Minimally Comparing Relational Abstract Domains

Kenny Ballou[0000−0002−6032−474X] and Elena Sherman[0000−0003−4522−9725]

Boise State University
kennyballou@u.boisestate.edu,elenasherman@boisestate.edu

Abstract. Value-based static analysis techniques express computed program invariants as logical formula over program variables. Researchers and practitioners use these invariants to aid in software engineering and verification tasks. When selecting abstract domains, practitioners weigh the cost of a domain against its expressiveness. However, an abstract domain's expressiveness tends to be stated in absolute terms; either mathematically via the sub-polyhedra the domain is capable of describing, empirically using a set of known properties to verify, or empirically via logical entailment using the entire invariant of the domain at each program point. Due to *carry-over* effects, however, the last technique can be problematic because it tends to provide simplistic and imprecise comparisons.

We address these limitations of comparing, in general, abstract domains via logical entailment in this work. We provide a fixed-point algorithm for including the minimally necessary variables from each domain into the compared formula. Furthermore, we empirically evaluate our algorithm, comparing different techniques of widening over the Zones domain and comparing Zones to an incomparable Relational Predicates domain. Our empirical evaluation of our technique shows an improved granularity of comparison. It lowered the number of more precise invariants when comparing analysis techniques, thus, limiting the prevalent *carry-over* effects. Moreover, it removed undecidable invariants and lowered the number of incomparable invariants when comparing two incomparable relational abstract domains.

Keywords: Static Analysis · Abstract Domain Comparison · Data-Flow Analysis · Abstract Interpretation

1 Introduction

Various value-based static analysis techniques express computed program invariants as a logical formula over program variables. For example, abstract interpretation [7] uses abstract domains such as Zones [16] and Octagons [18] to describe an invariant as a set of linear integer inequalities in a restricted format. Other techniques such as symbolic execution [12] and predicate analysis combined with a symbolic component [21] do the same, only using a general linear integer arithmetic format. These invariants are then used for program verification [4,25], program optimization [1,11], and for software development tasks.

Static analysis developers rarely use a computed invariant by itself, but rather compare them to determine effects of new algorithms or abstract domain choices on the invariant precision. For example, to evaluate tuning analyzer parameters, static analysis researchers compare invariant values $\mathcal{I}$ and $\widetilde{\mathcal{I}}$ from the original and tuned analyzer runs, respectively. If an invariant becomes more precise, we conclude that the new technique or a different domain choice results in a more precise analysis. For relational domains, one can use queries to an SMT solver, such as Z3 [19], to determine which invariant is more precise by checking their implication relations.

However, to objectively measure such effects in a computed invariant after statement s, $\mathcal{I}_s$, we need to compare only the part of $\mathcal{I}_s$ affected by the transfer function of s, τ_s. This way, if $\widetilde{\mathcal{I}}$ has already been more precise than $\mathcal{I}$ before s and τ_s has not changed the relevant facts, then the comparison should disregard the *carry-over* precision improvement in $\widetilde{\mathcal{I}}_s$.

The comparison of two relational invariants $\mathcal{I}$ and $\widetilde{\mathcal{I}}$ involves two steps: (1) identifying a changed component of each invariant at a given statement and (2) performing minimal comparison between the changed components of $\mathcal{I}$ and $\widetilde{\mathcal{I}}$. In our previous work [3] we addressed step (1) for the Zones domain where using data-flow analysis (DFA) information, we developed efficient algorithms that find a minimally changed set of inequalities in a Zone invariant.

In this work we target step (2), assuming that an abstract domain has some means to perform step (1) using either elementary or sophisticated algorithms. Thus, the contributions of this paper include: **(a)** development and analysis of a minimal comparison algorithm for relational abstract domains and **(b)** investigating its effect on comparisons between different widening techniques for Zones domain as well as comparison between Zones and incomparable Predicate domains with a relational component.

The rest of the paper is organized as follows. In Section 2, we provide the background, context, and motivation for our work. In Section 3, we describe our fixed-point algorithm. In Section 4, we explain our experimental setup and evaluation, and in Section 5, we examine the results of our experiments. We connect this work with previous research in Section 6. Finally, we conclude and discuss future work in Section 7.

2 Background and Motivation

We refer to an invariant and the corresponding abstract domain as relational if it is expressed as a conjunction of formulas over program variables, e.g., a set of linear integer inequalities. We first explain the concept of the minimal/dependent change for an invariant and then explain challenges of comparing two relational domains, and sketch how our proposed approach works.

2.1 Minimal changes in relational abstract domains

Consider the relational invariants computed by a data-flow analysis framework using the Zones abstract domain as shown in Figure 1a. Let us assume the analyzed code has four program variables: w, x, y, and z. Here, the incoming flow to the conditional statement has the following invariant: $\mathcal{I}_{in} = z \leq x \wedge w \rightarrow \top \wedge y \rightarrow \top$. That is, variables w and y are unbounded while x and z are bounded by a $\leq$ relation. The transfer function of the true branch adds the $y \leq x$ inequality, thus, making y bounded. This results in the $\mathcal{I}_t = z \leq x \wedge y \leq x \wedge w \rightarrow \top$ invariant. Similarly, the invariant for the false branch becomes $\mathcal{I}_f = z \leq x \wedge x \leq y - 1 \wedge w \rightarrow \top$.

Even though $\mathcal{I}_f$ and $\mathcal{I}_t$ are new invariants, they inherit two unchanged inequalities $z \leq x$ and $w \rightarrow \top$ from $\mathcal{I}_{in}$. This suggests that some part of a previously computed invariants have not changed by the transfer function of the conditional statement. Thus, if for some program, $\mathcal{I}_{in}$ is more precise because of $z \leq x$ and remains more precise in $\mathcal{I}_t$ because of the same inequality, such *carry-over* precision results should be disregarded.

