SOFTWARE—PRACTICE AND EXPERIENCE
Softw. Pract. Exper. 0000; **00**:1–28
Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/spe

Selective Conjunction of Context-sensitivity and Octagon Domain toward Scalable and Precise Global Static Analysis

Kihong Heo[1], Hakjoo Oh[2]* Kwangkeun Yi[1]

[1] *Seoul National University*
[2] *Korea University*

SUMMARY

We present a practical technique for achieving a scalable and precise global static analysis by selectively applying context-sensitivity and the octagon relational domain. For precise analysis, context-sensitivity and relational analysis are key properties but it has been hard to practically combine both of them. Our approach turns on those precision improvement features only when the analysis is likely to improve the precision to resolve given queries. The guidance comes from an impact pre-analysis that estimates the impact of a fully context-sensitive and relational octagon analysis. We designed a cost-effective pre-analysis and implemented this method in a realistic octagon analysis for full C. The experimental results show that our approach proves 8 times more queries, while saving the time cost by 73.1% compared to a partially relational octagon analysis enabled by a syntactic heuristic.

Received . . .

KEY WORDS: Static analysis; Context-sensitive analysis; Relational analysis; Abstract interpretation

1. INTRODUCTION

In practice, both context-sensitivity and relational analysis are essential for verifying elaborate semantic properties of programs in static analysis. Context-sensitivity, such as k-callstring approach [13, 14], differentiates the analysis of a procedure by different calling contexts, which is essential for precision in global analysis; relational domain, such as the octagon abstract domain [9], reasons about the relationship between variables to verify non-trivial numerical properties.

In order to employ these techniques in a cost-effective way, the selective X-sensitive approach [12] was proposed to use context-sensitivity and relational domain cost-effectively. Given a target precision improving technique (e.g., context-sensitivity or relational analysis), the approach first runs a pre-analysis that uses the technique in a fully precise way but aggressively approximates other precision aspects. This pre-analysis estimates the impact of the X-sensitivity on the main analysis, which guides the main analysis to apply the precision only when it improves the final analysis results. In [12], the approach has been used to develop selective context-sensitive (but non-relational) analysis and selective relational (but context-insensitive) analysis.

However, in practice, it is often imprecise to separately apply those sensitivities, and a naive combination of them does not scale. For example, an octagon relational analysis can be hardly

[†] E-mail: hakjoo_oh@korea.ac.kr
[*] Correspondence to: Programming Research Laboratory, College of Informatics, Korea University, Anam-dong 5-ga, Seongbuk-gu Seoul 136-713, Korea

precise without context-sensitivity, since the analysis cannot maintain variable relationships across procedure boundaries. Suppose that a procedure is called at two call-sites, one with constraint $a = x$ and the other with $b = x$ where both a and b are variables of which values are unknown. Because the octagon domain cannot express the disjunction $a = x \ \vee \ b = x$, the procedure is analyzed with $\top$ (i.e., no relational information). We observed that, in real C programs, queries that require such inter-procedural reasoning on variable relationships are prevalent (Section 7). Unfortunately, the previous selective X-sensitive analyses estimate each sensitivity separately so that it cannot select those queries. Furthermore, naively combing the existing pre-analyses suffers from the huge cost of the combinatory sensitivities.

In this paper, we present a practical technique that selectively combines context-sensitivity and the octagon relational analysis. The basic idea is the existing selective X-sensitive approach [12]. We take both the octagon domain and context-sensitivity into account simultaneously: 1) we design a pre-analysis that conservatively estimates the "symbiotic" effect of both the fully context-sensitive and fully relational octagon analysis; 2) the pre-analysis predicts where the combination of the two precision-improving techniques will help to prove given queries; 3) the selective main analysis is derived from the guidance of the pre-analysis results. Our new analysis is different from the previous one [12] in two aspects. First, our analysis is selective both in tracked variable relations and context-sensitivity while the previous approach is selective only in each sensitivity. Second, we propose a cost-effective pre-analysis equipped with the summary-based context-sensitivity [13]. Instead of computing all possible contexts, the new pre-analysis differentiates only a particular set of contexts related to queries that is derived from a backward pre-analysis.

Our experimental results show that the selective context-sensitive and relational analysis is precise and scalable. We implemented the selective analysis on top of our Sparrow framework [15]. In experiments with various C benchmarks, the analysis scaled up to 100KLOC and proved 201 queries among 206 queries. In the comparison with the conventional octagon analysis with syntactic variable packing [9], the new analysis proves 8 times more queries, saving 73.1% of the analysis's time overhead on average.

Contributions Our contributions are as follows:

- We present a new method for achieving context-sensitive relational analysis that is precise and also scalable. We use the selective X-sensitive approach to develop an analysis that applies the octagon relational analysis and context-sensitivity only when they are necessary.
- To this end, we design a cost-effective impact pre-analysis. The pre-analysis aims at predicting where context-sensitivity and relational information would help during the main analysis. We provide a practical design to tame the huge sensitivities instead of naively combining the existing pre-analyses.
- We experimentally show the effectiveness of our approach on an industrial-strength static analyzer for C. Our approach proved 8 times more queries with 73.1% less overhead compared to the existing syntactic heuristic-based approach on average.

2. OVERVIEW

We illustrate our approach with a simple example program in Figure 1. The program contains two queries at lines 9 and 14. The first query asks whether variable `x` is greater than `n`, and the second one asks the question for variables `z` and `m`. The goal of static analysis is to prove (or disprove) the queries. Note that the first query always holds but the second one does not. When procedure `f` is called (from call-sites 21 and 24), the value of parameter `x` is always greater than parameter `y`, i.e., `x` $>$ `y` in the body of `f`. By the conditional statements at lines 5–7, `y` $>$ `n` holds at line 9, and hence `x` $>$ `n`. On the other hand, at line 13, `m` may be greater than `z` depending on the user input.

DOI: 10.1002/spe

```
1  int inc(int i) { return i + 1; }
2
3  void f(int x, int y){ // x > y
4    n = input();
5    if(n >= y){
6      y = n + 64;
7      x = y;
8    }
9    assert(n < x);  // Query 1
10 }
11
12 void g(int z){
13   m = input();
14   assert(m < z);  // Query 2
15 }
16 void main(){
17   b = input();
18   d = input();
19
20   a = inc(b);
21   f(a, b);  // a > b
22
23   c = inc(d);
24   f(c, d);  // c > d
25
26   g(a);
27   g(c);
28 }
```

Figure 1. Example Program

Context-Sensitive & Relational Analysis In order to prove the first query, a static analysis must be *both* context-sensitive and relational. Non-relational analyses, such as interval analysis, fail (even with context-sensitivity), simply because they are unable to express invariants like $\mathtt{x} > \mathtt{y}$ and $\mathtt{y} > \mathtt{n}$. Relational analyses such as octagon analysis are able to express such invariants but cannot properly maintain them without context-sensitivity. For example, suppose that at lines 21 and 24 (after parameter binding) an octagon analysis infers $\mathtt{a} > \mathtt{b} \wedge \mathtt{x} = \mathtt{a} \wedge \mathtt{y} = \mathtt{b}$ (hence $\mathtt{x} > \mathtt{y}$) and $\mathtt{c} > \mathtt{d} \wedge \mathtt{x} = \mathtt{c} \wedge \mathtt{y} = \mathtt{d}$ (hence $\mathtt{x} > \mathtt{y}$), respectively. Without context-sensitivity, these two input states are merged, but because the octagon domain is not disjunctive, the merged state is represented by $\top$.[†] Thus, the relationship $\mathtt{x} > \mathtt{y}$ is lost in the body of procedure $\mathtt{f}$.

Need of Selective Approach However, the fully context-sensitive and relational octagon analysis is prohibitively impractical. Notwithstanding its impractical performance, such a full precision is not even necessary to prove given queries. For instance, in order to prove the query at line 9, it is necessary to apply context-sensitivity only to call-sites 20, 21, 23, and 24. Furthermore, we do not need to track, for example, relationships between variables $\mathtt{a}$ and $\mathtt{c}$. For the query at line 14, it is useless to apply context-sensitivity and relational domain, since the query is impossible to prove.

Our Selective Context-Sensitive & Relational Analysis Our approach applies context-sensitivity and the octagon relational domain only when they are needed. We first run a pre-analysis and predict that applying context-sensitivity and relational analysis is likely to help prove the query at line 9, but is unlikely to help answer the query at line 14. The pre-analysis also finds out the sensitivities to answer the first query: it is sufficient to analyze 1) variable relationships within $\{\mathtt{a}, \mathtt{b}, \mathtt{i}, \mathtt{x}, \mathtt{y}\}$ while distinguishing call-sites 20 and 21, and 2) variable relationships within $\{\mathtt{c}, \mathtt{d}, \mathtt{i}, \mathtt{x}, \mathtt{y}\}$ while distinguishing call-sites 23 and 24. Then, the main octagon analysis runs in a selective context-sensitive and relational way using this information, which still proves the query at line 9.

Impact Pre-Analysis We design the pre-analysis following the approach in [12]. That is, we design an impact pre-analysis that estimates the impact of the fully context-sensitive and relational octagon analysis. The pre-analysis is fully precise in keeping context-sensitivity and variable relations while abstracting other precision aspects.

[†] For simplicity, we used an imprecise join operation. In practice, however, context-sensitivity of relational analyses is still required to represent disjunctive properties, for example, when pointers are involved.

The pre-analysis uses a simple abstract domain of which element is a sound approximation of a set of the octagons [9]. While the octagon domain represents a set of constraints in $\pm x \pm y \leq c$ where $x, y \in \mathsf{Var}, c \in \mathbb{Z} \cup \{+\infty\}$, the pre-analysis domain describes a set of abstract constraints of $\pm x \pm y \sqsubseteq v$ where $v \in \{\bigstar, \top\}$. Here, $\bigstar$ means finite integers and $\top$ includes $+\infty$. The abstraction is characterized by γ such that $\gamma(\bigstar) = \mathbb{Z}$ and $\gamma(\top) = \mathbb{Z} \cup \{+\infty\}$. Intuitively, the domain used in the pre-analysis only distinguishes whether the difference of two variables is finite ($\bigstar$) or cannot be bounded ($\top$). For example, the assignment at line 6 produces two abstract constraints $\mathtt{y} - \mathtt{n} \sqsubseteq \bigstar$ and $\mathtt{n} - \mathtt{y} \sqsubseteq \bigstar$.

For efficiency, we further abstract the fully context-sensitive pre-analysis by only distinguishing a set of specific input states related to queries. To this end, a backward pre-analysis infers the weakest pre-conditions for each procedure that are sufficient to select each query in a procedure. For example, the backward analysis for the first query starts from line 9 in Figure 1 and infers the sufficient pre-condition at line 3 as $\mathtt{y} - \mathtt{x} \sqsubseteq \bigstar$, for the query to hold. The approximated pre-analysis only differentiates the calling context of `f` whether the inferred pre-condition holds or not.

We run this new pre-analysis and select queries that are proven in the pre-analysis domain. For example, the first query is selected because $\mathtt{n} - \mathtt{x} \sqsubseteq \bigstar$ holds in the pre-analysis. Finally, we identify the call-sites to apply context-sensitivity and the variable groups to relate together. The sensitivities are derived by observing contexts and relationships that contribute the query to be $\bigstar$.

3. PROGRAMS

We assume that a program is represented by a control-flow graph $(\mathbb{C}, \rightarrow, \mathbb{F}, \iota)$, where $\mathbb{C}$ denotes the set of program points, $(\rightarrow) \subseteq \mathbb{C} \times \mathbb{C}$ denotes control-flow edges between program points. $\mathbb{F}$ is the set of procedure ids and $\iota \in \mathbb{C}$ is the entry node of the `main` procedure. Node $c \in \mathbb{C}$ in the program is one of the five types:

$$\begin{array}{rcllllll} \mathbb{C} & = & \mathbb{C}_e & \text{(Entry Nodes)} & \uplus\ \mathbb{C}_x & \text{(Exit Nodes)} & & \\ & \uplus & \mathbb{C}_c & \text{(Call Nodes)} & \uplus\ \mathbb{C}_r & \text{(Return Nodes)} & \uplus\ \mathbb{C}_i & \text{(Internal Nodes)} \end{array}$$

Each procedure $f \in \mathbb{F}$ has a single entry and exit node. In case the function has multiple exit nodes — return statements in C —, we replace them, except for the last one, by goto statements whose destinations are the last exit node of the function. We write $\mathsf{entryof}(f)$ and $\mathsf{exitof}(f)$ for these nodes. Given node $c \in \mathbb{C}$, $\mathsf{fid}(c)$ denotes the procedure enclosing the node. $\mathbb{C}_f$ denotes a set of all program points in f. Each call-site in the program is represented by a pair of call and return nodes. Given return node $c \in \mathbb{C}_r$, we write $\mathsf{callof}(c)$ for the corresponding call node. We assume for simplicity that there are no indirect function calls such as calls via function pointers (In our implementation, we resolve all function pointers using a flow-insensitive pointer analysis, so that no indirect calls appear in the main analysis).

