a program-verification framework that generates and discharges classical verification conditions. The other is based on WALi [11], a tool for solving program-analysis problems using an abstract domain.
– Not only does the work provide insight on fundamental limits in abstract interpretation, the algorithms that we present are also practical. For instance, for 19 examples for which the Corral model checker gave an indefinite result using Houdini-supplied invariants [8], invariants discovered using BII allowed Corral to give a definite result in 9 cases (47%); see §4.

§5 discusses related work. Proofs are given in App. A.

2 Basic Insights

Fig. 1(a) shows an example program that we will use to illustrate finding the best inductive affine-equality invariant. We concentrate on lines (1), (6), and (12). Fig. 1(b) depicts the dependences in the equation system over node-variables $\{V_1, V_6, V_{12}\}$. Fig. 1(c) gives formulas for the transition relations among $\{V_1, V_6, V_{12}\}$. The remainder of this section illustrates how to solve the BII problem for the following equation system, which corresponds to Figs. 1(b) and 1(c):

$$V_1 = \top \qquad V_6 = \widehat{\mathrm{Post}}[\tau_{1,6}](V_1) \sqcup \widehat{\mathrm{Post}}[\tau_{6,6}](V_6) \qquad V_{12} = \widehat{\mathrm{Post}}[\tau_{6,12}](V_6)$$

It is convenient to rewrite these equations as

$$V_1 = \top \quad V_6 = V_6 \sqcup \widehat{\mathrm{Post}}[\tau_{1,6}](V_1) \sqcup \widehat{\mathrm{Post}}[\tau_{6,6}](V_6) \quad V_{12} = V_{12} \sqcup \widehat{\mathrm{Post}}[\tau_{6,12}](V_6) \qquad (3)$$

(a)

```
(1)  // Initialize
(2)  a = read_input();
(3)  b = a;
(4)  x = 0;
(5)  y = 0;
(6)  while (*) {   // Loop invariant: a==b && x==y
(7)      a = a+2;
(8)      b = (x==y) ? b+2 : read_input();
(9)      x = x+1;
(10)     y = (a==b) ? y+1 : read_input();
(11) }
(12) . . .  // Exit invariant: a==b && x==y
```

(b)

$$\begin{tikzcd}
V_1 \arrow[d, "\tau_{1,6}"] \\
V_6 \arrow[loop right, "\tau_{6,6}"] \arrow[d, bend right, "\tau_{6,12}"'] \\
V_{12}
\end{tikzcd}$$

(c)

$$\tau_{1,6} \overset{\text{def}}{=} b' = a' \wedge x' = 0 \wedge y' = 0$$

$$\tau_{6,6} \overset{\text{def}}{=} \begin{pmatrix} a' = a + 2 \\ \wedge\ (x = y) \Rightarrow (b' = b + 2) \\ \wedge\ x' = x + 1 \\ \wedge\ (a' = b') \Rightarrow (y' = y + 1) \end{pmatrix}$$

$$\tau_{6,12} \overset{\text{def}}{=} a' = a \wedge b' = b \wedge x' = x \wedge y' = y$$

Fig. 1: (a) Example program. (b) Dependences among node-variables in the program's equation system (over node-variables $\{V_1, V_6, V_{12}\}$). (c) The transition relations among $\{V_1, V_6, V_{12}\}$ (expressed as formulas).

Solving the BII Problem from Below. In the most basic approach to solving the BII problem, we assume that we have an essentially standard fixed-point solver that performs chaotic iteration. The method will create successively better under-approximations to the solution, until it finds a fixed point, which will also be the best inductive invariant. We illustrate the algorithm on Eqn. (3).

What is special, compared to standard equation solvers, is that each application of the right-hand side of an equation in Eqn. (3)—defined by the corresponding formula in Fig. 1(c)—is given the *best-transformer interpretation* by means of a function $\widehat{\text{Post}}$ for applying the best abstract transformer. That is, $\widehat{\text{Post}}$ solves problem (1) from §1, and satisfies $\widehat{\text{Post}}[\tau](a) = (\alpha \circ \text{Post}[\tau] \circ \gamma)(a)$. A specific instance of $\widehat{\text{Post}}$ is the function $\widehat{\text{Post}}^{\uparrow}$, given as Alg. 1.

Each call to $\widehat{\text{Post}}^{\uparrow}[\tau](v)$ performs a successive-approximation process, working up from $\bot$ (line 2), to identify v' such that $v' = (\alpha \circ \text{Post}[\tau] \circ \gamma)(v)$. $\widehat{\text{Post}}^{\uparrow}$ imposes certain requirements: (i) abstract domain $\mathcal{A}$ must be a join semi-lattice with a least element $\bot$, and have no infinite ascending chains; (ii) logic $\mathcal{L}$ must be closed under $\wedge$ and $\neg$; and (iii) certain operations must be available:

– The operation of *symbolic concretization* (line 5 of Alg. 1), denoted by $\widehat{\gamma}$, maps an abstract value $a \in \mathcal{A}$ to a formula $\widehat{\gamma}(a) \in \mathcal{L}$ such that a and $\widehat{\gamma}(a)$ represent the same set of concrete states (i.e., $\gamma(a) = [\![\widehat{\gamma}(a)]\!]$, where $[\![\cdot]\!]$ denotes the meaning function of $\mathcal{L}$).
– Given a formula $\psi \in \mathcal{L}$, `Model`(ψ) returns (i) a satisfying model S if an SMT solver is able to determine that ψ is satisfiable in a given time limit, (ii) None if the SMT solver is able to determine that ψ is unsatisfiable in the time limit, and (iii) TimeOut otherwise.

Algorithm 1: $\widehat{\mathrm{Post}}^{\uparrow}[\tau](v)$

```
1
2  $lower' \leftarrow \bot$
3  while true do
4
5      $\langle S, S' \rangle \leftarrow$ Model($\widehat{\gamma}(v) \wedge \tau \wedge \neg\widehat{\gamma}(lower')$)
6      if $\langle S, S' \rangle$ is TimeOut then
7          return $\top$
8      else if $\langle S, S' \rangle$ is None then
9          break                    // $\widehat{\mathrm{Post}}[\tau](v) = \mathtt{lower}'$
10     else                         // $S' \not\models \widehat{\gamma}(\mathtt{lower}')$
11         $lower' \leftarrow lower' \sqcup \beta(S')$
12 $v' \leftarrow lower'$
13 return $v'$
```

Algorithm 2: $\widehat{\mathrm{Post}}^{\updownarrow}[\tau](v)$

```
1  $upper' \leftarrow \top$
2  $lower' \leftarrow \bot$
3  while $lower' \neq upper' \wedge$ ResourcesLeft do
4      $p' \leftarrow$ AbstractConsequence($lower', upper'$)
   // $p' \sqsupseteq \mathtt{lower}', p' \not\sqsupseteq \mathtt{upper}'$
5      $\langle S, S' \rangle \leftarrow$ Model($\widehat{\gamma}(v) \wedge \tau \wedge \neg\widehat{\gamma}(p')$)
6      if $\langle S, S' \rangle$ is TimeOut then
7          break
8      else if $\langle S, S' \rangle$ is None then
9          $upper' \leftarrow upper' \sqcap p'$      // $\widehat{\mathrm{Post}}[\tau](v) \sqsubseteq p'$
10     else                                  // $S' \not\models \widehat{\gamma}(p')$
11         $lower' \leftarrow lower' \sqcup \beta(S')$
12 $v' \leftarrow upper'$
13 return $v'$
```

The formula $\widehat{\gamma}(v) \wedge \tau \wedge \neg\widehat{\gamma}(lower')$ to which `Model` is applied in line 5 is a transition formula, and thus `Model` returns a two-state model $\langle S, S' \rangle$, which we refer to as a *state-pair*. (In general, we use unprimed variables to denote pre-state quantities, and primed variables to denote post-state quantities.)