Previous work determining minimal changes in a relational abstract domain approach [3] addresses this problem by identifying the dependent portion of the invariant affected by the statement's transfer function. For example, the minimal change algorithm for Zones [3] can compute the minimal sub-formula given the potentially changed variables x and y. Specifically, the algorithm identifies only the $y \leq x$ part of $\mathcal{I}_t$ having changed from $\mathcal{I}_{in}$. Likewise for $\mathcal{I}_f$, the algorithm identifies two inequalities: $z \leq x$ and $x \leq y - 1$ as the changed portion of the invariant[1].

The minimal change algorithm can be sophisticated and accurately compute the changed part of the invariants, or can be over-approximating, and in the worst case return the entire invariant. In our previous work we developed an efficient collection of such algorithms for the Zones abstract domain. In this work, we assume that a relational domain has an invariant change method Δ implemented, which takes as input an invariant and a set of updated variables and returns a portion of $\mathcal{I}$, e.g., in this example $\Delta(\mathcal{I}_t, \{x, y\}) = y \leq x$. The shaded regions of the invariants in the Figures 1a and 1b indicate the changed parts of the out state for each branch.

2.2 Comparing relational domains

Now consider invariants in Figure 1b computed for the same code fragment, but using an improved algorithm. This algorithm is able to compute additional information for $\widetilde{\mathcal{I}}_{in} = z \leq x \wedge w \leq y$, which is more precise than $\mathcal{I}_{in}$ since $\widetilde{\mathcal{I}}_{in}$ constrains the values of w and y. The checkmark symbol, ✓, by $\widetilde{\mathcal{I}}$ in Figure 1b indicates an increased precision comparing to the corresponding invariants $\mathcal{I}$ in Figure 1a.

[1] $z \leq x$ is included due to transitive effects through x.

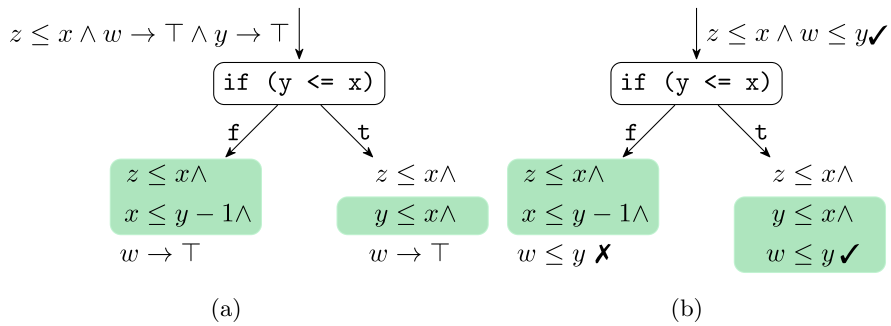

Fig. 1: Original Static Analysis (a) and Improved Static Analysis (b)

When we compare using the entirety of the invariants instead of simply the changed portion of the invariants for the false branch, the result would be that $\widetilde{\mathcal{I}_t}$ is more precise than $\mathcal{I}_t$. Thus, simply applying Δ for both invariants can filter out erroneous, *carry-over* improvements, which we annotate with the ✗ symbol.

In the case of the false branch, the set of variables in their respective changed portions of the invariants are the same. However, this is not always the case, which we can see on the true branch. There, $\Delta(\mathcal{I}_t, \{x, y\}) = y \leq x$, but $\Delta(\widetilde{\mathcal{I}_t}, \{x, y\}) = y \leq x \wedge w \leq y$ has an extra variable w. To make a sound comparison, we need to conjoin $w \to \top$ with the result of $\Delta(\mathcal{I}_t, \{x, y\})$. The challenge here is to identify the smallest necessary additions to the changed portions of the invariants to perform a sound comparison.

In the next section we present our proposed approach that addresees this problem by developing a fixed-point algorithm that, in each iteration, discovers a minimal set of inequalities (modulo Δ) in one invariant that is adequate for comparison with the changed part of the other invariant.

3 Approach

In this section, we explain the theoretical basis for our approach to minimally compare relational invariants via logical entailment. We start by defining the problem, and then we present our algorithm that solves it. At the end, we perform an analysis of the proposed algorithm.

3.1 Problem definition

We define the problem in a context of a DFA framework, where the framework provides a set of updated variables, dv, that resulted in a new invariant $\mathcal{I}$. An abstract domain for $\mathcal{I}$ has a function Δ implemented, which returns a portion of $\mathcal{I}$ that have been updated or are dependent on the variables in the set dv. In the worst case, $\Delta(\mathcal{I}, dv) = \mathcal{I}$, i.e., a transfer function affects the entire invariant. In the best case $\Delta(\mathcal{I}, dv) = \emptyset$, i.e., nothing has changed. We also introduce a