For simplicity, we handle parameter passing and return values of procedures via simple syntactic encoding. Recall that we represent call statement $x := f_p(e)$ (where p is a formal parameter of procedure f) with call and return nodes. In our program, the call node has command $p := e$, so that actual parameter e is assigned to formal parameter p. For return values, we assume that each procedure f has variable r_f and the return value is assigned to r_f: that is, we represent return statement `return` e of procedure f by $r_f := e$. The return node has command $x := r_f$, so that the return value is assigned to the original return variable. We assume that all variables have unique names. For brevity, we further assume that there are no recursive procedures. [‡]

We assume that each node $c \in \mathbb{C}$ contains a single primitive command. We interchangeably write c for both a program point and its corresponding primitive command. For simplicity, we consider

[‡] Our approach does not apply context-sensitivity to recursive functions and analyzes them context-insensitively. Note that we cannot use the functional approach to context-sensitivity [13], because the domain of octagons is infinite. Applying the functional approach in octagon analysis is an open problem and beyond the scope of this paper.

DOI: 10.1002/spe

primitive commands in the following grammar:

$$c \quad \rightarrow \quad \mathit{skip} \mid x := k \mid x := \pm y + k \mid \pm x \leq k \mid \pm x \pm y \leq k \mid x :=? \mid x \leq ?$$

where $k \in \mathbb{Z}$ denotes an integer constant and $x, y \in \mathsf{Var}$ range over program variables. The first five primitive commands are the ones whose semantics can be precisely tracked by the octagon domain: the octagon domain can precisely handle assignments of the forms $x := k$ and $x := \pm y + k$, and guards of the forms $\pm x \leq k$ and $\pm x \pm y \leq k$. The last two primitive commands represent all the other types of commands whose semantics is too complex to be described in the octagon domain, such as non-octagonal assignments like $x := ay + b$ and non-octagonal guards like $x \leq ay + b$ ($a \neq 0, \pm 1$).

We assume that a set of queries ($\mathcal{Q} \subseteq \mathbb{C} \times \mathsf{Var} \times \mathsf{Var}$) is given in the program. Query $(c, x, y) \in \mathcal{Q}$ represents a predicate on integer $y - x < 0$ at program point c.

Other Language Features Although we use a subset of C for presentation, our implementation supports full C, including pointer arithmetics, function pointers, and recursions. We pre-run a flow-insensitive and allocation-site-based pointer analysis to resolve pointers (including function pointers) in the program. For each abstract location (i.e., allocation-site), we maintain the size, offset, and type information of allocation-sites, which allows us to handle pointer arithmetics and type casts. For example, consider the type cast `p = (char*)a`, where `a` points to an array of 10 integers. After the type casting, the pointer `p` points to an array of 40 characters. All elements of an array are merged into a single abstract cell (i.e., array smashing). Ignoring recursive functions is only for presentation brevity; in our approach, recursive functions are analyzed context-insensitively (see Section 7). More detailed information about the baseline analyzer (Sparrow) is available in [11].

4. SELECTIVE CONTEXT-SENSITIVE & SELECTIVE RELATIONAL ANALYSIS

In this section, we design a class of octagon analyses that is parameterized by context-sensitivity and variable packs. We begin with the definition of the conventional octagon analysis (Section 4.1). We first extend the analysis to a *packed octagon analysis* that selectively tracks the relationships between variables (Section 4.2); the octagon for a *pack* of variables keeps only the octagonal constraints for those variables. Then, we extend the resulting analysis to its selective context-sensitive version (Section 4.3).

4.1. The Basic Octagon Analysis [9]

The octagon domain aims to track the lower and upper bounds of $x + y$ and $x - y$ for all program variables x and y. That is, an octagon is represented by a set of *octagonal constraints* of form $\pm x \pm y \leq k$ where $x, y \in \mathsf{Var}$ and k is an element in $\mathbb{Z}_\infty = \mathbb{Z} \cup \{+\infty\}$.

A set of octagonal constraints is represented by a *difference bound matrix* on $\mathbb{Z}_\infty$ (DBM, for short). Let $\bar{x}$ denote the negative form (i.e. $-x$) of variable $x \in \mathsf{Var}$. Also, let Var be the set of variables in Var and their negative forms: $\mathit{Var} = \mathsf{Var} \cup \{\bar{x} \mid x \in \mathsf{Var}\}$. Then, every octagonal constraint over Var can be represented by *potential constraints* over Var of form $j - i \leq k$ where $i, j \in \mathit{Var}$. For example, octagonal constraint $x + y \leq 1$ is represented by two potential constraints $x - \bar{y} \leq 1$ and $y - \bar{x} \leq 1$. Finally, octagonal constraints over variables in Var are represented by $2n \times 2n$ DBM where $n = |\mathsf{Var}|$. Each entry o_{ij} of DBM o represents one potential constraint $j - i \leq o_{ij}$ where $i, j \in \mathit{Var}$ and $o_{ij} \in \mathbb{Z}_\infty$. Note that not every DBM represents octagonal constraints and only coherent DBMs do (in the following, $\bar{\bar{x}} = x$): $\forall o, i, j \in \mathit{Var}.\ o_{ij} = o_{\bar{j}\bar{i}}$.

The octagon domain is a complete lattice $(\mathbb{O}, \sqsubseteq_{\mathbb{O}}, \bot_{\mathbb{O}}, \top_{\mathbb{O}}, \sqcup_{\mathbb{O}}, \sqcap_{\mathbb{O}})$ where $\mathbb{O}$ is the set of coherent DBMs, $\bot_{\mathbb{O}}$ is the bottom element, and $\top_{\mathbb{O}}$ is the top element such that $(\top_{\mathbb{O}})_{ij} = +\infty$ for all i, j. The partial order $\sqsubseteq_{\mathbb{O}}$, join $\sqcup_{\mathbb{O}}$, and meet operator $\sqcap_{\mathbb{O}}$ are defined as pointwise liftings of those on $\mathbb{Z}_\infty$.

$$\llbracket x := k \rrbracket(o) = o'^{\bullet} \text{ where } o'_{ij} = \begin{cases} -2k & (i = x, j = \bar{x}) \\ 2k & (i = \bar{x}, j = x) \\ \big(\llbracket x :=? \rrbracket(o)\big)_{ij} & \textit{otherwise} \end{cases}$$

$$\llbracket x := y + k \rrbracket(o) = o'^{\bullet} \text{ where } o'_{ij} = \begin{cases} -k & (i = x, j = y \text{ or } i = \bar{y}, j = \bar{x}) \\ k & (i = y, j = x \text{ or } i = \bar{x}, j = \bar{y}) \\ \big(\llbracket x :=? \rrbracket(o)\big)_{ij} & \textit{otherwise} \end{cases}$$

$$\big(\llbracket x := x + k \rrbracket(o)\big)_{ij} = \begin{cases} o_{ij} - k & (i = x, j \notin \{x, \bar{x}\} \text{ or } i \notin \{x, \bar{x}\}, j = \bar{x}) \\ o_{ij} + k & (i = \bar{x}, j \notin \{x, \bar{x}\} \text{ or } i \notin \{x, \bar{x}\}, j = x) \\ o_{ij} - 2k & (i = x, j = \bar{x}) \\ o_{ij} + 2k & (i = \bar{x}, j = x) \\ o_{ij} & \textit{otherwise} \end{cases}$$

$$\big(\llbracket x := -x \rrbracket(o)\big)_{ij} = \begin{cases} o_{\bar{i}j} & (i \in \{x, \bar{x}\}, j \notin \{x, \bar{x}\}) \\ o_{i\bar{j}} & (i \notin \{x, \bar{x}\}, j \in \{x, \bar{x}\}) \\ o_{\bar{i}\bar{j}} & (i, j \in \{x, \bar{x}\}) \\ o_{ij} & \textit{otherwise} \end{cases}$$

$$\llbracket x := -y \rrbracket = \llbracket x := -x \rrbracket \circ \llbracket x := y \rrbracket$$

$$\llbracket x := -x + k \rrbracket = \llbracket x := x + k \rrbracket \circ \llbracket x := -x \rrbracket$$

$$\llbracket x := -y + k \rrbracket = \llbracket x := x + k \rrbracket \circ \llbracket x := -y \rrbracket$$

$$\big(\llbracket x :=? \rrbracket(o)\big)_{ij} = \begin{cases} +\infty & (i \in \{x, \bar{x}\}, j \notin \{x, \bar{x}\} \text{ or } i \notin \{x, \bar{x}\}, j \in \{x, \bar{x}\}) \\ 0 & (i = j = x \text{ or } i = j = \bar{x}) \\ o_{ij} & \textit{otherwise} \end{cases}$$

$$\big(\llbracket x \leq k \rrbracket(o)\big)_{ij} = o'^{\bullet} \text{ where } o'_{ij} = \begin{cases} \min(o_{ij}, 2k) & (i = \bar{x}, j = x) \\ o_{ij} & \mathit{otherwise} \end{cases}$$

$$\big(\llbracket -x \leq k \rrbracket(o)\big) = o'^{\bullet} \text{ where } o'_{ij} = \begin{cases} \min(o_{ij}, 2k) & (i = x, j = \bar{x}) \\ o_{ij} & \mathit{otherwise} \end{cases}$$

$$\big(\llbracket x - y \leq k \rrbracket(o)\big) = o'^{\bullet} \text{ where } o'_{ij} = \begin{cases} \min(o_{ij}, k) & (i = y, j = x \text{ or } i = \bar{x}, j = \bar{y}) \\ o_{ij} & \mathit{otherwise} \end{cases}$$

$$\big(\llbracket x + y \leq k \rrbracket(o)\big) = o'^{\bullet} \text{ where } o'_{ij} = \begin{cases} \min(o_{ij}, k) & (i = \bar{y}, j = x \text{ or } i = \bar{x}, j = y) \\ o_{ij} & \mathit{otherwise} \end{cases}$$

$$\big(\llbracket -x - y \leq k \rrbracket(o)\big) = o'^{\bullet} \text{ where } o'_{ij} = \begin{cases} \min(o_{ij}, k) & (i = y, j = \bar{x} \text{ or } i = x, j = \bar{y}) \\ o_{ij} & \mathit{otherwise} \end{cases}$$

$$\llbracket x \leq ? \rrbracket(o) = o$$

Figure 2. Abstract semantics of primitive commands with octagon domain.

Among many DBMs that represent the same set of octagonal constraints, there exists the smallest element which is strongly closed. We write $o^{\bullet}$ for o's strong closure. DBM o is strongly closed if and only if the following conditions are satisfied:

$$\begin{cases} \forall i, j, k \in \mathit{Var}. & o_{ij} \leq o_{ik} + o_{kj} \\ \forall i, j \in \mathit{Var}. & o_{ij} \leq (o_{i\bar{i}} + o_{\bar{j}j})/2 \\ \forall i \in \mathit{Var}. & o_{ii} = 0. \end{cases}$$

DOI: 10.1002/spe

Figure 2 shows the abstract semantics $[\![c]\!] \in \mathbb{O} \to \mathbb{O}$ of primitive commands when the input state is not $\perp_{\mathbb{O}}$ (for all command c, $[\![c]\!](\perp_{\mathbb{O}}) = \perp_{\mathbb{O}}$). The first two and the middle five cases explicitly enforce the resulting octagon to be strongly closed. Instead, $[\![x := x + k]\!]$ preserves strong closures. $[\![x :=?]\!]$ forgets the constraints that involve the assigned variable, as the octagon domain cannot track this case precisely. [§]

Example 1. For example, consider the following example program:

```
int a = b;
int c = input();            // User input
for (i = 0; i < b; i++) {
  assert (i < a);           // Query
}
```

At the query point of the program, the octagon analysis computes the following DBM:

$$\begin{array}{r|c|c|c|c|c|c|c|c}
 & \mathtt{a} & -\mathtt{a} & \mathtt{b} & -\mathtt{b} & \mathtt{c} & -\mathtt{c} & \mathtt{i} & -\mathtt{i} \\ \hline
\mathtt{a} & 0 & \infty & 0 & \infty & \infty & \infty & -1 & \infty \\ \hline
-\mathtt{a} & \infty & 0 & \infty & 0 & \infty & \infty & \infty & \infty \\ \hline
\mathtt{b} & 0 & \infty & 0 & \infty & \infty & \infty & -1 & \infty \\ \hline
-\mathtt{b} & \infty & 0 & \infty & 0 & \infty & \infty & \infty & \infty \\ \hline
\mathtt{c} & \infty & \infty & \infty & \infty & 0 & \infty & \infty & \infty \\ \hline
-\mathtt{c} & \infty & \infty & \infty & \infty & \infty & 0 & \infty & \infty \\ \hline
\mathtt{i} & \infty & \infty & \infty & \infty & \infty & \infty & 0 & \infty \\ \hline
-\mathtt{i} & \infty & -1 & \infty & -1 & \infty & \infty & \infty & 0
\end{array} \tag{1}$$

The ij-th entry m_{ij} of this matrix means an upper bound $e_j - e_i \leq m_{ij}$, where e_j and e_i are expressions associated with the j-th column and the i-th row of the matrix respectively and they are variables with or without the minus sign. Note that the matrix records -1 and ∞ as upper bounds for $\mathtt{i} - \mathtt{a}$, which means that the value of `i` is less than the value of `a` and the query is proved.