– The *representation function* β (line 11 of Alg. 1) maps a singleton concrete state $S \in \mathcal{C}$ to the least value in $\mathcal{A}$ that over-approximates $\{S\}$.

The variable $lower'$ is initialized to $\bot$ (line 2). Then, on each iteration of the while-loop on lines 3–11 a concrete state-pair $\langle S, S' \rangle$ is identified that satisfies transition relation τ—as constrained by $\widehat{\gamma}(v)$ and $\neg\widehat{\gamma}(lower')$—(line 5), and the abstraction $\beta(S')$ of post-state S' is joined into $lower'$ (line 11). Each concrete state-pair $\langle S, S' \rangle$ is obtained by calling an SMT solver to obtain a satisfying assignment of the transition relation under consideration. Abstract value $lower'$ characterizes the set of already-found post-states. To ensure that a new post-state is found on each iteration, the formula $\neg\widehat{\gamma}(lower')$ is used as a blocking clause for the next call on the SMT solver; see the third conjunct in line 5.

Fig. 2 shows a possible chaotic-iteration sequence when a BII solver is invoked to find the best inductive affine-equality invariant for Eqn. (3), namely, $\langle \mathtt{V_1} \mapsto \top, \mathtt{V_6} \mapsto [\mathtt{a} = \mathtt{b}, \mathtt{x} = \mathtt{y}], \mathtt{V_{12}} \mapsto [\mathtt{a} = \mathtt{b}, \mathtt{x} = \mathtt{y}] \rangle$.[4] Note that this value corresponds exactly to the loop-invariant and exit-invariant shown in the comments on lines 6 and 12 of Fig. 1(a). The value is arrived at via some sequence of choices made during chaotic iteration to find the least fixed-point of Eqn. (3).

One such sequence is depicted in Fig. 2, where three chaotic-iteration steps are performed before the least fixed point is found. The three steps propagate information from V_1 to V_6; from V_6 to V_6; and from V_6 to V_{12}, respectively.

[4] We write abstract values in Courier typeface (e.g., $[\mathtt{a} = \mathtt{b}, \mathtt{x} = 0, \mathtt{y} = 0]$ or $[\mathtt{a}' = \mathtt{b}', \mathtt{x}' = 0, \mathtt{y}' = 0]$ are pre-state and post-state abstract values, respectively); concrete state-pairs in Roman typeface (e.g., $[a \mapsto 42, b \mapsto 27, x \mapsto 5, y \mapsto 19, a' \mapsto 17, b' \mapsto 17, x' \mapsto 0, y' \mapsto 0]$); and approximations to BII solutions as mappings from node-variables to abstract values (e.g., $\langle \mathtt{V_1} \mapsto \top, \mathtt{V_6} \mapsto [\mathtt{a} = \mathtt{b}, \mathtt{x} = 0, \mathtt{y} = 0], \mathtt{V_{12}} \mapsto [\mathtt{a} = 28, \mathtt{b} = 28, \mathtt{x} = 35, \mathtt{y} = 35] \rangle$).

$$
\begin{array}{lrl}
\text{Initialization:} & \mathtt{ans} & := \langle \mathtt{V}_1 \mapsto \top, \mathtt{V}_6 \mapsto \bot, \mathtt{V}_{12} \mapsto \bot \rangle \\
\text{Iteration 1:} & V_6 & := V_6 \sqcup \widehat{\mathrm{Post}}^{\uparrow}[\tau_{1,6}](V_1) \\
 & & = \bot \sqcup \widehat{\mathrm{Post}}^{\uparrow}[\tau_{1,6}](\top)
\end{array}
$$

$$
\boxed{\begin{array}{rl}
\mathit{lower}' & := \bot \\
\langle S, S' \rangle & := \mathtt{Model}(\mathtt{true} \wedge \tau_{1,6} \wedge \neg\widehat{\gamma}(\bot)) \\
 & = \begin{bmatrix} a \mapsto 42, b \mapsto 27, x \mapsto 5, y \mapsto 99, \\ a' \mapsto 17, b' \mapsto 17, x' \mapsto 0, y' \mapsto 0 \end{bmatrix} \quad \begin{array}{l} \text{// A satisfying concrete} \\ \text{// state-pair} \end{array} \\
\mathit{lower}' & := \bot \sqcup [\mathtt{a}' = 17, \mathtt{b}' = 17, \mathtt{x}' = 0, \mathtt{y}' = 0] \\
\langle S, S' \rangle & := \mathtt{Model}(\mathtt{true} \wedge \tau_{1,6} \wedge \neg\widehat{\gamma}([\mathtt{a}' = 17, \mathtt{b}' = 17, \mathtt{x}' = 0, \mathtt{y}' = 0])) \\
 & = \begin{bmatrix} a \mapsto 73, b \mapsto 2, x \mapsto 15, y \mapsto 19, \\ a' \mapsto 28, b' \mapsto 28, x' \mapsto 0, y' \mapsto 0 \end{bmatrix} \quad \begin{array}{l} \text{// A satisfying concrete} \\ \text{// state-pair} \end{array} \\
\mathit{lower}' & := [\mathtt{a}' = 17, \mathtt{b}' = 17, \mathtt{x}' = 0, \mathtt{y}' = 0] \sqcup [\mathtt{a}' = 28, \mathtt{b}' = 28, \mathtt{x}' = 0, \mathtt{y}' = 0] \\
v' & := [\mathtt{a}' = \mathtt{b}', \mathtt{x}' = 0, \mathtt{y}' = 0]
\end{array}}
$$

$$
\begin{array}{lrl}
 & V_6 & := \bot \sqcup [\mathtt{a} = \mathtt{b}, \mathtt{x} = 0, \mathtt{y} = 0] = [\mathtt{a} = \mathtt{b}, \mathtt{x} = 0, \mathtt{y} = 0] \\
 & \mathtt{ans} & := \langle \mathtt{V}_1 \mapsto \top, \mathtt{V}_6 \mapsto [\mathtt{a} = \mathtt{b}, \mathtt{x} = 0, \mathtt{y} = 0], \mathtt{V}_{12} \mapsto \bot \rangle \\
\text{Iteration 2:} & V_6 & := V_6 \sqcup \widehat{\mathrm{Post}}^{\uparrow}[\tau_{6,6}](V_6) \\
 & & = [\mathtt{a} = \mathtt{b}, \mathtt{x} = 0, \mathtt{y} = 0] \sqcup \widehat{\mathrm{Post}}^{\uparrow}[\tau_{6,6}]([\mathtt{a} = \mathtt{b}, \mathtt{x} = 0, \mathtt{y} = 0])
\end{array}
$$

$$
\boxed{\begin{array}{rl}
\mathit{lower}' & := \bot \\
\langle S, S' \rangle & := \mathtt{Model}(\widehat{\gamma}([\mathtt{a} = \mathtt{b}, \mathtt{x} = 0, \mathtt{y} = 0]) \wedge \tau_{6,6} \wedge \neg\widehat{\gamma}(\bot)) \\
 & = \begin{bmatrix} a \mapsto 56, b \mapsto 56, x \mapsto 0, y \mapsto 0, \\ a' \mapsto 58, b' \mapsto 58, x' \mapsto 1, y' \mapsto 1 \end{bmatrix} \quad \begin{array}{l} \text{// A satisfying concrete} \\ \text{// state-pair} \end{array} \\
\mathit{lower}' & := \bot \sqcup [\mathtt{a}' = 58, \mathtt{b}' = 58, \mathtt{x}' = 1, \mathtt{y}' = 1] \\
\langle S, S' \rangle & := \mathtt{Model}(\widehat{\gamma}([\mathtt{a} = \mathtt{b}, \mathtt{x} = 0, \mathtt{y} = 0]) \wedge \tau_{6,6} \wedge \neg\widehat{\gamma}([\mathtt{a}' = 58, \mathtt{b}' = 58, \mathtt{x}' = 1, \mathtt{y}' = 1])) \\
 & = \begin{bmatrix} a \mapsto 16, b \mapsto 16, x \mapsto 0, y \mapsto 0, \\ a' \mapsto 18, b' \mapsto 18, x' \mapsto 1, y' \mapsto 1 \end{bmatrix} \quad \begin{array}{l} \text{// A satisfying concrete} \\ \text{// state-pair} \end{array} \\
\mathit{lower}' & := [\mathtt{a}' = 58, \mathtt{b}' = 58, \mathtt{x}' = 1, \mathtt{y}' = 1] \sqcup [\mathtt{a}' = 18, \mathtt{b}' = 18, \mathtt{x}' = 1, \mathtt{y}' = 1] \\
v' & := [\mathtt{a}' = \mathtt{b}', \mathtt{x}' = 1, \mathtt{y}' = 1]
\end{array}}
$$