Algorithm 1 Common minimal changed variable set

Require: $V(\mathcal{I}_1) = V(\mathcal{I}_2) \wedge V(\Delta(\mathcal{I}_1, dv_1)) \subseteq V(\mathcal{I}_1) \wedge V(\Delta(\mathcal{I}_2, dv_2)) \subseteq V(\mathcal{I}_2)$
Ensure: $S_1 = S_2 \subseteq V(\mathcal{I}_1)$
function COMMONVARSET(dv_1, dv_2, $\mathcal{I}_1$, $\mathcal{I}_2$)
    $S_1 \leftarrow \mathrm{V}(\Delta(\mathcal{I}_1, dv_1))$
    $S_2 \leftarrow \mathrm{V}(\Delta(\mathcal{I}_2, dv_2))$
    while $S_1 \neq S_2$ **do**
        if $S_1 \supset S_2$ **then**
            $dv_2 \leftarrow S_1 \setminus S_2$
            $S_2 \leftarrow S_2 \cup \mathrm{V}(\Delta(\mathcal{I}_2, dv_2)))$
        else if $S_2 \supset S_1$ **then**
            $dv_1 \leftarrow S_2 \setminus S_1$
            $S_1 \leftarrow S_1 \cup \mathrm{V}(\Delta(\mathcal{I}_1, dv_1)))$
        else if $S_1 \supset\subset S_2$ **then**
            $dv_1 \leftarrow S_2 \setminus S_1$
            $dv_2 \leftarrow S_1 \setminus S_2$
            $S_1 \leftarrow S_1 \cup \mathrm{V}(\Delta(\mathcal{I}_1, dv_1)))$
            $S_2 \leftarrow S_2 \cup \mathrm{V}(\Delta(\mathcal{I}_2, dv_2)))$
        end if
    end while
    return S_1
end function

function V that returns the set of variables used in $\mathcal{I}$. For example, we use it to define the following property: $V(\Delta(\mathcal{I}, dv)) \subseteq V(\mathcal{I})$.

Let $\mathcal{I}_1$ and $\mathcal{I}_2$ be two relational invariants, and let dv_1 and dv_2 be their corresponding sets of updated variables. Then the problem of finding a minimal changed part of two invariants reduces to finding a common minimal updated set of variables S such that

$$S = V(\Delta(\mathcal{I}_1, S)) = V(\Delta(\mathcal{I}_2, S)) \tag{1}$$

A minimal solution for such recursive definitions is commonly obtained by a fixed-point iteration algorithm with initial values S_0 set to the smallest set, which in our case is $S_0 = dv_1 \cup dv_2$. If $S_0 = \emptyset$, then $dv_1 = dv_2 = \emptyset$, $\Delta(\mathcal{I}_1, dv_1) = \emptyset$, $\Delta(\mathcal{I}_2, dv_2) = \emptyset$, and, ultimately, $S = \emptyset$. That is, nothing has changed between the two invariants. However, if $S_0 \neq \emptyset$, then we need to iteratively solve for S in Equation 1.

3.2 Finding a common changed variable set

Algorithm 1 shows the pseudocode of the optimized fixed-point computation algorithm to solve Equation 1. The algorithm takes as arguments, the updated variables for each domain, dv_1 and dv_2, two invariants to compare, $\mathcal{I}_1$ and $\mathcal{I}_2$. It requires basic conditions for its correctness: each set of invariants are described over the same set of variables and Δ does not introduce any new variables. The output is the solution for Equation 1.

The algorithm first computes the initial changed variable sets, S_1 and S_2 for each invariant, lines 2 and 3, affected by the updated variables dv_1 and dv_2, respectively.

At line 4, the algorithm compares the two sets and if they are not equal, i.e., the fixed-point has not been reached, the algorithm enters the main iteration loop. Inside the body of the loop, the algorithm first tests whether one set of variables is a proper superset of the other, lines 5 and 8.

As a simple optimization, if one of the sets is a proper superset, it only augments the smaller set as done on lines 6–7 and lines 9–10, respectively. For example, if $S_1 \supset S_2$, S_2 is augmented by the variables which are not already in S_2. Afterwards, a new updated variable set is computed from the set difference of S_1 and S_2, line 6. Then, the algorithm computes the changed variable set as the union between the existing set S_2 and the newly computed minimum variables, line 7. Similar computations are done for the case when $S_2 \supset S_1$, lines 9–10.

Finally, when the changed variable sets are incomparable— line 11— then both changed variable sets are recomputed in a similar fashion as described in lines 12–15. Upon the loop's termination, i.e., when $S_1 = S_2$, the algorithm returns one of the dependent sets, line 18.

To demonstrate how Algorithm 1 compares two invariants, consider the invariants on the true branch from our example in Figure 1b. There, $\mathcal{I}_1 = z \leq x \wedge y \leq x \wedge w \rightarrow \top$ and $\mathcal{I}_2 = z \leq x \wedge y \leq x \wedge w \leq y$. The updated variables are $dv_1 = \{x, y\}$ and $dv_2 = \{x, y\}$.

The algorithm computes $\{x, y\}$ for S_1 and $\{w, x, y\}$ for S_2. Since S_2 is a proper superset of S_1, we recompute S_1, lines 9 and 10. Specifically, dv_1 becomes $\{w\}$. S_1 is then recomputed: $S_1 = S_1 \cup V(\Delta(\mathcal{I}_1, dv_1))$, which results in $S_1 = \{x, y\} \cup \{w\} = \{w, x, y\}$. At this point, $S_1 = S_2$, terminating the loop, and the algorithm returns the set $S_1 = \{w, x, y\}$. Then, an SMT solver can be used to compare logical relations of $\Delta(\mathcal{I}_1, S_1)$ and $\Delta(\mathcal{I}_2, S_1)$, for example, using implication relations. Or, in case of comparisons between Zones, one can use its custom equivalence and inclusion operations [16].

As mentioned, under worst-case conditions, Algorithm 1 returns the entire set of variables. In other words, it devolves into a full invariant comparison. This can happen if the variables within the invariant are tightly coupled with all other variables. Another situation which can cause a worst-case comparison is when an abstract domain has an ineffective Δ function, which performs a basic dependency analysis such as slicing [3,24].

Below we present termination and complexity analysis for Algorithm 1. We start with a proof sketch of termination.