4.2. *Packed Octagon Analysis*

We consider a domain of *packed octagons* that assigns an octagon to each group of variables, *pack*. The octagon associated with a pack expresses only the octagonal constraints over the variables in that pack. The intuition behind the packed octagon domain is that not every pair of variables has a meaningful relationship and only a portion of them contributes to the precision of the analysis. We call a set $\Pi \subseteq \wp(\mathsf{Var})$ of packs as *packing configuration*. We assume that a packing configuration Π covers the set of program variables ($\bigcup \Pi = \mathsf{Var}$), and all packs are disjoint. Packed octagon domain $\mathbb{PO}(\Pi)$ parameterized by packing configuration Π is a complete lattice $(\mathbb{PO}(\Pi), \sqsubseteq_{\mathbb{PO}}, \perp_{\mathbb{PO}}, \top_{\mathbb{PO}}, \sqcup_{\mathbb{PO}}, \sqcap_{\mathbb{PO}})$ where

$$\mathbb{PO}(\Pi) = \Pi \to \mathbb{O}.$$

The domain operators (join, meet, widening, strong closure, etc), the top and the bottom elements can be obtained from pointwise lifting of those of the octagon domain.

[§] To restore the precision to some degree, several methods are proposed to model more elaborate constraints in the octagon domain, but are orthogonal to our method.

We extend the abstract semantics of commands $[\![c]\!] : \mathbb{O} \to \mathbb{O}$ to $[\![c]\!]^{\Pi} : \mathbb{PO}(\Pi) \to \mathbb{PO}(\Pi)$ as follows:

$$[\![c]\!]^{\Pi}(po) = \lambda\pi \in \Pi. \begin{cases} [\![x := k]\!](po(\pi)) & (c = x := k \ \wedge\ x \in \pi) \\ [\![x := \pm y + k]\!](po(\pi)) & (c = x := \pm y + k \ \wedge\ x \in \pi \ \wedge\ y \in \pi) \\ [\![x \leq k]\!](po(\pi)) & (c = x \leq k \ \wedge\ x \in \pi) \\ [\![x \leq \pm y + k]\!](po(\pi)) & (c = x \leq \pm y + k \ \wedge\ x \in \pi \ \wedge\ y \in \pi) \\ [\![x :=?]\!](po(\pi)) & (c = x := y + k \ \wedge\ x \in \pi \ \wedge\ y \notin \pi) \\ [\![x \leq ?]\!](po(\pi)) & (c = x \leq y + k \ \wedge\ x \in \pi \ \wedge\ y \notin \pi) \\ [\![x :=?]\!](po(\pi)) & (c = x :=? \ \wedge\ x \in \pi) \\ [\![x \leq ?]\!](po(\pi)) & (c = x \leq ? \ \wedge\ x \in \pi) \\ po(\pi) & \textit{otherwise} \end{cases}$$

The extended abstract semantics transforms the octagon of a pack only when the primitive command can update any variable in the pack. All the cases are obvious pointwise liftings of $[\![c]\!]$ except the fifth and sixth one. If a pack misses some variables in the commands (i.e. the command is $x := y + k$ and y is not in the pack), the abstract semantics cannot establish a new octagonal constraint, but forgets the relations for the assigned variable.

Example 2. Consider the program in Example 1 and the variable pack $\Pi = \{\{a, b, i\}, \{c\}\}$. Then, the packed octagon analysis computes the following DBM at the query point:

	a	−a	b	−b	i	−i
a	0	∞	0	∞	-1	∞
−a	∞	0	∞	0	∞	∞
b	0	∞	0	∞	-1	∞
−b	∞	∞	∞	0	∞	∞
i	∞	∞	∞	∞	0	∞
−i	∞	-1	∞	-1	∞	∞

(2)

Note that the matrix still records -1 as upper bounds for `i-a` and therefore the analysis can prove the query. This analysis has the same precision as the original octagon analysis with reduced cost.

4.3. Context-Sensitive, Packed Octagon Analysis

Now, following [12], we extend the packed octagon analysis to be selectively context-sensitive. The context-sensitivity of the analysis is determined by a context selector. Context selector K maps procedures to sets of calling contexts, which are defined as sequences of call nodes:

$$K \in \mathbb{F} \to \wp(\mathbb{C}_c^*)$$

where $\mathbb{C}_c^*$ denotes the set of sequences of call-sites. We abuse the notation and denote by K the entire set of calling contexts in K: that is, we write $\kappa \in K$ for $\kappa \in \bigcup_{f \in \mathbb{F}} K(f)$. Nodes of the control-flow graph are enriched to a set $\mathbb{C}_K \subseteq \mathbb{C} \times \mathbb{C}_c^*$ by contexts in K:

$$\mathbb{C}_K = \{(c, \kappa) \mid c \in \mathbb{C} \ \wedge\ \kappa \in K(\mathsf{fid}(c))\}.$$

Control flow relation $(\to) \subseteq \mathbb{C} \times \mathbb{C}$ is also extended to $\to_K$ on $\mathbb{C}_K$:

Definition 1 ($\to_K$). $(\to_K) \subseteq \mathbb{C}_K \times \mathbb{C}_K$ is the context-enriched control flow relation:

$$(c, \kappa) \to_K (c', \kappa') \ \textit{iff}$$
$$\begin{cases} c \to c' \ \wedge\ \kappa' = \kappa & (c' \notin \mathbb{C}_e \uplus \mathbb{C}_r) \\ c \to c' \ \wedge\ \kappa' = c ::_K \kappa & (c \in \mathbb{C}_c \ \wedge\ c' \in \mathbb{C}_e) \\ c \to c' \ \wedge\ \kappa = \mathsf{callof}(c') ::_K \kappa' & (c \in \mathbb{C}_x \ \wedge\ c' \in \mathbb{C}_r) \end{cases}$$

where $(::_\kappa) \in \mathbb{C}_c \times \mathbb{C}_c^* \to \mathbb{C}_c^*$ updates contexts according to κ:

$$c ::_K \kappa = \begin{cases} c \cdot \kappa & (c \cdot \kappa \in K) \\ \epsilon & \textit{otherwise} \end{cases}$$

where ϵ is the empty call sequence.

DOI: 10.1002/spe

Next, we define the context-sensitive and packed octagon analysis. Abstract domain $\mathbb{D}$ of the analysis is defined with K and Π:

$$\mathbb{D} = \mathbb{C}_K \to \mathbb{PO}(\Pi)$$

The analysis keeps multiple abstract states at each program node c, one for each context $\kappa \in K(\mathsf{fid}(c))$. Abstract semantic function $F : \mathbb{D} \to \mathbb{D}$ is defined as follows:

$$F(X) = X_0 \sqcup \mathit{Next}(X)$$

where X_0 is the initial state. Next is defined with $[\![c]\!]^{\Pi}$ and $\to_K$:

$$\mathit{Next}(X)(c, \kappa) = [\![c]\!]^{\Pi}\big(\bigsqcup\{X(c_0, \kappa_0) \mid (c_0, \kappa_0) \to_K (c, \kappa)\}\big).$$

5. IMPACT PRE-ANALYSIS AND ITS USE

The goal of our pre-analysis is to estimate the behavior of the octagon analysis that is fully relational and context-sensitive at the same time. At each program point, the pre-analysis computes a sound approximation of a set of possible octagons during the main analysis and considers all the possible calling contexts and variable relationships (that is, the pre-analysis is fully context-sensitive and fully relational). Yet, the pre-analysis is more efficient than the main analysis because its numerical domain is approximated.

In Section 5.1, we design such an impact pre-analysis. In Section 5.2, we use the pre-analysis results to derive a context selector and a packing configuration for the main octagon analysis. Also, we prove that the resulting selective main analysis is guaranteed to benefit from the pre-analysis results (Proposition 1).

5.1. Abstract Domain and Semantics

The pre-analysis uses a simple relational domain of which element approximates a set of the octagons. Just like the octagon analysis, our pre-analysis aims to track lower and upper bounds of $x + y$ and $x - y$ for all program variables x and y. However, unlike the octagon domain, our pre-analysis approximately tracks the bounds in $\{\bigstar, \top_{\mathbb{V}}\}$, distinguishing only the information whether the bound can be $+\infty$ ($\top_{\mathbb{V}}$) or not ($\bigstar$). [¶] For instance, whenever our pre-analysis computes *abstract* constraint $x - y \sqsubseteq \bigstar$ at a program point, the main octagon analysis is likely to compute $x - y \leq n$ where $n \neq +\infty$ at the same program point. On the other hand, the pre-analysis does not precisely predict the bound information in case of $\top_{\mathbb{V}}$. By tracking only the binary values ($\bigstar$ and $\top_{\mathbb{V}}$), instead of the infinitely many elements in $\mathbb{Z}_{\infty}$, we can efficiently run a fully relational analysis as a pre-analysis.

Example 3. Consider the program in Example 1. The pre-analysis computes the following matrix for the query.

$$\begin{array}{r|c|c|c|c|c|c|c|c}
 & \texttt{a} & -\texttt{a} & \texttt{b} & -\texttt{b} & \texttt{c} & -\texttt{c} & \texttt{i} & -\texttt{i} \\ \hline
\texttt{a} & \bigstar & \top & \bigstar & \top & \top & \top & \bigstar & \top \\ \hline
-\texttt{a} & \top & \bigstar & \top & \bigstar & \top & \top & \top & \top \\ \hline
\texttt{b} & \bigstar & \top & \bigstar & \top & \top & \top & \bigstar & \top \\ \hline
-\texttt{b} & \top & \bigstar & \top & \bigstar & \top & \top & \top & \top \\ \hline
\texttt{c} & \top & \top & \top & \top & \bigstar & \top & \top & \top \\ \hline
-\texttt{c} & \top & \top & \top & \top & \top & \bigstar & \top & \top \\ \hline
\texttt{i} & \top & \top & \top & \top & \top & \top & \bigstar & \top \\ \hline
-\texttt{i} & \top & \bigstar & \top & \bigstar & \top & \top & \top & \bigstar
\end{array} \tag{3}$$

Each entry of this matrix stores the pre-analysis's prediction on whether the main octagon analysis would be precise or not, i.e., whether putting a *finite* upper bound at the corresponding entry of its

[¶] We do not consider $-\infty$ because $\mathbb{Z}_{\infty}$ serves as upper bounds of octagons; octagons do not have $-\infty$ as an upper bound.

matrix at the same program point (Example 1). Here, ★ means likely, and $\top$ unlikely. For instance, the above matrix contains ★ for the entry for $\mathtt{i} - \mathtt{a}$, which means that the main octagon analysis is likely to infer finite upper bounds of $\mathtt{i} - \mathtt{a}$. Notice that this predication is correct because the actual upper bound inferred by the octagon analysis is -1, as can be seen in Example 1.