$$
\begin{array}{lrl}
 & V_6 & := [\mathtt{a} = \mathtt{b}, \mathtt{x} = 0, \mathtt{y} = 0] \sqcup [\mathtt{a} = \mathtt{b}, \mathtt{x} = 1, \mathtt{y} = 1] = [\mathtt{a} = \mathtt{b}, \mathtt{x} = \mathtt{y}] \\
 & \mathtt{ans} & := \langle \mathtt{V}_1 \mapsto \top, \mathtt{V}_6 \mapsto [\mathtt{a} = \mathtt{b}, \mathtt{x} = \mathtt{y}], \mathtt{V}_{12} \mapsto \bot \rangle \\
\text{Iteration 3:} & V_{12} & := V_{12} \sqcup \widehat{\mathrm{Post}}^{\uparrow}[\tau_{6,12}](V_6) \\
 & & = \bot \sqcup \widehat{\mathrm{Post}}^{\uparrow}[\tau_{6,12}]([\mathtt{a} = \mathtt{b}, \mathtt{x} = \mathtt{y}])
\end{array}
$$

$$
\boxed{\begin{array}{rl}
\mathit{lower}' & := \bot \\
\langle S, S' \rangle & := \mathtt{Model}(\widehat{\gamma}([\mathtt{a} = \mathtt{b}, \mathtt{x} = \mathtt{y}]) \wedge \tau_{6,6} \wedge \neg\widehat{\gamma}(\bot)) \\
 & = \begin{bmatrix} a \mapsto 17, b \mapsto 17, x \mapsto 99, y \mapsto 99, \\ a' \mapsto 17, b' \mapsto 17, x' \mapsto 99, y' \mapsto 99 \end{bmatrix} \quad \begin{array}{l} \text{// A satisfying concrete} \\ \text{// state-pair} \end{array} \\
\mathit{lower}' & := \bot \sqcup [\mathtt{a}' = 17, \mathtt{b}' = 17, \mathtt{x}' = 99, \mathtt{y}' = 99] \\
\langle S, S' \rangle & := \mathtt{Model}(\widehat{\gamma}([\mathtt{a} = \mathtt{b}, \mathtt{x} = \mathtt{y}]) \wedge \tau_{6,6} \wedge \neg\widehat{\gamma}([\mathtt{a}' = 17, \mathtt{b}' = 17, \mathtt{x}' = 99, \mathtt{y}' = 99])) \\
 & = \begin{bmatrix} a \mapsto 28, b \mapsto 28, x \mapsto 35, y \mapsto 35, \\ a' \mapsto 28, b' \mapsto 28, x' \mapsto 35, y' \mapsto 35 \end{bmatrix} \quad \begin{array}{l} \text{// A satisfying concrete} \\ \text{// state-pair} \end{array} \\
\mathit{lower}' & := [\mathtt{a}' = 17, \mathtt{b}' = 17, \mathtt{x}' = 99, \mathtt{y}' = 99] \sqcup [\mathtt{a}' = 28, \mathtt{b}' = 28, \mathtt{x}' = 35, \mathtt{y}' = 35] \\
v' & := [\mathtt{a}' = \mathtt{b}', \mathtt{x}' = \mathtt{y}']
\end{array}}
$$

$$
\begin{array}{lrl}
 & V_{12} & := \bot \sqcup [\mathtt{a} = \mathtt{b}, \mathtt{x} = \mathtt{y}] = [\mathtt{a} = \mathtt{b}, \mathtt{x} = \mathtt{y}] \\
 & \mathtt{ans} & := \langle \mathtt{V}_1 \mapsto \top, \mathtt{V}_6 \mapsto [\mathtt{a} = \mathtt{b}, \mathtt{x} = \mathtt{y}], \mathtt{V}_{12} \mapsto [\mathtt{a} = \mathtt{b}, \mathtt{x} = \mathtt{y}] \rangle \\
\text{Fixed Point!} & &
\end{array}
$$

Fig. 2: A possible chaotic-iteration sequence when a BII solver is invoked to find the best inductive affine-equality invariant for Eqn. (3). The parts of the trace enclosed in boxes show the actions that take place in calls to Alg. 1 ($\widehat{\mathrm{Post}}^{\uparrow}$). (By convention, primes are dropped from the abstract value returned from a call on $\widehat{\mathrm{Post}}^{\uparrow}$.)

(At this point, to discover that chaotic iteration has quiesced, the solver would have to do some additional work, which we have not shown because it does not provide any additional insight on how BII problems are solved.)

The Value of a Bilateral Algorithm. $\widehat{\text{Post}}^{\uparrow}$ is not resilient to timeouts. A query to the SMT solver—or the cumulative time for $\widehat{\text{Post}}^{\uparrow}$—might take too long, in which case the only answer that is safe for $\widehat{\text{Post}}^{\uparrow}$ to return is $\top$ (line 7 of Alg. 1). To remedy this situation, we adopt an idea from [20] and create a *bilateral* algorithm for $\widehat{\text{Post}}$. A bilateral algorithm maintains both an under-approximation and an over-approximation of the desired answer. The advantage of a bilateral algorithm is that in case of a timeout, it is safe to return the over-approximation. At various stages of the algorithm, effort is expended on improving the over-approximation, and if the algorithm were given additional time, it might return a better answer. (Such an algorithm is called an "anytime algorithm".) In our case, the over-approximation serves as an "insurance policy" against being forced to return $\top$ in case the $\widehat{\text{Post}}$ algorithm runs out of time.

Alg. 2 shows our bilateral algorithm, called $\widehat{\text{Post}}^{\updownarrow}$. The differences between $\widehat{\text{Post}}^{\uparrow}$ and $\widehat{\text{Post}}^{\updownarrow}$ are highlighted in gray. Most concern the variables *upper′* or p'. The requirements for $\widehat{\text{Post}}^{\updownarrow}$ are only slightly different from those for $\widehat{\text{Post}}^{\uparrow}$:

– Abstract domain $\mathcal{A}$ must be a lattice (i.e., with both meet and join) with a least element $\bot$ and a greatest element $\top$.
– `AbstractConsequence` must meet the following requirements:

  Let *lower′* and *upper′* be two $\mathcal{A}$ values such that $\mathit{lower}' \sqsubseteq \mathit{upper}'$.
  If $p' = \texttt{AbstractConsequence}(\mathit{lower}', \mathit{upper}')$, then $p' \sqsupseteq \mathit{lower}'$ and $p' \not\sqsupseteq \mathit{upper}'$.