Proof. First, we begin with the following assumptions: the variable projections for both domains are equivalent, i.e., $V(\mathcal{I}_1) = V(\mathcal{I}_2)$; and we assume the invariant minimization functions for each domain yield a subset of the variable projections, that is, $\Delta(\mathcal{I}_1, dv_1) \subseteq V(\mathcal{I}_1)$, and similarly for $\mathcal{I}_2$.

At each iteration, the union of variables over the minimization function is always increasing by at least one variable in either S_1 or S_2. Therefore, within a finite number of iterations S_1 and S_2 reach fixed-point, which is bounded by $V(\mathcal{I}_1) = V(\mathcal{I}_2)$ condition. Thus, Algorithm 1 terminates. □

The time-complexity of Algorithm 1 depends on the number of variables and the complexity of the Δ functions of the abstract domains. That is, the complexity of Algorithm 1 is $O(N) \cdot (C_{\Delta_1} + C_{\Delta_2})$, where N is the number of variables in the program under analysis and C_{Δ_i} is the complexity of the invariant minimization function for the corresponding domain. In the worst-case, at each iteration the sets S_1 and S_2 augmented by a single variable from Δ computations.

4 Methodology

To determine the effectiveness of the proposed algorithm, we use it to compare invariants produced by different techniques and by different relational abstract domains on the same program. For each subject program, each analysis outputs invariants after each statement. Over the corpus of programs, we compute 6564 total invariants. We store the invariants as logical formulas in SMT-LIB format. We run analyses on two relational domains, Zones and Relational Predicates [21], and compare the results of a standard Zones analysis to advanced Zones analyses, and Zones analysis to Relational Predicates analysis.

The goal of the empirical evaluation is to answer the following research questions:

RQ1 Does our technique affect the invariant comparison between different analysis techniques for the same abstract domain?

RQ2 Does our technique affect the invariant comparison between two different relational domains?

RQ3 How effective and efficient is Algorithm 1 on real-world invariant comparisons?

We consider different analysis techniques over the Zones domain to measure the precision gained by various advanced techniques. We consider the iteration parameter before widening. We also consider the widening method employed, which ensures termination for Zones analysis.

We then compare the most precise Zones technique to Relational Predicates [21], two incomparable domains. Our previous work [3] has shown the benefit of minimally comparing incomparable domains to demonstrate realized precision. However, in this case, we extend the invariants of the Predicates domain with a symbolic relational component.

For Relational Predicates, the minimization function is a selection based solely on notions of variable reachability, e.g., variable dependence, but it might not be minimal because of the generality of inequalities used in the relational

```
(push)
(forall ((w Int) (x Int) (y Int) (z Int))
        (assert (=> (and (<= z y) (<= y x))
                    (and (<= z y) (<= y x) (<= w x))))))
(check-sat)
(pop)
(push)
(forall ((w Int) (x Int) (y Int) (z Int))
        (assert (=> (and (<= z y) (<= y x) (<= w x))
                    (and (<= z y) (<= y x))))))
(check-sat)
(pop)
```

Fig. 2: Logical implication between two example abstract states in SMT-LIB.

part. We also computed minimization over Relational Predicates using a purely connected component concept, similar to the technique by Visser et al. [24], however, the reachable variant performed marginally better.

We use the Minimal Neighbors (MN) minimization function from our previous work [3] for Zones which provides the smallest invariant partition given a set of changed variables. This minimization algorithm considers the semantics of the formulas under the changed variables. Using these semantics, it selects the minimal dependent substate from the logical formula representing the invariant.

Subject programs Our subject programs consist of 192 Java methods from previous research on the Predicates domain [21]. These methods were extracted from a wide range of real-world, open-source projects and have a high number of integer operations. The subject programs range from 1 to 1993 Jimple instructions, a three address intermediate representation. The average branch count for the methods is 6 ($\sigma = 11$), with one method containing a maximal 56 branches. A plurality of our subject methods, 81 methods, contain at least one loop, with one method containing 12 loops.

Experimental platform We execute each of the analyses on a cluster of CentOS 7 GNU/Linux compute nodes, running Linux version `3.10.0-1160.76.1`, each equipped with an Intel® Xeon® Gold 6252 and 192 GB of system memory. We use an existing DFA static analysis tool [2,21] implemented in the Java programming language. The analysis framework uses Soot [20,23] version `4.2.1`. Similarly, we use Z3 [19], version `4.8.17` with Java bindings to compare SMT expressions for the abstract domain states. Finally, we use Java version 11 to execute the analyses, providing the following JVM options: `-Xms4g`, `-XX:+UseG1GC`, `-XX:+UseStringDeduplication`, and `-XX:+UseNUMA`.

Implementation We modified an existing DFA framework such that the Zones analysis outputs its entire invariant for each program point. Each invariant is further reduced using a redundant inequality reduction technique proposed by Larsen et al. [13]. For all domains, unbounded variables are set to top, $\top$, and excluded from the output expression. This further simplifies the formulas. Using the formulas from each analysis, in the usual way, we entail them into implication SMT formulas. For example, if an analysis produces $\mathcal{I}_1 = z \leq x \wedge y \leq x$ and another produces $\mathcal{I}_2 = z \leq x \wedge y \leq x \wedge w \leq y$. We entail these two expressions into the logical implication SMT query as shown in Figure 2.

After entailment, we use Z3, using the linear integer arithmetic (LIA) theory for Zones to Zones comparisons and the non-linear integer arithmetic (NIA) theory for Zones to Relational Predicates comparisons, to decide model behavior of each domain. While Zones, and numerical abstract domains in general, have understood equality mechanisms such as double inclusion based deciders, entailment allows us to determine the pre-order between the two domain instances.