Note that a single state of the pre-analysis domain denotes a set of states of the octagon analysis (in the same sense that a state of an ordinary static analysis denotes a set of concrete states). Formally, the domain of sets of octagons ($\bar{\wp}(\mathbb{O}) = \wp(\mathbb{O}) \setminus \{\emptyset\}$) is complete lattice ($\bar{\wp}(\mathbb{O}), \subseteq, \{\bot\}, \mathbb{O}, \sqcup_{\bar{\wp}}, \sqcap_{\bar{\wp}}$) that is downward-closed, i.e.,

$$\forall O \in \bar{\wp}(\mathbb{O}).o, o' \in \mathbb{O}.\ o \in O \ \wedge\ o' \sqsubseteq o \implies o' \in O$$

and the operations are defined pointwise:

$$\begin{aligned} X_1 \sqcup_{\bar{\wp}} X_2 &= \{x' \mid x_1 \in X_1 \ \wedge\ x_2 \in X_2 \ \wedge\ x' \sqsubseteq x_1 \sqcup x_2\} \\ X_1 \sqcap_{\bar{\wp}} X_2 &= \{x' \mid x_1 \in X_1 \ \wedge\ x_2 \in X_2 \ \wedge\ x' \sqsubseteq x_1 \sqcap x_2\} \end{aligned}$$

The domain of abstract states is the following complete lattice ($\mathbb{O}^\sharp, \sqsubseteq^\sharp, \bot^\sharp, \top^\sharp, \sqcup^\sharp, \sqcap^\sharp$) where

$$\mathbb{O}^\sharp = \{\bot^\sharp\} \cup \{\bigstar, \top_{\mathbb{V}}\}^{2n \times 2n}.$$

Abstract state $o^\sharp \in \mathbb{O}^\sharp$ is either $\bot^\sharp$ or an $2n \times 2n$ "abstract difference bound matrix" on $\{\bigstar, \top_{\mathbb{V}}\}$ (DBM$^\sharp$, for short). As in the octagon domain, the $o^\sharp_{xy}$ entry of the matrix stores the upper bound for $y - x$ (that is, $y - x \sqsubseteq o^\sharp_{xy}$). Each entry of the matrix is represented by ★ or $\top_{\mathbb{V}}$, and abstract state $o^\sharp \in \mathbb{O}^\sharp$ denotes a set of octagons that is characterized by the following Galois connection:

$$\bar{\wp}(\mathbb{O}) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \mathbb{O}^\sharp$$

$$\begin{array}{ll} \alpha(\{\bot\}) = \bot^\sharp, & \big(\alpha(O)\big)_{ij} = \left\{ \begin{array}{ll} \bigstar & \text{if } \bigsqcup O \neq \bot \ \wedge\ (\bigsqcup O)_{ij} \neq +\infty \\ \top_{\mathbb{V}} & \text{o.w.} \end{array} \right. \\ \gamma(\bot^\sharp) = \{\bot\}, & \gamma(o^\sharp) = \{o \mid \forall i, j.\ o^\sharp_{ij} = \bigstar \implies o_{ij} \neq +\infty\} \end{array}$$

As in the octagon domain, we assume that $\mathbb{O}^\sharp$ is the set of coherent DBM$^\sharp$: $\forall o^\sharp.i, j \in \mathit{Var}.o^\sharp_{ij} = o^\sharp_{\bar{j}\bar{i}}$. The domain operations (order, join, and meet) are defined pointwise using total order $\bigstar \sqsubseteq \top_{\mathbb{V}}$.

Lemma 1 (Galois connection (α, γ)). (α, γ) is a Galois connection:

$$\forall O \in \bar{\wp}(\mathbb{O}), o^\sharp \in \mathbb{O}^\sharp.\ \alpha(O) \sqsubseteq o^\sharp \iff O \subseteq \gamma(o^\sharp)$$

Proof
When $O = \{\bot\}$ or $o^\sharp = \bot^\sharp$, it is trivial by definition. It is enough to show the following:

$$\forall i, j \in \mathit{Var}.\ \big(\alpha(O)\big)_{ij} \sqsubseteq o^\sharp_{ij} \iff \{o_{ij} \mid o \in O\} \subseteq \{o'_{ij} \mid o' \in \gamma(o^\sharp)\}$$

Let X and Y be $\{o_{ij} \mid o \in O\}$ and $\{o'_{ij} \mid o' \in \gamma(o^\sharp)\}$, respectively. We consider the following cases:
$(\implies)$

- When $(\alpha(O))_{ij} = \bigstar$:
  For $(\alpha(O))_{ij} \sqsubseteq o^\sharp_{ij}$, $o^\sharp_{ij}$ is either ★ or $\top_{\mathbb{V}}$.
  - If $o^\sharp_{ij} = \bigstar$ then $+\infty \notin Y = \mathbb{Z}$ by definition. Because $(\alpha(O))_{ij} = \bigstar$, $+\infty \notin X$. Therefore, $X \subseteq Y$.
  - If $o^\sharp_{ij} = \top_{\mathbb{V}}$ then $Y = \mathbb{Z}_\infty$ by definition. Trivially, $X \subseteq Y$.
- When $(\alpha(O))_{ij} = \top_{\mathbb{V}}$:
  For $(\alpha(O))_{ij} \sqsubseteq o^\sharp_{ij}$, $o^\sharp_{ij}$ is $\top_{\mathbb{V}}$. Thus, $Y = \mathbb{Z}_\infty$ by definition. Trivially, $X \subseteq Y$.

DOI: 10.1002/spe

($\Longleftarrow$)

- When $+\infty \notin X$:
  For $X \subseteq Y$, Y is either $\mathbb{Z}$ or $\mathbb{Z}_\infty$.
  - If $Y = \mathbb{Z}$ then $o^\sharp_{ij} = \bigstar$ by definition. Because $+\infty \notin X$, $(\alpha(O))_{ij} \sqsubseteq o^\sharp_{ij}$.
  - If $Y = \mathbb{Z}_\infty$ then $o^\sharp_{ij} = \top_{\mathbb{V}}$ by definition. Trivially, $(\alpha(O))_{ij} \sqsubseteq o^\sharp_{ij}$.
- When $+\infty \in X$:
  For $X \subseteq Y$, Y is $\mathbb{Z}_\infty$. Thus, $o^\sharp_{ij} = \top_{\mathbb{V}}$. Trivially, $(\alpha(O))_{ij} \sqsubseteq o^\sharp_{ij}$.

□

We also define the notion of strong closure for DBM$^\sharp$. DBM$^\sharp$ $o^\sharp$ is strongly closed if and only if the following conditions are satisfied:

$$\begin{cases} \forall i,j,k & o^\sharp_{ij} \sqsubseteq o^\sharp_{ik} \sqcup o^\sharp_{kj} \\ \forall i,j & o^\sharp_{ij} \sqsubseteq o^\sharp_{i\bar{i}} \sqcup o^\sharp_{\bar{j}j} \\ \forall i & o^\sharp_{ii} = \bigstar \end{cases}$$

Note that, in the $\{\bigstar, \top_{\mathbb{V}}\}$ domain, the addition of two abstract values is equivalent to their join ($\sqcup$), i.e., $\forall v_1, v_2 \in \mathbb{V}.\ v_1 +_{\mathbb{V}} v_2 = v_1 \sqcup v_2$, and hence additions in the definition of strongly closed octagon are interpreted as joins in this definition. Intuitively, the first two conditions mean 1) if relationships $k - i$ and $j - k$ are bounded then relationship $j - i$ is also bounded, 2) if i and j are constants then relationship $j - i$ is bounded.

Abstract domain $\mathbb{D}^\sharp$ of the pre-analysis is defined using context-selector for full context-sensitivity $K_\infty = \lambda f.\mathbb{C}^*_c$ and $\mathbb{O}^\sharp$ as follows:

$$\mathbb{D}^\sharp = \mathbb{C}_{K_\infty} \rightarrow \mathbb{O}^\sharp$$

Abstract semantics $[\![c]\!]^\sharp : \mathbb{O}^\sharp \rightarrow \mathbb{O}^\sharp$ of our pre-analysis is defined in Figure 3. Whenever the pre-analysis computes $\bigstar$, say $x - y \sqsubseteq \bigstar$, it is guaranteed that the octagon analysis captures relation $x - y \leq c$ and $c \neq +\infty$. For instance, the pre-analysis computes $\bigstar$ in statements $[\![x := k]\!]^\sharp$ and $[\![x := y + k]\!]^\sharp$, because the octagon analysis can precisely analyze those cases. The semantics is not symmetric in case of $x := x + k$ where $k \geq 1$. Suppose there are two different variables x and y. The pre-analysis approximates the upper bound of $x - y$ to infinity (i.e. $x - y \sqsubseteq \top_{\mathbb{V}}$) as x increases by $x := x + k$. On the other hand, the upper bound of $y - x$ is always finite in the octagon analysis, hence $y - x \sqsubseteq \bigstar$. This aggressive abstraction on the upper bound of $x - y$ ($\top_{\mathbb{V}}$ rather than $\bigstar$) is because of a practical issue. Our pre-analysis is proven to be sound with respect to the least fixed point of the octagon analysis. However, the least fixed point cannot be computed in practice, and instead, a widening operator is used to compute an over-approximation of it. So we defined the abstract semantics of the pre-analysis to soundly handle common cases where the octagon analysis loses precision due to widening. Consider the following code:

```
x = y;
while(*){
  assert(x - y <= 0);
  x = x + 1;
}
```

In this example, the main octagon analysis with the standard widening yields $x - y \leq +\infty$ inside of the loop. To over-approximate the analysis result, we defined the semantics for $x := x + 1$ approximately:

$$\left([\![x := x + 1]\!]^\sharp(o^\sharp)\right)_{yx} = \top_{\mathbb{V}}$$

If we defined the semantics to produce $\bigstar$, the pre-analysis would select the query in the example program. We found that this alternative selects too many unprovable queries in practice.

$$
\begin{aligned}
\llbracket x := k \rrbracket^\sharp(o^\sharp) &= ({o^\sharp}')^\bullet \text{ where } {o^\sharp}'_{ij} = \begin{cases} \bigstar & (i = x, j = \bar{x} \text{ or } i = \bar{x}, j = x) \\ \big(\llbracket x :=? \rrbracket^\sharp(o^\sharp)\big)_{ij} & \textit{otherwise} \end{cases} \\
\llbracket x := y + k \rrbracket^\sharp(o^\sharp) &= ({o^\sharp}')^\bullet \text{ where } {o^\sharp}'_{ij} = \begin{cases} \bigstar & (i = x, j = y \text{ or } i = \bar{y}, j = \bar{x}) \\ \bigstar & (i = y, j = x \text{ or } i = \bar{x}, j = \bar{y}) \\ \big(\llbracket x :=? \rrbracket^\sharp(o^\sharp)\big)_{ij} & \textit{otherwise} \end{cases} \\
\big(\llbracket x := x + k \rrbracket^\sharp(o^\sharp)\big)_{ij} \text{ where } k \geq 1 &= \begin{cases} o^\sharp_{ij} & (i = x, j \notin \{x, \bar{x}\} \text{ or } i \notin \{x, \bar{x}\}, j = \bar{x}) \\ \top_{\mathbb{V}} & (i = \bar{x}, j \notin \{x, \bar{x}\} \text{ or } i \notin \{x, \bar{x}\}, j = x) \\ o^\sharp_{ij} & (i = x, j = \bar{x}) \\ \top_{\mathbb{V}} & (i = \bar{x}, j = x) \\ o^\sharp_{ij} & \textit{otherwise} \end{cases} \\
\big(\llbracket x := x + k \rrbracket^\sharp(o^\sharp)\big) \text{ where } k \leq 0 &= o^\sharp \\
\big(\llbracket x := -x \rrbracket^\sharp(o^\sharp)\big)_{ij} &= \begin{cases} o^\sharp_{\bar{i}j} & (i \in \{x, \bar{x}\}, j \notin \{x, \bar{x}\}) \\ o^\sharp_{i\bar{j}} & (i \notin \{x, \bar{x}\}, j \in \{x, \bar{x}\}) \\ o^\sharp_{\bar{i}\bar{j}} & (i, j \in \{x, \bar{x}\}) \\ o^\sharp_{ij} & \textit{otherwise} \end{cases} \\
\llbracket x := -y \rrbracket^\sharp &= \llbracket x := -x \rrbracket^\sharp \circ \llbracket x := y \rrbracket^\sharp \\
\llbracket x := -x + k \rrbracket^\sharp &= \llbracket x := x + k \rrbracket^\sharp \circ \llbracket x := -x \rrbracket^\sharp \\
\llbracket x := -y + k \rrbracket^\sharp &= \llbracket x := x + k \rrbracket^\sharp \circ \llbracket x := -y \rrbracket^\sharp \\
\big(\llbracket x :=? \rrbracket^\sharp(o^\sharp)\big)_{ij} &= \begin{cases} \top_{\mathbb{V}} & (i \in \{x, \bar{x}\}, j \notin \{x, \bar{x}\} \text{ or } i \notin \{x, \bar{x}\}, j \in \{x, \bar{x}\}) \\ \bigstar & (i = j = x \text{ or } i = j = \bar{x}) \\ o^\sharp_{ij} & \textit{otherwise} \end{cases} \\
\big(\llbracket x \leq k \rrbracket^\sharp(o^\sharp)\big) &= ({o^\sharp}')^\bullet \text{ where } {o^\sharp}'_{ij} = \begin{cases} \bigstar & (i = \bar{x}, j = x) \\ o^\sharp_{ij} & otherwise \end{cases} \\
\big(\llbracket -x \leq k \rrbracket^\sharp(o^\sharp)\big) &= ({o^\sharp}')^\bullet \text{ where } {o^\sharp}'_{ij} = \begin{cases} \bigstar & (i = x, j = \bar{x}) \\ o^\sharp_{ij} & otherwise \end{cases} \\
\big(\llbracket x - y \leq k \rrbracket^\sharp(o^\sharp)\big) &= ({o^\sharp}')^\bullet \text{ where } {o^\sharp}'_{ij} = \begin{cases} \bigstar & (i = y, j = x \text{ or } i = \bar{x}, j = \bar{y}) \\ o^\sharp_{ij} & otherwise \end{cases} \\
\big(\llbracket x + y \leq k \rrbracket^\sharp(o^\sharp)\big) &= ({o^\sharp}')^\bullet \text{ where } {o^\sharp}'_{ij} = \begin{cases} \bigstar & (i = \bar{y}, j = x \text{ or } i = \bar{x}, j = y) \\ o^\sharp_{ij} & otherwise \end{cases} \\
\big(\llbracket -x - y \leq k \rrbracket^\sharp(o^\sharp)\big) &= ({o^\sharp}')^\bullet \text{ where } {o^\sharp}'_{ij} = \begin{cases} \bigstar & (i = y, j = \bar{x} \text{ or } i = x, j = \bar{y}) \\ o^\sharp_{ij} & otherwise \end{cases} \\
\llbracket x \leq ? \rrbracket^\sharp(o^\sharp) &= o^\sharp
\end{aligned}
$$

Figure 3. Abstract semantics of primitive commands for the impact pre-analysis.