  This property ensures that each iteration of the while-loop on lines 3–11 makes progress: The concrete state-pair $\langle S, S' \rangle$ is obtained by calling $\texttt{Model}(\widehat{\gamma}(v) \wedge \tau \wedge \neg\widehat{\gamma}(p'))$ (line 5). If the formula is not satisfiable, then *upper′* is decreased by meeting it with p' (line 9); otherwise, *lower′* is increased by joining it with $\beta(S')$ (line 11).
– It may appear that it is necessary for $\mathcal{A}$ to have no infinite descending chains (as well as no infinite ascending chains). However, we can modify the algorithm slightly to ensure that (i) it does not get trapped updating *upper′* along an infinite descending chain, and that (ii) it exits when *lower′* has converged to $\widehat{\text{Post}}[\tau](v)$. We can accomplish these goals by forcing the algorithm to perform the basic iteration step from $\widehat{\text{Post}}^{\uparrow}$ at least once every N iterations, for some fixed N. (See [21, App. C] for further discussion.)

$\widehat{\text{Post}}^{\updownarrow}$ initializes *upper′* to $\top$ (line 1). A value for p' is obtained by calling $\texttt{AbstractConsequence}(\mathit{lower}', \mathit{upper}')$ (line 4). The SMT solver is invoked by calling $\texttt{Model}(\widehat{\gamma}(v) \wedge \tau \wedge \neg\widehat{\gamma}(p'))$ (line 5). If the formula has a model $\langle S, S' \rangle$, Alg. 2 proceeds as in Alg. 1: $\mathit{lower}' \leftarrow \mathit{lower}' \sqcup \beta(S')$. If the formula has no model, then $\widehat{\text{Post}}[\tau](v) \sqsubseteq p'$, and thus it is safe to update *upper′* by performing

$$\begin{aligned}
V_6 &:= V_6 \sqcup \widehat{\mathrm{Post}}^{\updownarrow}[\tau_{6,6}](V_6)\\
&= [\mathtt{a} = \mathtt{b}, \mathtt{x} = \mathtt{0}, \mathtt{y} = \mathtt{0}] \sqcup \widehat{\mathrm{Post}}^{\updownarrow}[\tau_{6,6}]([\mathtt{a} = \mathtt{b}, \mathtt{x} = \mathtt{0}, \mathtt{y} = \mathtt{0}])
\end{aligned}$$

$$\boxed{\begin{aligned}
\mathit{upper}' &:= \top\\
\mathit{lower}' &:= \bot\\
p' &:= \mathtt{AbstractConsequence}(\bot, \top)\\
&= \bot\\
\langle S, S'\rangle &:= \mathtt{Model}(\widehat{\gamma}([\mathtt{a} = \mathtt{b}, \mathtt{x} = \mathtt{0}, \mathtt{y} = \mathtt{0}]) \wedge \tau_{6,6} \wedge \neg\widehat{\gamma}(\bot))\\
&= \begin{bmatrix} a \mapsto 56, b \mapsto 56, x \mapsto 0, y \mapsto 0,\\ a' \mapsto 58, b' \mapsto 58, x' \mapsto 1, y' \mapsto 1 \end{bmatrix} \quad \begin{array}{l} \text{// A satisfying concrete}\\ \text{// state-pair}\end{array}\\
\mathit{lower}' &:= \bot \sqcup [\mathtt{a}' = \mathtt{58}, \mathtt{b}' = \mathtt{58}, \mathtt{x}' = \mathtt{1}, \mathtt{y}' = \mathtt{1}]\\
p' &:= \mathtt{AbstractConsequence}([\mathtt{a}' = \mathtt{58}, \mathtt{b}' = \mathtt{58}, \mathtt{x}' = \mathtt{1}, \mathtt{y}' = \mathtt{1}], \top)\\
&= [\mathtt{x}' = \mathtt{1}]\\
\langle S, S'\rangle &:= \mathtt{Model}(\widehat{\gamma}([\mathtt{a} = \mathtt{b}, \mathtt{x} = \mathtt{0}, \mathtt{y} = \mathtt{0}]) \wedge \tau_{6,6} \wedge \neg\widehat{\gamma}([\mathtt{x}' = \mathtt{1}]))\\
&= \mathsf{None}\\
\mathit{upper}' &:= \top \sqcap [\mathtt{x}' = \mathtt{1}] = [\mathtt{x}' = \mathtt{1}]\\
p' &:= \mathtt{AbstractConsequence}([\mathtt{a}' = \mathtt{58}, \mathtt{b}' = \mathtt{58}, \mathtt{x}' = \mathtt{1}, \mathtt{y}' = \mathtt{1}], [\mathtt{x}' = \mathtt{1}])\\
&= [\mathtt{y}' = \mathtt{1}]\\
\langle S, S'\rangle &:= \mathtt{Model}(\widehat{\gamma}([\mathtt{a} = \mathtt{b}, \mathtt{x} = \mathtt{0}, \mathtt{y} = \mathtt{0}]) \wedge \tau_{6,6} \wedge \neg\widehat{\gamma}([\mathtt{y}' = \mathtt{1}]))\\
&= \mathsf{None}\\
\mathit{upper}' &:= [\mathtt{x}' = \mathtt{1}] \sqcap [\mathtt{y}' = \mathtt{1}] = [\mathtt{x}' = \mathtt{1}, \mathtt{y}' = \mathtt{1}]\\
p' &:= \mathtt{AbstractConsequence}([\mathtt{a}' = \mathtt{58}, \mathtt{b}' = \mathtt{58}, \mathtt{x}' = \mathtt{1}, \mathtt{y}' = \mathtt{1}], [\mathtt{x}' = \mathtt{1}, \mathtt{y}' = \mathtt{1}])\\
&= [\mathtt{a}' = \mathtt{58}]\\
\langle S, S'\rangle &:= \mathtt{Model}(\widehat{\gamma}([\mathtt{a} = \mathtt{b}, \mathtt{x} = \mathtt{0}, \mathtt{y} = \mathtt{0}]) \wedge \tau_{6,6} \wedge \neg\widehat{\gamma}([\mathtt{a}' = \mathtt{58}]))\\
&= \begin{bmatrix} a \mapsto 19, b \mapsto 19, x \mapsto 0, y \mapsto 0,\\ a' \mapsto 21, b' \mapsto 21, x' \mapsto 1, y' \mapsto 1 \end{bmatrix} \quad \begin{array}{l} \text{// A satisfying concrete}\\ \text{// state-pair}\end{array}\\
\mathit{lower}' &:= [\mathtt{a}' = \mathtt{58}, \mathtt{b}' = \mathtt{58}, \mathtt{x}' = \mathtt{1}, \mathtt{y}' = \mathtt{1}] \sqcup [\mathtt{a}' = \mathtt{21}, \mathtt{b}' = \mathtt{21}, \mathtt{x}' = \mathtt{1}, \mathtt{y}' = \mathtt{1}]\\
&= [\mathtt{a}' = \mathtt{b}', \mathtt{x}' = \mathtt{1}, \mathtt{y}' = \mathtt{1}]\\
p' &:= \mathtt{AbstractConsequence}([\mathtt{a}' = \mathtt{b}', \mathtt{x}' = \mathtt{1}, \mathtt{y}' = \mathtt{1}], [\mathtt{x}' = \mathtt{1}, \mathtt{y}' = \mathtt{1}])\\
&= [\mathtt{a}' = \mathtt{b}']\\
\langle S, S'\rangle &:= \mathtt{Model}(\widehat{\gamma}([\mathtt{a} = \mathtt{b}, \mathtt{x} = \mathtt{0}, \mathtt{y} = \mathtt{0}]) \wedge \tau_{6,6} \wedge \neg\widehat{\gamma}([\mathtt{a}' = \mathtt{b}']))\\
&= \mathsf{None}\\
\mathit{upper}' &:= [\mathtt{x}' = \mathtt{1}, \mathtt{y}' = \mathtt{1}] \sqcap [\mathtt{a}' = \mathtt{b}']\\
&= [\mathtt{a}' = \mathtt{b}', \mathtt{x}' = \mathtt{1}, \mathtt{y}' = \mathtt{1}]\\
\mathit{lower}' \neq \mathit{upper}' &= \mathsf{false}\\
v' &:= [\mathtt{a}' = \mathtt{b}', \mathtt{x}' = \mathtt{1}, \mathtt{y}' = \mathtt{1}]
\end{aligned}}$$

$$\begin{aligned}
V_6 &:= [\mathtt{a} = \mathtt{b}, \mathtt{x} = \mathtt{0}, \mathtt{y} = \mathtt{0}] \sqcup [\mathtt{a} = \mathtt{b}, \mathtt{x} = \mathtt{1}, \mathtt{y} = \mathtt{1}] = [\mathtt{a} = \mathtt{b}, \mathtt{x} = \mathtt{y}]\\
\mathtt{ans} &:= \langle \mathtt{V}_1 \mapsto \top, \mathtt{V}_6 \mapsto [\mathtt{a} = \mathtt{b}, \mathtt{x} = \mathtt{y}], \mathtt{V}_{12} \mapsto \bot\rangle
\end{aligned}$$