Evaluations In total, we perform *three* different invariant comparisons, summarized in the following list:

$Z \preceq Z_{k=5}$— Zones using standard widening after two iterations and Zones widening after five iterations.
$Z \preceq Z_{ths}$— Zones with standard widening and Zones with threshold widening.
$Z_{ths} \prec\succ P$— Zones with threshold widening and Relational Predicates.

In all instances of Zones sans $Z_{k=5}$, widening happens after *two* iterations over widening nodes. We use a generic set of thresholds for Zones based on powers of 10: $\{0, 1, 10, 100, 1000\}$. Using a tuned set of thresholds for each program would yield better individual results, but overall does not affect our conclusions.

We use a generic disjoint domain for the basis of the Relational Predicates, based on Collberg et al.'s [6] study of numerical constants in Java Programs. Specifically, the predicate domain used in this study consists of the following set of disjoint elements: $\{(-\infty, -5],\ (-5, -2],\ -1,\ 0,\ 1,\ [2, 5),\ [5, +\infty)\}$. The relational component of the Predicates domain consists of symbolic information gathered through the process of analysis [21].

5 Evaluation Results and Discussions

In this section, we present the results of our experiments and discuss their implications to the research questions posed in the previous section.

5.1 Technique Comparisons

To answer **RQ1**, we consider the comparisons of different techniques using the Zones abstract domain. Since different techniques using the same domain create a partial ordering of their respective precision, we need only consider equivalent

Comparison	$Z \equiv Z_{k=5}$	$Z \prec Z_{k=5}$
Full	6555	9
Minimal	6562	2

Table 1: Zones $k = 2$ widening compared to Zones $k = 5$ widening

Comparison	$Z \equiv Z_{ths}$	$Z \prec Z_{ths}$
Full	6519	45
Minimal	6545	19

Table 2: Zones compared to Zones with Threshold Widening

Comparison	$Z_{ths} \equiv P$	$Z_{ths} \prec P$	$Z_{ths} \succ P$	$Z_{ths} \prec\succ P$	$Z_{ths}\, ?\, P$
Full	1227	3173	196	1947	21
Minimal	3675	2353	248	288	0

Table 3: Zones with Threshold Widening compared to Relational Predicates

and less precise outcomes. To verify correctness of our implementation, however, we ensured that no other precision outcomes occurred.

Table 1 shows the breakdown of invariants computed by standard widening after *two* iterations and standard widening after *five* iterations. Comparing invariants using the entire invariant, deferred widening produces *nine* more precise invariants. However, when using our minimized comparison technique, the slim advantage reduces to *two* invariants.

Table 2 shows the breakdown of invariants between standard widening after two iterations and threshold widening after two iterations. Here, we see the largest gain in precision. Using the entire invariant to compare, threshold widening computes 45 more precise invariants. Again, however, the precision gain is cut by more than 50% when using minimal comparisons. The choice of thresholds could improve the precision, but for best results, the set of thresholds needs to be tailored specifically to each program.

As we can see between $Z \preceq Z_{k=5}$ and $Z \preceq Z_{ths}$, our comparison technique lowers the number of more precise invariants, thus eliminating the *carry-over* precision instances. That is, our technique lowers the number of more precise invariants advanced techniques compute. However, in doing so, our technique presents a more nuanced image of the realized precision gain advanced techniques offer.

5.2 Zones versus Relational Predicates

Table 3 shows the precision breakdown of Zones with threshold widening compared to Relational Predicates, **RQ2**. Given that Zones and Predicates are inherently incomparable domains, we must consider all precision comparison categories. With the full invariant compairsions, Relational Predicates are more precise than Zones in about 50% of the invariants. The next largest category of invariants is incomparable, $\prec\succ$, which accounts for 30% of invariants. Here, Zones and Predicates are complementary, neither more nor less precise than the other. Zones and Predicates are equivalent in 19% of all invariants, and Zones are more precise in about 3% of all invariants. Finally, using the full invariant, 21 of the program points, the relation between two invariants could not be established by Z3 since it returned `UNKNOWN`.

Our technique eliminates the undecidable results. Moreover, it dramatically reduces the number of incomparable invariants– only 4% of invariants remain incomparable. Similar to *carry-over* precision, incomparable invariants arise when one domain computes a more precise invariant for one variable, and the other domain computes a more precise invariant for another, unrelated variable at a later program point. Considering the entire invariant results in incomparable precision. However, by comparing only the relevant, changed variables, our technique largely disentangles the imprecision in the comparison.

The equivalent invariant category is the next largest affected category, where more than half, 56%, of computed invariants between Zones and Relational Predicates become equivalent. Relational Predicates lose 13% of more precise invariants, and Zones gains about 1% of invariants which it computes more precisely than Relational Predicates.

By comparing only the necessary variables at each program point, our technique allows general, relational abstract domains to be compared without undecidable results. The reduction in incomparable invariants between two otherwise difficult to compare domains provides a clearer precision performance picture between the two domains.

Effect on efficiency of comparison To demonstrate the effect on efficiency of comparing our minimal comparison to the full state comparison with respect to the logical entailment and solver queries, we collected five (5) executions of the Z3 solver processing the logical entailment queries. Figure 3 shows the averaged runtime comparisons between Z3 comparing states using the entire state and our proposed minimal technique. In Figure 3 (a), we compare the runtimes for Zones versus Zones with Threshold Widening. We see the two runtimes appear similar. Indeed, a statistical t-test confirms the two distributions fail to be rejected as similar. However, in the range above the average, 0.04, the majority of the points are below the diagonal line, indicating that the minimum comparison is faster than the full comparison. This runtime behavior is expected for these two abstract domains since the two domains are similar and as shown in Table 2, the number of states where the two domains are equal is significant. In Figure 3 (b), we compare the runtimes of Zones with Threshold Widening against Relational Predicates. The average runtime for the full comparisons is about 2.7 seconds. The minimum comparison has an average of about 0.8 seconds. We see a significant difference between the two visually as the majority of points are below the diagonal line. As before, these results seem intuitive since the resulting queries for the proposed technique result in fewer invariants per abstract state. Overall, we see our technique improves the efficiency of relational domain comparison.