The pre-analysis result is defined as the least fixpoint of semantic function $F^\sharp : \mathbb{D}^\sharp \to \mathbb{D}^\sharp$:

$$F^\sharp(X) = X_0 \sqcup \mathit{Next}^\sharp(X)$$

where X_0 is the initial state and $Next^\sharp$ is defined in a fully context-sensitive and relational way with $[\![c]\!]^\sharp$ and $\to_{K_\infty}$.

$$Next^\sharp(X)(c,\kappa) = [\![c]\!]^\sharp\big(\bigsqcup\{X(c_0,\kappa_0) \mid (c_0,\kappa_0) \to_{K_\infty} (c,\kappa)\}\big).$$

The following lemma shows the relation between the pre-analysis and the main octagon analysis.

Lemma 2 (Soundness). Let $F_\infty : (\mathbb{C}_{K_\infty} \to \mathbb{O}) \to (\mathbb{C}_{K_\infty} \to \mathbb{O})$ be the semantic function for the fully context-sensitive and relational analysis, i.e. $K_\infty = \lambda f.\mathbb{C}_c^*$ and $\Pi = \{\mathsf{Var}\}^{\|}$:

$$F_\infty(X)(c,\kappa) = [\![c]\!]\Big(\bigsqcup\{X(c_0,\kappa_0) \mid (c_0,\kappa_0) \to_{K_\infty} (c,\kappa)\Big)\}$$

Then,

$$\forall c,\kappa.\ \mathsf{lfp}F_\infty(c,\kappa) \in \gamma((\mathsf{lfp}F^\sharp)(c,\kappa)) \tag{4}$$

Proof

To prove the desired relationship (4), we first define the "collecting octagon analysis", whose abstract domain is

$$\mathbb{C}_{K_\infty} \to \bar{\wp}(\mathbb{O})$$

and semantic function is

$$\bar{\wp}F_\infty : (\mathbb{C}_{K_\infty} \to \bar{\wp}(\mathbb{O})) \to (\mathbb{C}_{K_\infty} \to \bar{\wp}(\mathbb{O}))$$
$$\bar{\wp}F_\infty(X)(c,\kappa) = \bar{\wp}[\![c]\!]\Big(\bigsqcup_{\bar{\wp}}\{X(c_0,\kappa_0) \mid (c_0,\kappa_0) \to_{K_\infty} (c,\kappa)\}\Big)$$

where $\bar{\wp}[\![c]\!](X) = \{[\![c]\!](x) \mid x \in X\}$. Our pre-analysis is an abstraction of the collecting octagon analysis with the following Galois connection (α_0,γ_0):

$$\mathbb{C}_{K_\infty} \to \bar{\wp}(\mathbb{O}) \mathrel{\underset{\alpha_0}{\overset{\gamma_0}{\leftrightarrows}}} \mathbb{C}_{K_\infty} \to \mathbb{O}^\sharp$$

Note that (α_0,γ_0) is a Galois connection because (α,γ) is a Galois connection by Lemma 1. Then, by the abstract interpretation framework,

$$\forall c,\kappa.\ \mathsf{lfp}\bar{\wp}F_\infty(c,\kappa) \subseteq \gamma\big(\mathsf{lfp}F^\sharp(c,\kappa)\big) \tag{5}$$

when

$$\alpha \circ \bar{\wp}([\![c]\!]) \sqsubseteq [\![c]\!]^\sharp \circ \alpha$$

holds. The latter holds because, according to the definitions of $[\![c]\!]$ and $[\![c]\!]^\sharp$ presented in Figure 2 and 3, the pre-analysis semantics ($[\![c]\!]^\sharp$) is defined in a conservative way such that it gives ★ only when the octagon semantics ($[\![c]\!]$) always computes constant values.

We showed that (5) holds, but it still remains to prove (4). We prove (4) by showing the following:

$$\forall c,\kappa.\ \mathsf{lfp}F_\infty(c,\kappa) \in \mathsf{lfp}\bar{\wp}F_\infty(c,\kappa)$$

which is proved by the fixpoint induction [8]. Let P be the following inclusive assertion:

$$P(X,Y) \overset{def}{=} \forall c,\kappa.\ X(c,\kappa) \in Y(c,\kappa)$$

To prove is the following:

$$P(\mathsf{lfp}F_\infty, \mathsf{lfp}\bar{\wp}F_\infty) = \forall c,\kappa.\ \mathsf{lfp}F_\infty(c,\kappa) \in \mathsf{lfp}\bar{\wp}F_\infty(c,\kappa)$$

By the fixpoint induction, it is enough to show

$$\forall n \in \mathbb{N}.\ P(F_\infty^n(\bot), \bar{\wp}F_\infty^n(\bot)).$$

The proof proceeds as follows:

[‖] For brevity, we use the octagon domain ($\mathbb{O}$) rather than the packed octagon domain ($\mathbb{PO}(\Pi)$) because we consider only a single full pack ($\Pi = \{\mathsf{Var}\}$).

DOI: 10.1002/spe

- Base case $P(\bot, \bot)$:

$$\forall c, \kappa.\ \bot(c, \kappa) = \bot \in \{\bot\} = \bot(c, \kappa)$$

- Inductive case $P(F^k_\infty(\bot), \bar{\wp}F^k_\infty(\bot)) \implies P(F^{k+1}_\infty(\bot), \bar{\wp}F^{k+1}_\infty(\bot))$: To prove it, we need the following two properties on $\bar{\wp}\llbracket c \rrbracket$ and $\sqcup_{\bar{\wp}}$:
  - By definition of $\bar{\wp}\llbracket c \rrbracket$,

$$\forall o \in \mathbb{O},\ O \in \bar{\wp}(\mathbb{O}).\ o \in O \implies \llbracket c \rrbracket(o) \in \bar{\wp}\llbracket c \rrbracket(O). \qquad (6)$$

  - By definition of $\sqcup_{\bar{\wp}}$,

$$\forall o_1, o_2 \in \mathbb{O},\ O_1, O_2 \in \bar{\wp}(\mathbb{O}).\ o_1 \in O_1 \wedge o_2 \in O_2 \implies o_1 \sqcup o_2 \in O_1 \sqcup_{\bar{\wp}} O_2. \qquad (7)$$

  The inductive case is proved as follows

$$\begin{aligned}
\forall c, \kappa.\ F^{k+1}_\infty(\bot)(c, \kappa) &= F_\infty \circ F^k_\infty(\bot)(c, \kappa) \\
&= \llbracket c \rrbracket\big(\bigsqcup\{F^k_\infty(\bot)(c_0, \kappa_0) \mid (c_0, \kappa_0) \rightarrow_{K_\infty} (c, \kappa)\big) && \text{(by definition)} \\
&\in \bar{\wp}\llbracket c \rrbracket\big(\bigsqcup_{\bar{\wp}}\{\bar{\wp}F^k_\infty(c_0, \kappa_0) \mid (c_0, \kappa_0) \rightarrow_{K_\infty} (c, \kappa)\}\big) && \text{(by (6) and (7) and I.H.)} \\
&= \bar{\wp}F^{k+1}_\infty(\bot)(c, \kappa)
\end{aligned}$$

□

5.2. Use of the Pre-Analysis Results

By running the pre-analysis, we select a set of queries that are likely to be proven by the main analysis. We remind the reader that a set of queries ($\mathcal{Q} \subseteq \mathbb{C} \times \mathsf{Var} \times \mathsf{Var}$) is given in the program and query $(c, x, y) \in \mathcal{Q}$ represents a predicate on integer $y - x < 0$ at program point c. Using the pre-analysis results, we select queries $\mathcal{Q}^\sharp \subseteq \mathcal{Q}$ as follows (we simply ignore κ such that $(\mathsf{lfp}F^\sharp)(c, \kappa) = \bot^\sharp$):

$$\mathcal{Q}^\sharp = \{(c, x, y) \in \mathcal{Q} \mid \forall \kappa \in K_\infty.\big((\mathsf{lfp}F^\sharp)(c, \kappa)\big)_{xy} = \bigstar\}.$$

That is, if the pre-analysis gives the precise result ($\bigstar$) for variable pair (x, y) under every calling context $\kappa \in K_\infty$ at program point c, then we select query (c, x, y). The intuition is that if the pre-analysis computes $\bigstar$ for a query then the main octagon analysis will not lose too much precision and there is a chance to prove the query. Otherwise, if the pre-analysis computes $\top_{\mathbb{V}}$, we regard this as an indication that the main analysis is unlikely to prove the query, since the analysis result may involve $+\infty$.

The last step is to build context selector K and packing configuration Π that are helpful for the main analysis to prove the selected query q. K and Π consists of two parts, respectively:

$$\begin{aligned}
K(f) &= K_\bigstar(f) \cup K_\top(f) \\
\Pi &= \Pi_\bigstar \cup \Pi_\top
\end{aligned}$$

The essential parts are $K_\bigstar$ and $\Pi_\bigstar$ that include sensitivities that are required for the pre-analysis to generate $\bigstar$ value at query q; without the sensitivities in $K_\bigstar$ and $\Pi_\bigstar$, the pre-analysis cannot assign $\bigstar$ to the query q. To build $K_\bigstar$ and $\Pi_\bigstar$, we derive a set of pairs of sensitivities $\mathsf{S}_q = \{(\mathsf{K}_1, \pi_1), \cdots, (\mathsf{K}_n, \pi_n)\}$ from the pre-analysis result. Intuitively, K_i and π_i are collections of calling contexts and variable packs that should be considered in order for the pre-analysis to compute $\bigstar$ at query q along different contexts (and hence the main octagon analysis produces a finite constant). Then, $K_\bigstar$ and $\Pi_\bigstar$ for query q are defined as follows:

$$K_\bigstar(f) = \bigcup_{1 \le i \le n} \mathsf{K}_i(f) \qquad \Pi_\bigstar = \{\pi_i \mid 1 \le i \le n\}$$

DOI: 10.1002/spe

The other parts denote context-insensitivity and non-relational variables. $K_\top$ describes contexts that do not need to be differentiated:

$$K_\top(f) = \begin{cases} \{\epsilon\} & \textit{if } f \neq \mathsf{fid}(c_q) \\ \emptyset & \textit{otherwise} \end{cases}$$

where c_q is the program point where the selected query q is located. Likewise, $\Pi_\top$ contains all other variables that we decide not to track their relations. For each unchosen variable x, we construct singleton pack $\{x\}$:

$$\Pi_\top = \{\{x\} \mid x \in \mathsf{Var} \setminus \bigcup \Pi_\bigstar\}$$

We now explain how to find the set of sensitivities, $\mathsf{S}_q = \{(\mathsf{K}_1, \pi_1), \cdots, (\mathsf{K}_n, \pi_n)\}$, for selected query $q \in \mathcal{Q}^\sharp$ from the pre-analysis result. The idea is to observe semantic dependencies among variable relations along each context. The dependency represents contexts and relations that are required to generate the $\bigstar$ value for the selected query.