Fig. 3: A possible trace of Iteration 2 from Fig. 2 when the call to $\widehat{\mathrm{Post}}^{\uparrow}$ (Alg. 1) is replaced by a call to $\widehat{\mathrm{Post}}^{\updownarrow}$ (Alg. 2).

a meet with p' (line 9). Moreover, this step is guaranteed to decrease the value of upper' because $p' \not\sqsupseteq \mathit{upper}'$.

When there is an SMT-solver timeout (line 6), $\widehat{\mathrm{Post}}^{\updownarrow}$ returns upper' as the answer (lines 7, 12, and 13). In this case, the value returned can be an over-approximation of the desired answer $\widehat{\mathrm{Post}}[\tau](v)$; however, if the loop exits without a timeout, then lower' must equal upper', and the return value equals $\widehat{\mathrm{Post}}[\tau](v)$.

Fig. 3 shows a possible trace of Iteration 2 from Fig. 2 when the call to $\widehat{\mathrm{Post}}^{\uparrow}$ (Alg. 1) is replaced by a call to $\widehat{\mathrm{Post}}^{\updownarrow}$ (Alg. 2). Note how a collection of individual, non-trivial, upper-bounding constraints are acquired on the second, third, and fifth calls to AbstractConsequence: $[\mathtt{x}' = \mathtt{1}]$, $[\mathtt{y}' = \mathtt{1}]$, and $[\mathtt{a}' = \mathtt{b}']$, respectively. By this means, *upper*$'$ works its way down the chain $\top \sqsupset [\mathtt{x}' = \mathtt{1}] \sqsupset [\mathtt{x}' = \mathtt{1}, \mathtt{y}' = \mathtt{1}] \sqsupset [\mathtt{a}' = \mathtt{b}', \mathtt{x}' = \mathtt{1}, \mathtt{y}' = \mathtt{1}]$. After each call to AbstractConsequence, the abstract-consequence constraint is tested to see if it really is an upper bound on the answer. For instance, the fourth call to AbstractConsequence returns $[\mathtt{a}' = \mathtt{58}]$. The assertion that $[\mathtt{a}' = \mathtt{58}]$ is an upper-bounding constraint is refuted by the concrete state-pair

$$\langle S, S' \rangle = \begin{bmatrix} a \mapsto 19, b \mapsto 19, x \mapsto 0, y \mapsto 0, \\ a' \mapsto 21, b' \mapsto 21, x' \mapsto 1, y' \mapsto 1 \end{bmatrix},$$

which is used to improve the value of *lower*$'$.

The important point is that if Iteration 2 is taking too much time, $\widehat{\mathrm{Post}}^{\updownarrow}$ can be stopped and *upper*$'$ returned as the answer. In contrast, if $\widehat{\mathrm{Post}}^{\uparrow}$ is stopped because it is taking too much time, the only safe answer that can be returned is $\top$. The "can-be-stopped-anytime" property of $\widehat{\mathrm{Post}}^{\updownarrow}$ can make a significant difference in the final answer. For instance, suppose that $\widehat{\mathrm{Post}}^{\uparrow}$ and $\widehat{\mathrm{Post}}^{\updownarrow}$ both stop early during Iteration 2 (Figs. 2 and 3, respectively), and that $\widehat{\mathrm{Post}}^{\updownarrow}$ returns $[\mathtt{x} = \mathtt{1}, \mathtt{y} = \mathtt{1}]$, whereas $\widehat{\mathrm{Post}}^{\uparrow}$ returns $\top$. Assuming no further timeouts take place during the evaluation of Eqn. (3), the respective final answers would be

$$\begin{aligned} &\widehat{\mathrm{Post}}^{\uparrow} : \langle \mathtt{V}_1 \mapsto \top, \mathtt{V}_6 \mapsto \top, \mathtt{V}_{12} \mapsto \top \rangle \\ &\widehat{\mathrm{Post}}^{\updownarrow} : \langle \mathtt{V}_1 \mapsto \top, \mathtt{V}_6 \mapsto [\mathtt{x} = \mathtt{y}], \mathtt{V}_{12} \mapsto [\mathtt{x} = \mathtt{y}] \rangle \end{aligned}$$

Because of the timeout, the answer computed by $\widehat{\mathrm{Post}}^{\updownarrow}$ is not the *best* inductive affine-equality invariant; however, the answer establishes that the equality constraint $[\mathtt{x} = \mathtt{y}]$ holds at both lines 6 and 12 of Fig. 1(a).

Attaining the Best Inductive $\mathcal{A}$-Invariant.

Lemma 1. *The least fixed-point of Eqn. (2) (the best $\mathcal{A}$-transformer equations of a transition system) is the best inductive invariant expressible in $\mathcal{A}$.*

Corollary 1. *Applying an equation solver to the best $\mathcal{A}$-transformer equations, using either $\widehat{\mathrm{Post}}^{\uparrow}$ or $\widehat{\mathrm{Post}}^{\updownarrow}$ to evaluate equation right-hand sides, finds the best inductive $\mathcal{A}$-invariant if there are no timeouts during the evaluation of any right-hand side.*

3 Best Inductive Invariants and Interprocedural Analysis

In this section, we present an analysis framework that extends the results of §2 to multi-procedure programs. The interprocedural-analysis algorithm presented here finds *best* inductive invariants, under the same set of assumptions used in §2. In contrast, past algorithms for interprocedural analysis are only guaranteed to find an *over-approximating* solution—i.e., an inductive invariant, but in general not the best inductive invariant.

Eqn. (2) from Obs. 1 is related to standard equation-solving methods for *intra*procedural dataflow analysis. That is, the equations are similar, except that each right-hand side occurrence of a state transformer $\tau_{i,j}$ is given the *best-transformer interpretation* by means of $\widehat{\mathrm{Post}}[\tau_{i,j}]$. Similarly, our method for solving the interprocedural BII problem resembles standard equation-solving methods for *inter*procedural dataflow analysis. In particular, our framework is similar to the so-called "functional approach" to interprocedural analysis of Sharir and Pnueli [19], which works with an abstract domain that abstracts transition relations. For instance, our approach

– also uses an abstract domain that abstracts transition relations, and
– creates a *summary transformer* for each reachable procedure P, which over-approximates the transition relation of P.