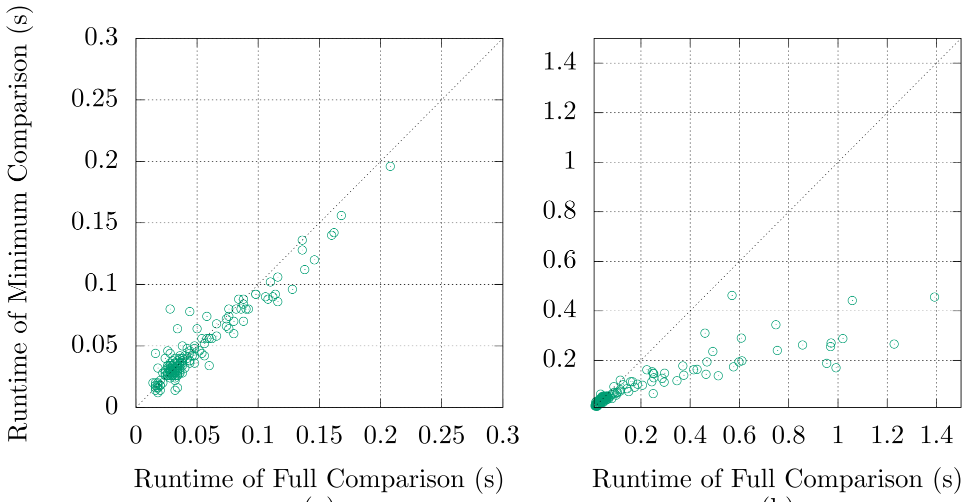

Fig. 3: Runtime (in seconds) comparisons between full and minimum invariant sets using Z3 to compute logical entailment. In (a) compares Zones to Zones with Threshold Widening. In (b) compares Zones with Threshold Widening to Relational Predicates.

5.3 Iterations and variable reductions

To determine if Algorithm 1 is efficient, **RQ3**, we use the iteration depth count to determine how many times the algorithm iterates before it reaches a stable set of variables for comparison. Over all instances of Zones comparisons, the iteration count was either *zero* or *one*, with no outliers. That is, either Zones computed the same set of changed variables and the dependent set between two techniques was immediately equivalent. Or, the set of dependent variables is captured with only a single extension, mostly to the Zones using standard widening, Z.

Comparing Zones to Relational Predicates, we see similar results. The average number of iterations is between *zero* and *one* iteration. However, we have several outliers at two iterations. Instrumentation found 12 instances of extreme outliers, 11 for *three* iterations, and one instance of *four* iterations. Furthermore, more variety exists in the branches for Zones versus Relational Predicates. Unlike comparing techniques between Zones invariants, comparing Zones to a more general, relational formula required more augmentation by each domain.

To evaluate effectiveness of Algorithm 1, **RQ3**, we consider the proportion of variables necessary for comparison. We instrumented our algorithm to compute the proportion of variables it returns after reaching a stable set, compared to the variable projection of the incoming invariants. We plot the frequency of proportions of variables returned by Algorithm 1 in Figure 4. In Figure 4a, we plot variable reductions across all comparisons of Zones: standard widening after two iterations versus standard widening after five iterations and standard

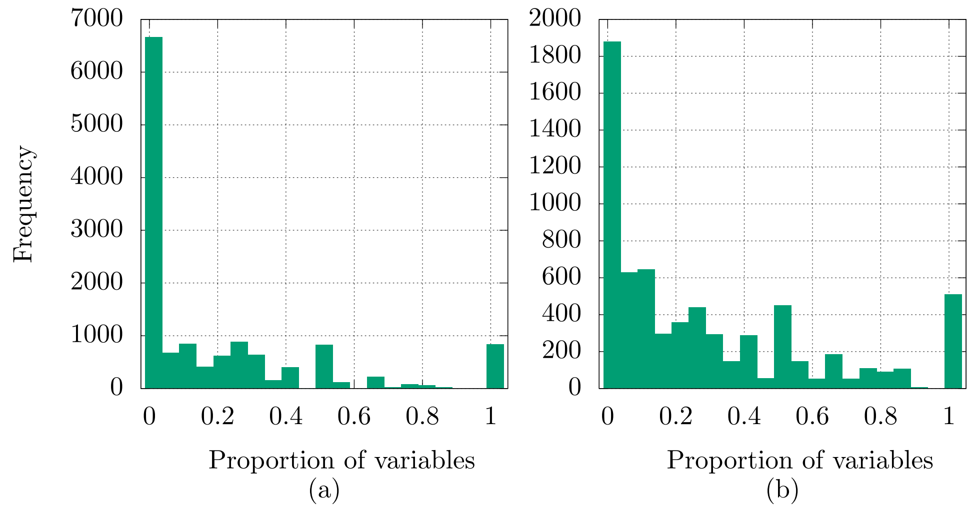

Fig. 4: Frequency plot of proportion of variables selected by Algorithm 1 which are necessary for comparing two invariants. (a) represents the frequencies of proportions when comparing techniques using Zones. (b) represents the frequencies of proportions when comparing Zones to Relational Predicates.

widening versus threshold widening. Figure 4b shows the variable reductions for Zones with threshold widening versus Relational Predicates. Considering a single bin in Figure 4, for example, 0.1, represents the frequency where Algorithm 1 needed only 10% of the variables occurring in the original invariants to adequately compare the two.