To do so, we first construct the value-flow graph $(\Theta, \hookrightarrow)$ that represents the semantic dependencies, where $(\Theta, \hookrightarrow)$ is defined as follows:

$$\Theta = \mathbb{C}_{K_\infty} \times \mathsf{Var} \times \mathsf{Var} \qquad (\hookrightarrow) \subseteq \Theta \times \Theta$$

To define edges, we define $U(c, x, y) \subseteq \mathsf{Var} \times \mathsf{Var}$, the set of pairs of variables whose relationships are required (during the pre-analysis) in order for variables x and y to have the $\bigstar$ value at program point c. Suppose command c is $x := y + k$ and $o^\sharp$ is the input state of $\llbracket c \rrbracket^\sharp$. According to the definition in Figure 3,

$$\llbracket x := y + k \rrbracket^\sharp(o^\sharp) = (o^{\sharp\prime})^\bullet \text{ where } {o^{\sharp}}'_{ij} = \begin{cases} \bigstar & (i = x, j = y \text{ or } i = \bar{y}, j = \bar{x}) \\ \bigstar & (i = y, j = x \text{ or } i = \bar{x}, j = \bar{y}) \\ \big(\llbracket x :=? \rrbracket^\sharp(o^\sharp)\big)_{ij} & \textit{otherwise} \end{cases}$$

Value $\big(\llbracket c \rrbracket^\sharp(o^\sharp)\big)_{xy}$ has no semantic dependencies on the input state, because it is always equivalent to $\bigstar$. However, $\big(\llbracket c \rrbracket^\sharp(o^\sharp)\big)_{xz}$, where $z \neq x$ and $z \neq y$, depends on $o^\sharp_{yz}$:

- When $o^\sharp_{yz} = \bigstar$:

$$\begin{array}{rcll} \big(\llbracket c \rrbracket^\sharp(o^\sharp)\big)_{xz} & \sqsubseteq_{\mathbb{V}} & \big(\llbracket c \rrbracket^\sharp(o^\sharp)\big)_{xy} \sqcup_{\mathbb{V}} \big(\llbracket c \rrbracket^\sharp(o^\sharp)\big)_{yz} & \text{(by definition of the strong closure)} \\ & = & \bigstar \sqcup_{\mathbb{V}} (o^{\sharp\prime})^\bullet_{yz} & \text{(by definition of } \llbracket c \rrbracket^\sharp\text{)} \\ & \sqsubseteq_{\mathbb{V}} & \bigstar \sqcup_{\mathbb{V}} (o^{\sharp\prime})_{yz} & \text{(by definition of the strong closure)} \\ & = & \bigstar \sqcup_{\mathbb{V}} \big(\llbracket x :=? \rrbracket^\sharp(o^\sharp)\big)_{yz} & \text{(by definition of the } o^{\sharp\prime}\text{)} \\ & = & \bigstar \sqcup_{\mathbb{V}} o^\sharp_{yz} & \text{(by definition of } \llbracket x :=? \rrbracket^\sharp\text{)} \\ & = & \bigstar \sqcup_{\mathbb{V}} \bigstar & ({o^\sharp}_{yz} = \bigstar) \\ & = & \bigstar & \end{array}$$

  That is, $\big(\llbracket c \rrbracket^\sharp(o^\sharp)\big)_{xz} = \bigstar$.
- When $o^\sharp_{yz} = \top_{\mathbb{V}}$:
  Suppose $\big(\llbracket c \rrbracket^\sharp(o^\sharp)\big)_{xz} = \bigstar$.

$$\begin{array}{rcll} \big(\llbracket c \rrbracket^\sharp(o^\sharp)\big)_{yz} & = & o^\sharp_{yz} & \text{(by definition of } \llbracket c \rrbracket^\sharp\text{)} \\ & = & \top_{\mathbb{V}} & \end{array}$$

$$\begin{array}{rcll} \big(\llbracket c \rrbracket^\sharp(o^\sharp)\big)_{yz} & \sqsubseteq_{\mathbb{V}} & \big(\llbracket c \rrbracket^\sharp(o^\sharp)\big)_{yx} \sqcup_{\mathbb{V}} \big(\llbracket c \rrbracket^\sharp(o^\sharp)\big)_{xz} & \text{(by definition of the strong closure)} \\ & = & \bigstar \sqcup_{\mathbb{V}} \big(\llbracket c \rrbracket^\sharp(o^\sharp)\big)_{xz} & \text{(by definition of } \llbracket c \rrbracket^\sharp\text{)} \\ & = & \bigstar \sqcup_{\mathbb{V}} \bigstar & \text{(by assumption)} \\ & = & \bigstar & \end{array}$$

  By contradiction, $\big(\llbracket c \rrbracket^\sharp(o^\sharp)\big)_{xz} = \top_{\mathbb{V}}$.

$$U(x := k, i, j) = \begin{cases} \{(j,j)\} & i = x, j \neq x \\ \{(i,i)\} & i \neq x, j = x \\ \{(i,j)\} & \text{otherwise} \end{cases}$$

$$U(x := y + k, i, j) = \begin{cases} \emptyset & i = x, j = y \text{ or } i = y, j = x \\ \{(y,j)\} & i = x, j \neq y \\ \{(x,j)\} & i = y, j \neq x \\ \{(i,x)\} & i \neq x, j = y \\ \{(i,y)\} & i \neq y, j = x \\ \{(i,j)\} & \text{otherwise} \end{cases}$$

$$U(x := x + k, i, j) \text{ where } k \geq 1 = \begin{cases} \{(i,j)\} & i = x, j \neq x \\ \emptyset & i \neq x, j = x \\ \{(i,j)\} & \text{otherwise} \end{cases}$$

$$U(x := x + k, i, j) \text{ where } k \leq 0 = \{(i,j)\}$$

$$U(x := -x, i, j) = \{(i,j)\}$$

$$U(x :=?, i, j) = \begin{cases} \emptyset & i = x \text{ or } j = x \\ \{(i,j)\} & \text{otherwise} \end{cases}$$

$$U(\pm x \leq k, i, j) = \begin{cases} \{(j,j)\} & i = x, j \neq x \\ \{(i,i)\} & i \neq x, j = x \\ \{(i,j)\} & \text{otherwise} \end{cases}$$

$$U(\pm x \pm y \leq k, i, j) = \begin{cases} \emptyset & i = x, j = y \text{ or } i = y, j = x \\ \{(y,j)\} & i = x, j \neq y \\ \{(x,j)\} & i = y, j \neq x \\ \{(i,x)\} & i \neq x, j = y \\ \{(i,y)\} & i \neq y, j = x \\ \{(i,j)\} & \text{otherwise} \end{cases}$$

$$U(x \leq ?, i, j) = \{(i,j)\}$$

Figure 4. The definition of U for each type of primitive commands. We omit the trivial case (i.e., $U(c, i, i) = \emptyset$).

In this example, $U(c, x, z) = \{(y, z)\}$. In general, U is defined as follows:

Definition 2 (U). Use set $U(c, x, y)$ for relation between variable x and y at program point c is defined as follows:

$$U(c, x, y) = \{(x_0, y_0) \mid \forall o^\sharp, i, j, i_0, j_0.\ o^\sharp \in \mathbb{O}^\sharp \ \wedge\ \left(\llbracket c \rrbracket^\sharp(o^\sharp)\right)_{ij} = \bigstar \ \wedge\ \left(\llbracket c \rrbracket^\sharp(o^\sharp \backslash_{i_0 j_0})\right)_{ij} = \top_{\mathbb{V}}\}$$

where $i \in \{x, \bar{x}\}$, $j \in \{y, \bar{y}\}$, $i_0 \in \{x_0, \bar{x_0}\}$, and $j_0 \in \{y_0, \bar{y_0}\}$. $o^\sharp \backslash_{i_0 j_0}$ is defined as follows:

$$\left(o^\sharp \backslash_{i_0 j_0}\right)_{ij} = \text{if } (i = i_0 \wedge j = j_0) \text{ then } \top_{\mathbb{V}} \text{ else } {o^\sharp}_{ij}$$

In this way, use-set $U(c, x, y)$ can be defined for each type of primitive commands in Figure 4. Now, we define value-flow relation ($\hookrightarrow$) that represents the semantic dependencies for variable relations according to the primitive commands:

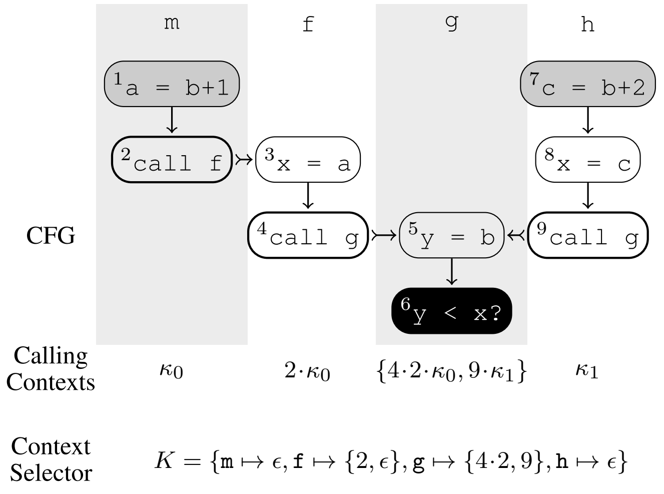


Figure 5. Example context selector. Gray and black nodes in CFG are source and query points, respectively.

Definition 3 ($\hookrightarrow$). The value-flow relation ($\hookrightarrow$) is defined as follows:

$$\begin{array}{l} ((c_0,\kappa_0),x_0,y_0) \hookrightarrow ((c,\kappa),x,y) \quad \text{iff} \\ \quad (\iota,\epsilon) \to^*_{K_\infty} (c_0,\kappa_0) \to_{K_\infty} (c,\kappa) \ \wedge \ (x_0,y_0) \in U(c,x,y) \end{array}$$

Example 4. Consider the control-flow graph in Figure 5. For node 2 and 8, the use sets are defined according to Definition 2: $U(2,\mathtt{a},\mathtt{b}) = \{(\mathtt{a},\mathtt{b})\}$ and $U(8,\mathtt{x},\mathtt{b}) = \{(\mathtt{c},\mathtt{b})\}$. Then the corresponding value-flow relations are defined as follows:

$$((1,\kappa_0),\mathtt{a},\mathtt{b}) \hookrightarrow ((2,\kappa_0),\mathtt{a},\mathtt{b})$$

$$((7,\kappa_1),\mathtt{c},\mathtt{b}) \hookrightarrow ((8,\kappa_1),\mathtt{x},\mathtt{b})$$

The next step is to extract program slices that include all semantic dependencies for each query. Program slices begin from *sources* (Φ) that do not have predecessors on the value-flow graph.

Definition 4 (Φ). Sources Φ are vertices in Θ where dependencies begin:

$$\Phi = \{((c_0,\kappa_0),x_0,y_0) \in \Theta \mid \nexists ((c,\kappa),x,y) \in \Theta.\ ((c,\kappa),x,y) \hookrightarrow ((c_0,\kappa_0),x_0,y_0)\}.$$

We compute the set of sources, $\Phi_{(c,x,y)}$, on which the query depends.

Definition 5 ($\Phi_{(c_q,x_q,y_q)}$). Sources on which the query (c_q,x_q,y_q) depends:

$$\Phi_{(c,x,y)} = \{((c_0,\kappa_0),x_0,y_0) \in \Phi \mid ((c_0,\kappa_0),x_0,y_0) \hookrightarrow^* ((c_q,\kappa),x_q,y_q)\}.$$

Example 5. Consider the control-flow graph in Figure 5. The black node (node 6) denotes the query point, i.e., $(c_q,x_q,y_q) = (6,\mathtt{x},\mathtt{y})$. The gray nodes (node 1 and node 7) represent sources on which the query depends, i.e., $\Phi_{(6,\mathtt{x},\mathtt{y})} = \{(1,\mathtt{a},\mathtt{b}),(7,\mathtt{c},\mathtt{b})\}$.