However, to make the approach suitable for the interprocedural context, the algorithm uses a generalization of $\widehat{\mathrm{Post}}$, called $\widehat{\mathrm{Compose}}[\tau](a)$, where $\tau \in \mathcal{L}[\overrightarrow{v}; \overrightarrow{v}']$ and $a \in \mathcal{A}[\overrightarrow{v}; \overrightarrow{v}']$ both represent transition relations over the program variables $\overrightarrow{v}$. The goal of $\widehat{\mathrm{Compose}}[\tau](a)$ is to create an $\mathcal{A}[\overrightarrow{v}; \overrightarrow{v}']$ value that is the best over-approximation of a's action followed by τ's action. $\widehat{\mathrm{Compose}}^{\updownarrow}$ is essentially identical to Alg. 2, except that in line 5, $\widehat{\mathrm{Compose}}^{\updownarrow}$ performs a query using a three-state formula,

$$\langle S, S', S'' \rangle \leftarrow \mathtt{Model}(\widehat{\gamma}_{[\overrightarrow{v},\overrightarrow{v}']}(a) \wedge \tau_{[\overrightarrow{v}',\overrightarrow{v}'']} \wedge \neg\widehat{\gamma}_{[\overrightarrow{v},\overrightarrow{v}'']}(p')),$$

and in line 11, $\widehat{\mathrm{Compose}}^{\updownarrow}$ applies a two-state version of β to S and S'', dropping S' completely: $\mathit{lower}' \leftarrow \mathit{lower}' \sqcup \beta(S, S'')$. ($\widehat{\mathrm{Compose}}^{\uparrow}$ is defined similarly.)

To simplify the presentation, we assume that the language does not support either local variables or parameter passing. Such features can be handled in our framework in a completely standard way, using *merge functions* [13, 14].

A *program* is defined by a set of procedures P_i, $0 \leq i \leq K$, and represented by a directed graph $G = (N, F)$ called an *Interprocedural Cut-Point Graph* (ICPG), which represents a partitioning of the program into loop-free fragments. G consists of a collection of intraprocedural Cut-Point Graphs (CPGs) $G_1, G_2, \ldots, G_K$, one of which, G_{main}, represents the program's main procedure. (Calls to *main* are forbidden.) The node set N_i of CPG $G_i = (N_i, F_i)$ is partitioned into five disjoint subsets: $N_i = E_i \uplus X_i \uplus C_i \uplus R_i \uplus L_i$. G_i contains exactly one *enter* node (i.e., $E_i = \{e_i\}$) and exactly one *exit* node (i.e., $X_i = \{x_i\}$). A procedure call in G_i is represented by two nodes, a *call* node $c \in C_i$ and a *return-site* node $r \in R_i$. For $n \in N = \bigcup_i \{N_i\}$, $\mathit{proc}(n)$ denotes the (index of the) procedure that contains n.

Each procedure P_i is partitioned into a collection of loop-free fragments, which may be accomplished by identifying the set of loop-heads L_i [3]. The edge-set F_i of G_i represents procedure P_i's loop-free fragments. A loop-free fragment $f[m_1, m_2]$ in procedure P_i is a single-entry/single-exit region from $m_1 \in N_i$ to $m_2 \in N_i$: $f[m_1, m_2]$ consists of all paths of length ≥ 1 in P_i's control-flow graph that begin at m_1, end at m_2, and contain no member of N other than the beginning and ending nodes. For each loop-free-fragment $f \in F_i$, there is a formula $\psi_f \in \mathcal{L}$ that captures the semantics of f.

In addition to the loop-free fragments that connect the nodes of the individual cut-point graphs in G, each procedure call, represented by call-node c and return-site node r, has two edges: (i) a *call-to-enter* edge from c to the enter node of the called procedure, and (ii) an *exit-to-return-site* edge from the exit node of the called procedure to r. The functions *call* and *ret* record matching call and return-site nodes: $\mathit{call}(r) = c$ and $\mathit{ret}(c) = r$. We assume that an enter node has no incoming edges except call-to-enter edges.

Solving the Interprocedural BII Problem. Our method for solving the BII problem for multi-procedure programs is similar to the method described in §2, except that instead of Eqn. (2), we find the least solution to Eqns. (4)–(8) below. As in §2, each application of the right-hand side of an equation is given the *best-transformer interpretation*—in this case, by means of $\widehat{\mathrm{Compose}}$ rather than $\widehat{\mathrm{Post}}$. ($\mathit{Id}|_a$ denotes the identity transformer restricted to inputs in $a \in \mathcal{A}$.)

$$\phi(e_{main}) = \mathit{Id}|_{a_1} \qquad a_1 \in \mathcal{A} \text{ describes the set of initial stores at } e_{main} \tag{4}$$

$$\phi(e_p) = \mathit{Id}|_a \qquad e_p \in E,\ p \neq \mathit{main}, \text{ and } a = \bigsqcup_{c \in C,\ c \text{ calls } p} V_c \tag{5}$$

$$\phi(n) = \bigsqcup_{f[m,n] \in F} \widehat{\mathrm{Compose}}[\psi_{f[m,n]}, \phi(m)] \qquad \text{for } n \in N,\ n \notin (R \cup E) \tag{6}$$

$$\phi(n) = \widehat{\mathrm{Compose}}[\widehat{\gamma}(\phi(x_q)), \phi(\mathit{call}(n))] \qquad \text{for } n \in R, \text{ and } \mathit{call}(n) \text{ calls } q \tag{7}$$

$$V_n = \mathrm{range}(\phi(n)) \tag{8}$$

The equations involve two sets of "variables": $\phi(n)$ and V_n, where $n \in N$. $\phi(n)$ is a partial function that represents a summary of the transformation from $e_{proc(n)}$ to n. The domain of $\phi(n)$ over-approximates the set of reachable states at $e_{proc(n)}$ from which it is possible to reach n; the range of $\phi(n)$ over-approximates the set of reachable states at n. V_n's value equals the range of $\phi(n)$.

An important difference between our algorithm and that of Sharir and Pnueli is that in our algorithm, the initial abstract value for the enter node e_p for procedure p is specialized to the reachable inputs of p (see Eqn. (5)). In the Sharir and Pnueli algorithm, $\phi(e_p)$ is always set to *Id*. Fig. 4 illustrates the effect of specializing the abstract pre-condition at enter node e_p, and the consequent effect on the inferred abstract post-condition. The abstract domain used in Fig. 4 is the domain of affine equalities over machine integers [6]. With that domain, it is possible to express that a 32-bit variable x holds an even number: $2^{31}x = 0$. Consequently, the initial abstract value for enter node e_{halve} is the identity relation, constrained so that x is even. Similarly, the initial abstract value for enter node e_{increase} is the identity relation, constrained so that x is odd.

Note that the abstract value at the exit point x_p of a procedure p serves as a procedure summary—i.e., an abstraction of p's transition relation. Fig. 4 shows that by giving halve and increase more precise abstract values at the respective enter nodes, we obtained more precise procedure summaries at the respective exit points. In particular, for halve, the constraint $x - 2x' = 0$ provides a good

```
main() {
    x = read_input();
    while(x != 1) {
        if(even(x)) halve();
        else increase();
    }
}

void halve() {
    x = x ≫ 1;
}

void increase() {
    x = 3*x + 1;
}
```

		Abstract value at enter node e_p	
		Id	$Id\|_a$, where $a = \bigsqcup_{c \in C,\, c \text{ calls } p} V_c$
halve	pre-condition (e_{halve})	$x' = x$	$2^{31}x = 0 \wedge x' = x$
	post-condition (x_{halve})	$\top$	$2^{31}x = 0 \wedge x - 2x' = 0$
increase	pre-condition (e_{increase})	$x' = x$	$2^{31}x = 1 \wedge x' = x$
	post-condition (x_{increase})	$x' = 3x + 1$	$2^{31}x = 1 \wedge 2^{31}x' = 0 \wedge x' = 3x + 1$

Fig. 4: The effect of specializing the abstract pre-condition at enter node e_p, and the resulting strengthening of the inferred abstract post-condition.

characterization of the effect of a right-shift operation, but only when x is known to be even (cf. the entries for halve's post-condition in columns 3 and 4 of Fig. 4).

Attaining the Best Inductive $\mathcal{A}$-Invariant. Essentially the same lemma and corollary stated in §2 carry over to the interprocedural case, except that they apply to an abstract domain that abstracts transition relations.