Shown in Figure 4, the large frequencies in the 0 bin shows our technique was able to remove all variables from the invariants from comparison, eliminating the need to compare the two invariants. Comparing advanced techniques utilizing Zones shows more than 6500 instances, and about 1850 in Zones versus Relational Predicates.

Our technique reduces the number of variables necessary for comparison by 50% or more in 90% of comparisons between techniques of Zones, and at least by 25% in 93% of comparisons. For Zones and Relational Predicates, our technique reduces the necessary, relevant variables by 50% or more in 80% of comparisons and by 12% in 93% of comparisons. That is, in the majority of comparisons, our technique reduces the number of variables necessary for comparing two relational domains or techniques. The quality of a domain's Δ function affects the performance and effectiveness of Algorithm 1. We see only a few iterations in the algorithm when comparing analysis techniques utilizing Zones since we used a minimal Δ function for Zones. However, we see an increase in iterations when comparing with a non-optimal Δ, as in Zones and Relational Predicates. That is, the quality of Δ can have an outsized impact on the practicality of our technique. However, given the preponderance of variable reductions and low iteration counts over the corpus of methods and comparisons, we conclude that the proposed algorithm is practical and effective.

5.4 Discussion

The evaluation results show our technique enables more precise comparison between relational abstract domain invariants. When comparing two techniques using the same domain, our minimal comparison strategy precisely captures the techniques' relative precision, disentangling accumulated *carry-over* effects from realized precision gains.

While we do not have a proven state minimization function for Relational Predicates, our technique still shows improvement when comparing incomparable relational abstract domains. Specifically, our comparison removes unknowns and dramatically reduced incomparable invariants, which makes it easier to make software engineering decisions.

The average iteration depth for Algorithm 1 shows the algorithm's efficiency and practicality. Even when using an imprecise minimization function for Relational Predicates, our technique only needed a maximum of *four* iterations to arrive at a stable set of common variables for comparison. Moreover, in the majority of comparisons, Algorithm 1 returned a significantly smaller proportion of variables than the entirety of the variables in each invariant, demonstrating the efficacy of the technique.

6 Related Work

Our previous work [3] found a set of algorithms for efficiently computing Δ for the Zones domain. Using the algorithms, it compared Zones to other non-relational domains, which in the context of data-flow analysis (DFA) and this work, have trivial Δ functions. We extend the previous work by considering comparisons between relational abstract domains, abstracting the Δ function for each domain.

Comparing the precision gain of new analysis techniques or comparing the precision of newly proposed abstract domains is a common problem in the literature. Previous work in this area generally compare precision in one of two ways. One, the comparison is based on known *a priori* program properties over benchmark programs [8,9,10,14,15]. Two, the comparison is based on logical entailment of computed invariants [10,17,21].

Close to our work, Casso et al. [5] propose several metrics for computing the *distance* between different abstract domain elements and, consequently, the distance between different analyses over those abstract domains. Thus, using distance metrics as a proxy, they are able to compute a categorization of precision over different abstract domains. However, the work and proposed metrics are constrained to non-numerical abstract domains within (Constraint) Logic Programming. We believe a combination of approaches toward (a set of) metrics that measures across different weakly-relational numerical abstract domains to be an interesting line of future work.

To the best of our knowledge, this work represents one of the first studies improving the granularity of precision characteristics for categorization of relational abstract precision comparisons. We believe this work would benefit existing work which compares relational abstract domains or new analysis techniques using relational abstract domains.

7 Conclusion and Future Work

In this study, we defined the problem of minimally comparing relational invariants, proposed an algorithm which solves the problem, and experimentally evaluated whether the algorithm indeed solves the problem using real-world programs. Using our algorithm, we can remove the precision *carry-over* effects advanced analysis techniques introduce, providing clear precision benefits for advanced techniques. For example, the benefits of deferred widening and threshold widening are smaller than anticipated. Moreover, our technique enables the comparison of relational abstract domains which are otherwise difficult to compare directly. Specifically, we see our technique removed the `UNKNOWN` invariants and dramatically reduced the incomparable invariants when comparing Zones to Relational Predicates. Finally, Algorithm 1's average iteration depth and variable reduction demonstrate the algorithm's overall practicality and usefulness when comparing analysis techniques and relational abstract domains.

Future Work Developing a minimization function, Δ for Relational Predicates would enable a comprehensive, empirical study of the relative precision of weakly-relational numerical abstract domains to Predicates. Furthermore, we believe the proposed technique of comparison can benefit adaptive analysis techniques which selectively choose the appropriate abstract domain during analysis. Similarly, an interesting, additional empirical comparison to consider is one where strictly the exit invariants are considered between domains and strategies. Octagons [18] are not included in this study because a minimization strategy for Octagons has not been developed. However, this is an interesting avenue to pursue and we intend to use the technique of this work to compare Zones to Octagons, which will empirically quantify the precision gain of Octagons over Zones.

Acknowledgments

The work reported here was supported by the U.S. National Science Foundation under award CCF-19-42044. [2]

[2] Publisher copy can be found at the following address: `https://doi.org/10.1007/978-3-031-45332-8_8`.