$\mathsf{Paths}_{(c,x,y)}$ denotes a set of all dependency paths for the query:

Definition 6 ($\mathsf{Paths}_{(c_q,x_q,y_q)}$). The set of all dependency paths for the query (c,x,y) is defined as follows:

$$\mathsf{Paths}_{(c_q,x_q,y_q)} = \{((c_0,\kappa_0),x_0,y_0) \hookrightarrow \cdots \hookrightarrow ((c,\kappa),x,y) \mid ((c_0,\kappa_0),x_0,y_0) \in \Phi_{(c_q,x_q,y_q)}\}.$$

Example 6. Suppose κ_0 and κ_1 are the initial context of function m and h in Figure 5, respectively. There exists two paths p_1 and p_2 for the query. p_1 is constructed as follows:

DOI: 10.1002/spe

- variable relation $b - a \leq \bigstar$ is generated at node 1 without any semantic dependencies and proceeds to node 2.
- in between x and b at node 3 depends on relation between b and a because $x = a$, and proceeds to node 4.
- relation between x and y at node 5 depends on relation between x and b because $y = b$, and proceeds to node 6.

p_2 is also constructed accordingly. That is, $\mathsf{Paths}_{(6,\mathtt{x},\mathtt{y})} = \{p_1, p_2\}$ where

$$\begin{aligned}p_1 =& ((1,\kappa_0),\mathtt{a},\mathtt{b}) \hookrightarrow ((2,\kappa_0),\mathtt{a},\mathtt{b}) \hookrightarrow ((3, 2\cdot\kappa_0),\mathtt{x},\mathtt{b}) \hookrightarrow ((4, 2\cdot\kappa_0),\mathtt{x},\mathtt{b})\\ &\hookrightarrow ((5, 4\cdot 2\cdot\kappa_0),\mathtt{x},\mathtt{y}) \hookrightarrow ((6, 4\cdot 2\cdot\kappa_0),\mathtt{x},\mathtt{y})\\ p_2 =& ((7,\kappa_1),\mathtt{c},\mathtt{b}) \hookrightarrow ((8,\kappa_1),\mathtt{x},\mathtt{b}) \hookrightarrow ((9,\kappa_1),\mathtt{x},\mathtt{b}) \hookrightarrow ((5, 9\cdot\kappa_1),\mathtt{x},\mathtt{y})\\ &\hookrightarrow ((6, 9\cdot\kappa_1),\mathtt{x},\mathtt{y})\end{aligned}$$

Finally, we derive context selectors and variable packs. Suppose we have a dependency path from source $((c_0,\kappa_0),x_0,y_0)$ to query $((c_n,\kappa_n),x_n,y_n)$.

$$((c_0,\kappa_0),x_0,y_0) \hookrightarrow ((c_1,\kappa_1),x_1,y_1) \hookrightarrow \cdots \hookrightarrow ((c_n,\kappa_n),x_n,y_n)$$

Note that $\kappa_0,\kappa_1,\cdots,\kappa_n$ are the calling contexts appearing in the pre-analysis which is fully context-sensitive and κ_0 is the context at the source. The partial contexts that will be used in the selective analysis are defined as the difference between κ_i and κ_0. If κ_0 is a suffix of κ_i, then the partial context for κ_i is defined as κ_i' where $\kappa_i = \kappa_i' \cdot \kappa_0$. The context κ_i' is formally defined as $\kappa_i \ominus \kappa_0 = \mathsf{suffix}(\kappa_i,\kappa_0)$, and $\mathsf{suffix}(\kappa_i,\kappa_0)$ is the longest common suffix of κ_i and κ_0. If κ_i is a suffix of κ_0 then we use ϵ as the partial context for κ_i. The variable pack for the path is constructed by collecting all variables on the path. Formally, the context selector and variable pack for each dependency path of the query is defined as follows:

Definition 7 ((K_p,π_p), Context Selector and Pack for Path p). Let p be a dependency path of query from a source $((c_0,\kappa_0),x_0,y_0)$ to a query. The context selector K_p and pack π_p for the path is defined as,

$$\begin{aligned}\mathsf{K}_p &= \lambda f.\{\kappa_i \ominus \kappa_0 \mid \mathsf{fid}(c_i) = f \ \wedge\ ((c_i,\kappa_i),\_) \in p\}\\ \pi_p &= \{z_i \mid (\_,z_0,z_1) \in p \ \wedge\ i \in \{0,1\}\}\end{aligned}$$

Example 7. For each path in Example 6, the context selectors and packs are defined as follows:

$$\begin{array}{ll}\mathsf{K}_{p_1} = \{\mathtt{m} \mapsto \epsilon, \mathtt{f} \mapsto \{2,\epsilon\}, \mathtt{g} \mapsto \{4{\cdot}2\}\} & \pi_{p_1} = \{a,b,x,y\}\\ \mathsf{K}_{p_2} = \{\mathtt{h} \mapsto \epsilon, \mathtt{g} \mapsto \{9\}\} & \pi_{p_2} = \{b,c,x,y\}\end{array}$$

Then, sensitivity S_q is derived from the context selectors and packs for each path:

Definition 8 (S_q, Sensitivity). Let q be a query. The sensitivity for context sensitive and relational analysis for our selective analysis is defined as follows:

$$\mathsf{S}_q = \{(\mathsf{K}_p,\pi_p) \mid p \in \mathsf{Paths}_q\}$$

Proposition 1 (Impact Realization). Let $(c,x,y) \in \mathcal{Q}^\sharp$ be a selected query. Let S be a set of sensitivities for the query derived from the impact pre-analysis. Let K and Π be the context selector and packing configuration defined with S. Let F be the semantic function of the selective context-sensitive and relational main analysis with K and Π. Then, the selective main analysis is at least as precise as the fully context-sensitive and relational pre-analysis for the selected query (c,x,y):

$$\begin{aligned}&\forall\kappa_0 \in K_\infty.\ \big(\mathsf{lfp}F^\sharp(c,\kappa_0)\big)_{xy} = \bigstar\\ &\quad\implies \forall(\mathsf{K},\pi) \in \mathsf{S}, \kappa \in \mathsf{K}(\mathsf{fid}(c)).\big(\mathsf{lfp}F(c,\kappa)(\pi)\big)_{xy} \neq +\infty.\end{aligned}$$

Proof
See Appendix. □

6. PRACTICAL IMPACT PRE-ANALYSIS

In this section, we further abstract the ideal impact pre-analysis to obtain practical efficiency. Though the pre-analysis in Section 5.1 aggressively abstracts the numerical domain, it is still very costly because of keeping full context-sensitivity and full relationships between variables. However, because the abstract domain of the pre-analysis is simple and finite, we can practically abstract the pre-analysis further by differentiating only the contexts that are relevant to the query. To this end, we first redefine the ideal pre-analysis using the summary-based context-sensitivity of which analysis results correspond with those of ∞-CFA [7]. The summary-based approach uses abstract states (rather than call-strings) as calling contexts of procedures so that the new pre-analysis abstracts the contexts by only considering a set of specific contexts (input states) related to the selected queries, instead of manipulating all of them. The query-driven input is generated by a backward pre-analysis that safely estimates the weakest pre-condition for query, at each function entry.

6.1. *Summary-based Context-Sensitivity*

First, we transform the pre-analysis in Section 5 to the summary-based one [13, 7]. We parameterize the summary-based approach so that the degree of context-sensitivity is determined by a (summary-based) context selector σ. Context selector σ assigns a set of calling contexts (i.e., abstract states) to each function, that is, σ maps procedures to sets of calling contexts:

$$\sigma \in \mathbb{F} \to \wp(\mathbb{O}^\sharp)$$

Let σ_∞ be the context selector for full context-sensitivity: σ_∞ assigns the entire set $\mathbb{O}^\sharp$ to each procedure, i.e., $\sigma_\infty = \lambda f.\mathbb{O}^\sharp$. Nodes of the control-flow graph are extended to context-enriched nodes $\mathbb{C}_\sigma \subseteq \mathbb{C} \times \mathbb{O}^\sharp$ with contexts in σ:

$$\mathbb{C}_\sigma = \{(c, o^\sharp) \mid c \in \mathbb{C} \ \wedge \ o^\sharp \in \sigma(\mathsf{fid}(c))\}.$$

The fully context-sensitive pre-analysis result is defined as the least fixpoint of semantic function $G^\sharp : (\mathbb{C}_{\sigma_\infty} \to \mathbb{O}^\sharp) \to (\mathbb{C}_{\sigma_\infty} \to \mathbb{O}^\sharp)$:

$$G^\sharp(X) = X_0 \sqcup \mathit{Next}^\sharp_{\sigma_\infty}(X)$$

where X_0 denotes the initial states and $\mathit{Next}^\sharp_{\sigma_\infty}$ is defined in a fully context-sensitive and relational way as follows:

$$\mathit{Next}^\sharp_{\sigma_\infty}(X)(c, o^\sharp) = \begin{cases} [\![c]\!]^\sharp\big(\bigsqcup\{X(c_0, o_0^\sharp) \mid c_0 \to c \ \wedge \ o^\sharp = o_0^\sharp\}\big) & (c \notin \mathbb{C}_e \cup \mathbb{C}_r) \\ \bigsqcup\{X(c_0, o_0^\sharp) \mid c_0 \to c \ \wedge \ o^\sharp = X(c_0, o_0^\sharp)\} & (c \in \mathbb{C}_e) \\ \bigsqcup\{X(c_0, o_0^\sharp) \mid c_0 \to c \ \wedge \ X(\mathsf{callof}(c), o^\sharp) = o_0^\sharp\} & (c \in \mathbb{C}_r) \end{cases}$$

At an intraprocedural node, which is neither an entry nor return node ($c \notin \mathbb{C}_e \cup \mathbb{C}_r$), the analysis result at the current point $(c, o^\sharp)$ is computed simply by applying the abstract semantics of commands ($[\![c]\!]^\sharp$) to the analysis results at preceding program points c_0 with the same context $o^\sharp$. When calling a procedure ($c \in \mathbb{C}_e$), the context of the procedure is initialized to the abstract state ($X(c_0, o_0^\sharp)$) at the call-site. When a procedure returns to a return node ($c \in \mathbb{C}_r$), the analysis resumes with the context that invoked the procedure at the corresponding call-site.

6.2. *Inferred Input States from a Backward Analysis*

The fully context-sensitive summary-based analysis is impractical, since the size of the domain $\mathbb{O}^\sharp$ is still huge. Thus, we further abstract the pre-analysis, so that it considers a particular set of calling contexts. Specifically, we consider only the input states of procedures that are relevant to proving queries. Given a query $q = (c_q, x_q, y_q)$, the corresponding input state of a procedure is obtained by analyzing the procedure backward starting from the program point containing q. Initially, abstract

DOI: 10.1002/spe

state $o^\sharp_q \in \mathbb{O}^\sharp$ at c_q is the weakest state in $\mathbb{O}^\sharp$ that satisfies q: (note that $q = (c_q, x_q, y_q)$)

$$oct_q)_{ij} = \begin{cases} \bigstar & (i = x_q, j = y_q \text{ or } i = \bar{y_q}, j = \bar{x_q}) \\ \top_{\mathbb{V}} & \textit{otherwise} \end{cases}$$

With $o^\sharp_q$, we perform the pre-analysis backwards, which leads to the corresponding input state. The backward pre-analysis is characterized by the following backward semantic function $B^\sharp_q \in (\mathbb{C} \to \mathbb{O}^\sharp) \to (\mathbb{C} \to \mathbb{O}^\sharp)$:

$$B^\sharp_q(X) \quad = \quad X_q \sqcap \mathit{Prev}(X)$$

where X_q is the weakest initial state that satisfies the given query q:

$$X_q = \top\{c_q \mapsto o^\sharp_q\}$$

Prev is a one-step backward semantic function:

$$\mathit{Prev}(X) = [\![c]\!]^\sharp_\uparrow\big(\bigsqcap\{X(c_0) \mid c \to c_0\}\big)$$

The backward semantics of primitive commands, $[\![c]\!]^\sharp_\uparrow$, is defined in Figure. 6. As we did for the forward semantics, the backward semantics of the pre-analysis is similarly obtained from the backward semantics of the octagon domain [9].[**] The pre-condition of procedure f is obtained by computing the greatest fixpoint of $B^\sharp_q$. That is, the set of abstract states, denoted $\sigma(f)$, relevant to proving the query q at the entry of procedure f is defined as follows:

$$\sigma(f) = \{\top^\sharp\} \cup \{(\mathsf{gfp} B^\sharp_q)(\mathsf{entryof}(f))\}.$$

Example 8. Consider the example program in Figure 1. The inferred input states for each procedure by the backward analysis are as follows:

$$\sigma(\texttt{inc}) = \{\top^\sharp\} \qquad \sigma(\texttt{f}) = \{\top^\sharp, o^\sharp\} \qquad \sigma(\texttt{g}) = \{\top^\sharp\}$$

where

$$o^\sharp_{ij} = \begin{cases} \bigstar & (i = x, j = y) \\ \top_{\mathbb{V}} & \textit{otherwise} \end{cases}$$

Since `inc` has no query, the only inferred input state is $\top$. For the query in `f`, the backward analysis infers an abstract state such that $y - x \sqsubseteq \bigstar$. The backward analysis cannot derive a promising input state for the query in g.

6.3. Summary-based Context-sensitivity modulo Query

Next, we abstract the impact pre-analysis using the inferred input states. Suppose σ denotes the context selector derived from the previous backward analysis. The abstraction is characterized by the following Galois connection:

$$(\mathbb{C}_{\sigma_\infty} \to \mathbb{O}^\sharp) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} (\mathbb{C}_\sigma \to \mathbb{O}^\sharp)$$

where

$$\alpha(X) = \lambda(c, o^\sharp).\bigsqcup\{X(c, o^\sharp_0) \mid o^\sharp_0 \sqsubseteq o^\sharp\}.$$

[**] In [9], the backward semantics is defined only for assignment statements. We extended the definition to conditional statements as well.