Lemma 2. *Let $\mathcal{A}[\overrightarrow{v}; \overrightarrow{v}']$ be an abstract domain that abstracts transition relations. The least fixed-point of Eqns. (4)–(8) for the "variables" $\{\phi(n) \mid n \in N\}$ is the best inductive $\mathcal{A}[\overrightarrow{v}; \overrightarrow{v}']$-invariant.*

Corollary 2. *Applying an equation solver to Eqns. (4)–(8), using either $\widehat{\text{Compose}}^{\uparrow}$ or $\widehat{\text{Compose}}^{\updownarrow}$ for $\widehat{\text{Compose}}$ when evaluating equation right-hand sides, finds the best inductive $\mathcal{A}[\overrightarrow{v}; \overrightarrow{v}']$-invariant if there are no timeouts during the evaluation of any right-hand side.*

What Does It Mean to Be Inductive in the Interprocedural Case? The *transition* invariants $\{\phi(n) \mid n \in N\}$ recovered by our approach to interprocedural analysis are inductive in a somewhat special sense. One has to consider the (hyper-)graph of dependences induced by Eqns. (4)–(8); each right-hand side of an equation becomes a higher-order transformer that maps transition relations to transition relations. In contrast, the *state* invariants $\{V_n \mid n \in N\}$ recovered by our method are generally not inductive with respect to the successor relation in the ICPG.

4 Evaluation

Santini. Our first implementation, called Santini, is a tool for inferring the strongest inductive post-conditions (or procedure summaries) using the abstract domain of predicate abstraction. Santini is implemented on the Boogie platform

[2], a program-verification framework that generates and discharges classical verification conditions using SMT solvers. Boogie already comes with an implementation of the Houdini algorithm [8] for inferring *conjunctive* invariants from a given set of predicates. Because Santini does full predicate abstraction, it can learn arbitrary Boolean combinations of predicates, and is thus strictly more powerful than Houdini. However, conjunctive invariants are often good enough, in which case Houdini provides a good scalable solution. In Santini, we retain the benefits of conjunctive invariants, as described next.

Santini uses Alg. 2 instantiated with the predicate-abstraction domain. Recall that Alg. 2 maintains both an over-approximation ($upper'$) and an under-approximation ($lower'$). We implement the `AbstractConsequence`($lower', upper'$) operation as follows: it searches the set of predicates for one (say p) that satisfies $p \sqsupseteq lower'$ and $p \not\sqsupseteq upper'$. If it does not find any such predicate, then it simply returns $lower'$ (which is a valid answer). With such an operation, it is easy to show that $upper'$ can only be a conjunction of predicates, until it converges to $lower'$ (and the algorithm terminates). Thus, $upper'$ is always a conjunctive over-approximation of the answer. Furthermore, we impose a timeout on the total time spent in the theorem prover whenever `AbstractConsequence` returned $lower'$ (and there is no timeout otherwise). Let Santini[$T = t$] refer to Santini with a timeout value of t seconds. Then Santini[$T = 0$] is merely Houdini (i.e., a tool for inferring conjunctive invariants), whereas Santini[$T = \infty$] is full predicate abstraction.

Santini Experiments. Corral is a model checker for C programs. For each program, we took the invariants obtained by Houdini and Santini, and supplied them to Corral. The experiments were designed to compare (i) the improvement in precision from Santini, as measured by the ability of the Corral model checker to obtain a better outcome: "indefinite" (= "timeout" or "inconclusive") converted to "definite" (= "proof" or "counterexample"), and (ii) the relative speed—end-to-end—of Corral[Houdini] vs. Corral[Santini].

The test suite consisted of checking multiple properties on multiple Windows Device Drivers selected randomly from SDV, resulting in a total of 378 driver-property pairs. The drivers ranged in size from 3K lines of code to 8K lines of code (excluding comments, whitespace, and type declarations). Our experiments confirmed that Santini[$T = 0$] does indeed behave like Houdini: the running time of these tools was always within 5% of each other, and the behavior of Corral was identical (in the number of procedures inlined), indicating that both Santini[$T = 0$] and Houdini produced semantically equivalent invariants. Fig. 5 shows a comparison of the running time of Corral[Houdini] vs. Corral[Santini[$T = 50$]]; a 50-second timeout value was large enough to avoid any SMT timeouts within Santini. The Corral timeout limit was 600 seconds.

Fig. 5 shows that Santini's running time was a burden for the easy cases, where its precision was not required, but provides benefits for the hard cases. In particular, (i) for benchmarks for which Corral[Houdini] gave a definite result in <60 seconds, Corral[Santini] was 22% slower; (ii) for benchmarks for which Corral[Houdini] gave a definite result in >60 seconds, Corral[Santini] was 17%

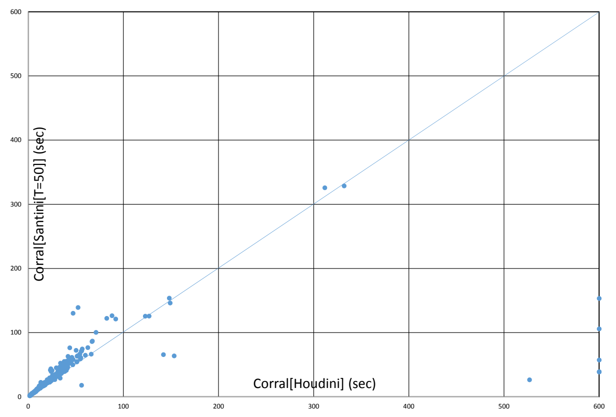

Fig. 5: A scatter plot of Corral[Houdini] against Corral[Santini[$T = 50$]].

Rule	Region modeled
$\langle p, u\rangle \hookrightarrow \langle p, v\rangle$	Intraprocedural loop-free fragment from u to v
$\langle p, c\rangle \hookrightarrow \langle p, e_f\ r\rangle$	Call to f from c that returns to r
$\langle p, x_f\rangle \hookrightarrow \langle p, \varepsilon\rangle$	Return from f at exit node x_f

(a) The encoding of an ICPG using PDS rules.

	Program Info.		Precision		Time (sec.)	
Program	#Procs	#BBs	equal	better	KS [6]	BII
finger	18	298	292	6	176.23	584.22
label	16	573	573	0	565.94	2760.07
subst	16	609	604	5	461.51	2432.51
chkdsk	18	787	787	0	477.42	3213.38
route	40	931	931	0	1080.48	3053.99
convert	38	1013	946	67	798.53	4459.82
logoff	46	1145	1140	5	1025.46	3710.44
comp	35	1261	1260	1	1440.99	4786.61
setup	67	1862	1850	12	255.38	748.82

(b) The WPDS experiment.

Fig. 6: In (b), KS refers to the method described by Elder et al. [6]. Columns 4 and 5 show the number of basic blocks where KS and BII found the same affine relations, and the number of blocks where BII identified more precise affine relations.

faster; (iii) for benchmarks for which Corral[Houdini] gave an indefinite result, Corral[Santini] gave a definite result in 9 out of 19 cases (47%).

A BII Framework Based on Weighted Pushdown Systems Our second implementation uses WALi [11], a library for Weighted Pushdown Systems (WPDSs)—a tool for solving program-analysis problems using an abstract domain. A program is modeled as a transition system with a stack, specified by a set of rules. Fig. 6(a) shows how to encode an ICPG via PDS rules. In addition, each rule has an associated *weight*—i.e., a value in a domain of abstract transition-relations. To implement the method from §3, we created a mixed-weight domain: (i) the weight on a PDS rule for an intraprocedural loop-free fragment f is the formula ψ_f; (ii) the weights for the encodings of other quantities in Eqns. (4)–(8) are the more usual kind of abstract-transition-relation weights. For the mixed-weight domain, the WPDS "extend" operation invokes $\widehat{\text{Compose}}^{\updownarrow}$ when given a formula-weight and an abstract-transition-relation weight.