References

1. Abate, C., Blanco, R., Ciobâcă, c., Durier, A., Garg, D., Hritţcu, C., Patrignani, M., Tanter, E., Thibault, J.: An extended account of trace-relating compiler correctness and secure compilation. ACM Transactions on Programming Languages and Systems **43**(4), 1–48 (Dec 2021). https://doi.org/10.1145/3460860
2. Ballou, K., Sherman, E.: Incremental transitive closure for zonal abstract domain. In: Deshmukh, J.V., Havelund, K., Perez, I. (eds.) NASA Formal Methods. pp. 800–808. Springer International Publishing, Cham (2022). https://doi.org/10.1007/978-3-031-06773-0_43, `http://dx.doi.org/10.1007/978-3-031-06773-0_43`
3. Ballou, K., Sherman, E.: Identifying minimal changes in the zone abstract domain. In: David, C., Sun, M. (eds.) Theoretical Aspects of Software Engineering. pp. 221–239. Springer Nature Switzerland, Cham (2023). https://doi.org/10.1007/978-3-031-35257-7_13, `http://dx.doi.org/10.1007/978-3-031-35257-7_13`
4. Blanchet, B., Cousot, P., Cousot, R., Feret, J., Mauborgne, L., Miné, A., Monniaux, D., Rival, X.: A static analyzer for large safety-critical software. Proceedings of the ACM SIGPLAN 2003 conference on Programming language design and implementation - PLDI '03 (2003). https://doi.org/10.1145/781131.781153
5. Casso, I., Morales, J.F., López-García, P., Giacobazzi, R., Hermenegildo, M.V.: Computing abstract distances in logic programs. Lecture Notes in Computer Science p. 57–72 (2020). https://doi.org/10.1007/978-3-030-45260-5_4, `http://dx.doi.org/10.1007/978-3-030-45260-5_4`
6. Collberg, C., Myles, G., Stepp, M.: An empirical study of java bytecode programs. Software: Practice and Experience **37**(6), 581–641 (2007). https://doi.org/10.1002/spe.776
7. Cousot, P., Cousot, R.: Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: Proceedings of the 4th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages. p. 238–252. POPL '77, Association for Computing Machinery, New York, NY, USA (1 1977). https://doi.org/10.1145/512950.512973
8. Gange, G., Ma, Z., Navas, J.A., Schachte, P., Søndergaard, H., Stuckey, P.J.: A fresh look at zones and octagons. ACM Transactions on Programming Languages and Systems **43**(3), 1–51 (Sep 2021). https://doi.org/10.1145/3457885
9. Gurfinkel, A., Chaki, S.: Boxes: A symbolic abstract domain of boxes. Lecture Notes in Computer Science p. 287–303 (2010). https://doi.org/10.1007/978-3-642-15769-1_18
10. Howe, J.M., King, A.: Logahedra: A new weakly relational domain. Lecture Notes in Computer Science p. 306–320 (2009). https://doi.org/10.1007/978-3-642-04761-9_23
11. Katz, S.: Program optimization using invariants. IEEE Transactions on Software Engineering **SE-4**(5), 378–389 (Sep 1978). https://doi.org/10.1109/tse.1978.233858
12. King, J.C.: Symbolic execution and program testing. Communications of the ACM **19**(7), 385–394 (Jul 1976). https://doi.org/10.1145/360248.360252, `http://dx.doi.org/10.1145/360248.360252`
13. Larsen, K., Larsson, F., Pettersson, P., Yi, W.: Efficient verification of real-time systems: Compact data structure and state-space reduction. In: Proceedings Real-Time Systems Symposium. pp. 14–24. IEEE Comput. Soc (1997). https://doi.org/10.1109/real.1997.641265

14. Laviron, V., Logozzo, F.: Subpolyhedra: A (more) scalable approach to infer linear inequalities. Verification, Model Checking, and Abstract Interpretation p. 229–244 (2008). https://doi.org/10.1007/978-3-540-93900-9_20
15. Logozzo, F., Fähndrich, M.: Pentagons: A weakly relational abstract domain for the efficient validation of array accesses. Science of Computer Programming **75**(9), 796–807 (9 2010). https://doi.org/10.1016/j.scico.2009.04.004
16. Miné, A.: A new numerical abstract domain based on difference-bound matrices. Lecture Notes in Computer Science p. 155–172 (2001). https://doi.org/10.1007/3-540-44978-7_10
17. Miné, A.: Weakly Relational Numerical Abstract Domains (12 2004), `https://pastel.archives-ouvertes.fr/tel-00136630`
18. Miné, A.: The octagon abstract domain. Higher-Order and Symbolic Computation **19**(1), 31–100 (3 2006). https://doi.org/10.1007/s10990-006-8609-1, `http://dx.doi.org/10.1007/s10990-006-8609-1`
19. de Moura, L., Bjørner, N.: Z3: An efficient SMT solver. Lecture Notes in Computer Science p. 337–340 (2008). https://doi.org/10.1007/978-3-540-78800-3_24
20. OSS, S.: Soot (2020), `https://soot-oss.github.io/soot/`
21. Sherman, E., Dwyer, M.B.: Exploiting domain and program structure to synthesize efficient and precise data flow analyses (t). 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE) (11 2015). https://doi.org/10.1109/ase.2015.41
22. Tange, O.: Gnu parallel 20220722 ('roe vs wade') (Jul 2022). https://doi.org/10.5281/zenodo.6891516, GNU Parallel is a general parallelizer to run multiple serial command line programs in parallel without changing them.
23. Vallée-Rai, R., Co, P., Gagnon, E., Hendren, L., Lam, P., Sundaresan, V.: Soot - a java bytecode optimization framework. In: Proceedings of the 1999 Conference of the Centre for Advanced Studies on Collaborative Research. p. 13. CASCON '99, IBM Press (1999)
24. Visser, W., Geldenhuys, J., Dwyer, M.B.: Green: Reducing, reusing and recycling constraints in program analysis. Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering (Nov 2012). https://doi.org/10.1145/2393596.2393665, `http://dx.doi.org/10.1145/2393596.2393665`
25. Zhu, H., Magill, S., Jagannathan, S.: A data-driven chc solver. Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation (Jun 2018). https://doi.org/10.1145/3192366.3192416