WPDS Experiments. In our experiment, weights on transitions of the answer automaton were affine-equality transformers for modular arithmetic (the KS domain from [6, §5]) over processor registers and inferred memory variables [1, §4]. Fig. 6(b) shows the preliminary experimental results. To limit the number of variables tracked by the KS domain, we performed the analysis on a per-procedure basis. For each call to a sub-procedure, instead of adding a PDS rule that represents the actual call, we added a rule to the return-site with a (summarized) weight that represents a havoc on all variables that might be modified by the called procedure. We used 9 Windows applications to compare a more conventional approach to affine-relation analysis [6, §6] with the BII approach. Columns 3 and 5 show the number of basic blocks, and the number of basic blocks where the BII technique found more precise affine relations. It resulted in 0–6.6% improvement in precision, with about a 4-fold slowdown in running time (computed as a geometric mean).

5 Related Work

Previous work related to BII falls into two categories:

1. Work on $\widehat{\text{Post}}$ (i.e., BAT) [10, 16, 17, 24, 18, 12, 4, 6, 15, 22, 20], which, as shown by Obs. 1 and Eqn. (2), provides a solution to the *intra*procedural BII problem.
2. Work specifically on the BII problem itself [8, 23, 9].

Much of the work in item 1 could be used for the *inter*procedural BII problem by (i) generalizing those algorithms to perform $\widehat{\text{Compose}}$, and (ii) using them to solve Eqns. (4)–(8).

Houdini [8] is the first algorithm that we are aware of that solves a version of the BII problem. The paper on Houdini does not describe the work in terms of abstract interpretation. Santini (§4) was directly inspired by Houdini, as an effort to broaden Houdini's range of applicability.

Yorsh et al. [23] introduced a particularly interesting technique. Their algorithm for the BII problem can be viewed as basically solving Eqn. (2) using $\widehat{\text{Post}}^{\uparrow}$. However, they observed that it is not necessary to rely on calls to an SMT solver for *all* of the concrete states used by $\widehat{\text{Post}}^{\uparrow}$; instead, they used concrete execution of the program as a way to generate concrete states very cheaply. If for some program point q of interest they have state-set S_q, they obtain an under-approximation for the abstract value V_q by performing $V_q = \sqcup\{\beta(\sigma) \mid \sigma \in S_q\}$. This idea is similar in spirit to the computation of candidate invariants from execution information by Daikon [7]. Because $\widehat{\text{Post}}^{\updownarrow}$ works simultaneously from below and from above, the Yorsh et al. heuristic can be used to improve the speed of convergence of *lower′* in line 11 of Alg. 2.

If we think of $\tau = \langle \ldots, \tau_{i,j}, \ldots \rangle$ as a monolithic transformer, an alternative way of stating the objective of intraprocedural BII—in a form similar to the statement of problem (1) from §1—is as follows:

– Given a concrete transformer τ and an abstract value $a \in \mathcal{A}$ that over-approximates the set of initial states, apply the best abstract transformer for τ^* to a (i.e., *apply* $\widehat{\text{Post}}[\tau^*](a)$).

This problem was the subject of a recent technical report by Garoche et al. [9].

By stating the problem in nearly the same terms as problem (1) from §1, it is natural to make a similar generalization of BAT problem (2) from §1, namely,

– Given a concrete transformer τ, obtain an explicit representation of the best abstract transformer for τ^* (i.e., *obtain* $\widehat{\text{Post}}[\tau^*]$).

The latter formulation provides an alternative characterization of what we achieved in §3 by solving Eqns. (4)–(8). In particular, for a monolithic transformer τ, $\widehat{\text{Post}}[\tau^*]$ is made up of exactly the values $\{\phi(n) \mid n \in N\}$ from least fixed-point of Eqns. (4)–(8).

References

1. G. Balakrishnan and T. Reps. WYSINWYX: What You See Is Not What You eXecute. *TOPLAS*, 32(6), 2010.
2. M. Barnett, B.-Y. Chang, R. DeLine, B. Jacobs, and K. Leino. Boogie: A modular reusable verifier for object-oriented programs. In *FMCO*, 2005.
3. F. Bourdoncle. Efficient chaotic iteration strategies with widenings. In *FMPA*, 1993.
4. J. Brauer and A. King. Automatic abstraction for intervals using Boolean formulae. In *SAS*, 2010.
5. P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *POPL*, 1979.
6. M. Elder, J. Lim, T. Sharma, T. Andersen, and T. Reps. Abstract domains of affine relations. In *SAS*, 2011.
7. M. Ernst, J. Perkins, P. Guo, S. McCamant, C. Pacheco, M. Tschantz, and C. Xiao. The Daikon system for dynamic detection of likely invariants. *SCP*, 69(1–3), 2007.
8. C. Flanagan and K. R. M. Leino. Houdini, an annotation assistant for Esc/Java. In *FME*, 2001.
9. P.-L. Garoche, T. Kahsai, and C. Tinelli. Invariant stream generators using automatic abstract transformers based on a decidable logic. Tech. Rep. CoRR abs/1205.3758, arXiv, 2012.
10. S. Graf and H. Saïdi. Construction of abstract state graphs with PVS. In *CAV*, 1997.
11. N. Kidd, A. Lal, and T. Reps. WALi: The Weighted Automaton Library, 2007. www.cs.wisc.edu/wpis/wpds/download.php.
12. A. King and H. Søndergaard. Automatic abstraction for congruences. In *VMCAI*, 2010.
13. J. Knoop and B. Steffen. The interprocedural coincidence theorem. In *CC*, 1992.
14. A. Lal, T. Reps, and G. Balakrishnan. Extended weighted pushdown systems. In *CAV*, 2005.
15. D. Monniaux. Automatic modular abstractions for template numerical constraints. *LMCS*, 6(3), 2010.
16. J. Regehr and A. Reid. HOIST: A system for automatically deriving static analyzers for embedded systems. In *ASPLOS*, 2004.
17. T. Reps, M. Sagiv, and G. Yorsh. Symbolic implementation of the best transformer. In *VMCAI*, 2004.
18. S. Sankaranarayanan, H. Sipma, and Z. Manna. Scalable analysis of linear systems using mathematical programming. In *VMCAI*, 2005.

19. M. Sharir and A. Pnueli. Two approaches to interprocedural data flow analysis. In *Program Flow Analysis: Theory and Applications*. Prentice-Hall, 1981.
20. A. Thakur, M. Elder, and T. Reps. Bilateral algorithms for symbolic abstraction. In *SAS*, 2012.
21. A. Thakur, M. Elder, and T. Reps. Bilateral algorithms for symbolic abstraction. TR 1713, CS Dept., Univ. of Wisconsin, Madison, WI, 2012.
22. A. Thakur and T. Reps. A method for symbolic computation of precise abstract operations. In *CAV*, 2012.
23. G. Yorsh, T. Ball, and M. Sagiv. Testing, abstraction, theorem proving: Better together! In *ISSTA*, 2006.
24. G. Yorsh, T. Reps, and M. Sagiv. Symbolically computing most-precise abstract operations for shape analysis. In *TACAS*, 2004.

A Proofs

Lemma 1 *The least fixed-point of Eqn. (2) (the best $\mathcal{A}$-transformer equations of a transition system) is the best inductive invariant expressible in $\mathcal{A}$.* $\square$

Proof. (i) The least fixed-point of the best-transformer equations is (a) an inductive invariant, and (b) the least post-fixed-point. (ii) All inductive invariants expressible in $\mathcal{A}$ are post-fixed-points of the best-transformer equations. Consequently, the least fixed-point is the least (i.e., best) inductive $\mathcal{A}$-invariant. $\square$

Lemma 2 *Let $\mathcal{A}$ be an abstract domain that abstracts transition relations. The least fixed-point of Eqns. (4)–(8) for the "variables" $\{\phi(n) \mid n \in N\}$ is the best inductive $\mathcal{A}$-invariant.* $\square$

Proof. Essentially identical to the proof of Lem. 1. $\square$