Then, we define the semantic function $G^\sharp_\sigma \in (\mathbb{C}_\sigma \rightarrow \mathbb{O}^\sharp) \rightarrow (\mathbb{C}_\sigma \rightarrow \mathbb{O}^\sharp)$ as follows:

$$G^\sharp_\sigma = X_0 \sqcup \mathit{Next}^\sharp_\sigma(X)$$

where X_0 is the initial state. $\mathit{Next}^\sharp_\sigma$ soundly approximates $\mathit{Next}^\sharp_{\sigma_\infty}$ by distinguishing only a particular set of inputs:

$$\mathit{Next}^\sharp_\sigma(X)(c, o^\sharp) =$$
$$\begin{cases} [\![c]\!]^\sharp\big(\bigsqcup\{X(c_0, o^\sharp_0) \mid c_0 \rightarrow c \ \wedge\ o^\sharp = o^\sharp_0\}\big) & (c \notin \mathbb{C}_e \cup \mathbb{C}_r) \\ \bigsqcup\{X(c_0, o^\sharp_0) \mid c_0 \rightarrow c \ \wedge\ o^\sharp \sqsupseteq X(c_0, o^\sharp_0)\} & (c \in \mathbb{C}_e) \\ \bigsqcup\{X(c_0, o^\sharp_0) \mid c_0 \rightarrow c \ \wedge\ o^\sharp_0 = \bigsqcap\{o \in \sigma(f) \mid X(\mathsf{callof}(c), o^\sharp) \sqsubseteq o\}\} & (c \in \mathbb{C}_r \ \wedge\ f = \mathsf{fid}(\mathsf{callof}(c))) \end{cases}$$

$\mathit{Next}^\sharp_\sigma$ normally computes the analysis results of internal nodes in a procedure. At call sites, the input state of the procedure does not lead to the calling context, but is over-approximated. Likewise, at return nodes, the most precise summary that subsumes the input state at the corresponding call-site is selected.

The following theorem shows that the query-driven, partially context-sensitive analysis is an over-approximation of the fully context-sensitive summary-based analysis, which implies that using the partial pre-analysis still guarantees the impact realization in the main analysis. Note that the result holds for every partial context selector σ.

Lemma 3 (Over-approximation). Let (c, x, y) be a query. Let $G^\sharp_{\sigma_\infty}$ be the semantic function for the summary-based fully context-sensitive analysis and $G^\sharp_\sigma$ be the query-driven partial context-sensitive analysis. Then,

$$o^\sharp \in \sigma(\mathsf{fid}(c)).\ \big(\mathsf{lfp} G^\sharp_\sigma(c, o^\sharp)\big)_{xy} = \bigstar \implies \big(\mathsf{lfp} G^\sharp_{\sigma_\infty}(c, o^\sharp)\big)_{xy} = \bigstar$$

Proof
Note that $\mathit{Next}^\sharp_\sigma$ is defined as a sound approximation of $\mathit{Next}^\sharp_{\sigma_\infty}$:

$$\alpha \circ \mathit{Next}^\sharp_{\sigma_\infty} \sqsubseteq \mathit{Next}^\sharp_\sigma \circ \alpha$$

By the fixpoint transfer theorem [3, 4], we have $\mathsf{lfp} G_{\sigma_\infty} \sqsubseteq \gamma(\mathsf{lfp} G_\sigma)$, which implies the claim of the lemma. □

Example 9. Suppose we analyze the example in Figure 1. Summary-based context selector σ is defined as in Example 8. At line 20, the calling context (input state) to `inc` is $\top^\sharp$ which is included in $\sigma(\texttt{inc})$. After analyzing `inc` with the calling context, the analysis yields abstract state $\mathtt{b} - \mathtt{a} \sqsubseteq \bigstar$ at line 21. Since the abstract state can be a possible calling context (after parameter binding) according to $\sigma(\texttt{f})$, the pre-analysis analyzes `f` with the input. Likewise, the pre-analysis analyzes the other procedure calls using σ. Finally, context selector K and variable packs Π are derived from the selected query as in Section 5.

$$\begin{aligned} K =& \{\texttt{inc} \mapsto \{20, 23, \epsilon\}, \texttt{f} \mapsto \{21, 24\}, g \mapsto \epsilon\} \\ \Pi =& \{\{a, b, n, x, y\}, \{c, d, n, x, y\}\} \end{aligned}$$

7. EXPERIMENTS

7.1. Setting

We implemented our selective context-sensitive and relational analysis on SPARROW [6, 10, 11, 12, 15], an industrial-strength buffer overrun analyzer for full C. The analyzer is flow- and field-sensitive and uses the interval domain [3] and octagon domain [9] as non-relational and relational numerical domains, respectively. Dynamically allocated heap locations are abstracted by their allocation sites. The analyzer performs a global analysis which analyzes the whole program starting from the `main` procedure. Our octagon analysis also uses the semantic modeling of Venet and Brat [16]. As reported, the octagon domain is not expressive enough to handle the byte-based representation of array offsets. Consider the following example:

```
1  int* p = (int*) malloc (sizeof(int) * s);
2  for (i = 0; i < s; i++)
3    *(p+i) = ...
```

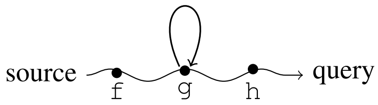
Assume that the size of an `int` variable is 4 bytes. Then, the exact loop invariant is as follows:

$$\begin{aligned}&p_{size} = 4 * s\\&p_{\mathit{offset}} = 0\\&0 \le i \le 4 * s - 4\end{aligned}$$

where p_{size} and p_{offset} are meta-variables that represent the size and offset of the array. The actual offset at line 3 is $p_{\mathit{offset}} + i$. Unfortunately, the octagon domain cannot express the general linear inequalities. Instead, we introduce another metavariable p_{stride} that represents the stride of the array. In the model, the actual size and offset in bytes are $p_{stride} * p_{size}$ and $p_{\mathit{offset}} + p_{stride} * i$, respectively. Now the loop invariant is given as follows:

$$\begin{aligned}&p_{size} = s\\&p_{\mathit{offset}} = 0\\&p_{stride} = 4\\&0 \le i \le s - 1\end{aligned}$$

We simply check buffer overflow queries based on the modeling. If a selected query is involved in a recursive call, we simply ignore the query and did not consider it in selecting context-sensitivity and variable packs. For example, suppose a query is judged promising and the slice of the program that contributes to the query looks as follows:



where `f`, `g`, and `h` are functions in the program, and `g` is a recursive function. Then, we exclude the query; it is analyzed by a context-insensitive interval analysis.

7.2. Performance

We evaluated our approach compared to the existing syntactic heuristic-based approach [9] and the context-insensitive selective relational analysis [12] for 8 GNU open-source projects. The syntactic heuristics relates variables that are located in the same syntactic blocks [9]. We limited the maximum pack size by 10 in the syntactic packing strategy. The existing selective approach [12] estimates the behavior of the context-*insensitive* octagon analysis and selects only necessary variables that will help to prove given queries. [††] Instead, our new pre-analysis derives both calling contexts and variable packs for the queries. The pre-analyses of both analyses use the same abstract domain and semantic function of primitive commands. Thanks to the simplicity of the domain ($\mathbb{O}^\sharp$), our approach is able to reduce time and memory consumption. We use a sparse representation for the matrices that have ★ and $\top_{\mathbb{V}}$ as elements. We compute the shortest-path closure [9] using Dijkstra's algorithm instead of the strong closure (Section 5.1). In addition, the sensitivity (Definition 8) is

[††] In the previous work [12], we manually inlined several functions to enable context-sensitive reasoning, but the analyzer is performed on the original programs to measure the net effect in these experiments.

Program	LOC	$\mathcal{Q}$	Syntactic		Selective Relational		Our Approach		Comparison	
			prvn	**time**	**prvn**	**time**	**prvn**	**time**	**prec**	**time**
spell-1.0	2,213	16	1	4.8	2	0.8 (2.0)	16	1.6 (1.5)	+15	-35.4%
httptunnel-3.3	6,174	28	16	26.0	16	4.0 (9.2)	26	5.5 (12.3)	+10	-31.5%
combine-0.3.3	11,472	23	0	142.7	2	20.4 (26.1)	23	18.1 (81.8)	+21	-30.0%
bc-1.06	13,093	10	2	247.1	3	39.9 (97.4)	9	34.8 (69.2)	+7	-57.9%
tar-1.17	20,258	17	6	1,043.2	6	98.9 (596.4)	17	191.0 (69.3)	+11	-75.0%
less-382	23,822	13	0	3,031.5	6	601.7 (1,729.0)	13	596.2 (126.2)	+13	-76.2%
bison-2.5	101,807	29	n/a	n/a	n/a	n/a	29	649.1 (1,080.8)	+29	n/a
bash-2.05a	105,174	70	n/a	n/a	51	481.3 (20,966.7)	70	634.6 (3,605.2)	+70	n/a
Total	284,013	206	25	4,495.3	86	1,246.9 (23,426.8)	201	2,130.9 (3,965.5)	+176	-73.1%

Table I. Performance of the selective context-sensitive and relational analysis compared to the syntactic heuristic-based packing strategy [9], and the context-insensitive selective relational analysis [12]. **LOC** reports lines of codes of the programs. $\mathcal{Q}$ denotes the number of buffer-overrun queries whose proofs require relational reasoning. **prvn** reports the number of queries that are proven by each analysis. Each X(Y) in **time** represent the time cost of main analysis (X) and pre-analysis cost (Y) for each selective analysis. **Comparison** shows additionally proven quires (**prec**) and the overhead (**time**) of our approach compared to the syntactic heuristic-based one.

naturally derived during the pre-analysis. All experiments were done on a Linux 2.6 system with a single core of Intel 3.07GHz box and 24GB of main memory.

Table I show the performance comparison between the three analyzers. Among all buffer accesses in the programs, we collected a set of buffer overrun queries ($\mathcal{Q}$) that require relational reasoning. We measured the precision by the number of proven queries. In addition, we compared the time cost of each main analysis and the pre-analysis cost of the two selective approaches.

The experimental results show that the proposed approach is precise and scalable. Among 206 queries, the new approach proved 201 (97.6%) queries while the syntactic heuristic and the selective relational analysis proved only 25 (12.1%) and 86 (41.7%), respectively. It shows that, in reality, most queries require relational reasoning beyond syntactic blocks, even across procedure boundaries; the syntactic approach and context-insensitive relational analyses are not precise enough. In addition, the proposed analysis scaled up to 100KLOC, while the previous one failed in one case (bison-2.5). Our approach reduced the time consumption by 73.1% on average for the small 6 programs compared to the syntactic heuristic-based analysis. In comparison to the previous selective relational analysis [12], the new approach increased the main analysis cost 18.8% but saved the pre-analysis cost by 87.6% on average; in total, the new approach saved 82.3%. Our analyzer missed 5 queries (2.4%) because of the over-approximation of the pre-analysis domain.

Our new pre-analysis effectively selects queries and infers necessary calling contexts and variable packs, subsuming the existing one, *i.e.* the proposed approach also selects queries that do not require context-sensitivity. In addition, the new approach is 5.9 times faster than the previous top-down one. The key factor for the analysis time reduction is the fixed set of input states of functions in the pre-analysis. Instead of considering all possible inputs during the analysis, our approach abstracts them into the prepared input states.

7.3. Discussion

This section reports additional information on the characteristics of context selectors and packing configurations found in experiments. We measured the number of different callstrings and (non-singleton) variable packs. In addition, we quantified the maximum and average sizes of them.

Table II shows the information for each benchmark program. The number of distinct call contexts $|K|$ and variable packs $|\Pi|$ depend on the number of selected queries estimated by the pre-analysis, since our approach generates a pair of a call string and a pack for each selected query. We observe that the derived call-strings for the octagon analysis are shorter than those for the interval analysis [12]. It means that tracking variable relationships between variables do not usually require long call chains. The numbers of packs are reasonably small and the sizes vary on the target programs, compared to those from the syntactic heuristics that blindly makes variable packs with a given fixed threshold.

	Context Selector			Packing Configuration		
Program	$\lvert K\rvert$	**max**	**avg**	$\lvert\Pi\rvert$	**max**	**avg**
spell-1.0	33	4	3.0	6	14	11.0
httptunnel-3.3	11	1	0.9	8	6	5.8
combine-0.3.3	12	1	0.9	13	7	3.7
bc-1.06	5	1	0.8	4	9	4.0
tar-1.17	3	1	0.7	7	8	3.9
less-382	7	1	0.9	8	18	6.3
bison-2.5	18	2	1.8	39	25	4.4
bash-2.05a	29	3	1.1	40	6	3.2

Table II. Characteristics of context selectors and packing configurations derived by our approach. $|K|$ and $|\Pi|$ report the number of different callstrings and non-singleton variable packs, respectively. **max** and **avg** report the maximum and average size of them.

We enumerate the representative cases that require context-sensitive and relational reasoning at the same time.

- xmalloc is a wrapper function of malloc that takes the size of array and returns a newly allocated array with the size. At line 4, the relationship between the size of nstr and len is established through the function call. The array access at line 6 is proven to be safe by tracking the relationships among pos, nstr and len in the loop. (excerpt from spell-1.0)

```
1  char* xmalloc(int size) { return malloc(size); }
2
3  void str_to_nstr(){
4    char* nstr = xmalloc(len+1);     // len > 0
5    for(pos = 0; pos < len; pos++)
6      nstr[pos] = ...
7  }
```

- read_until establishes the relationship between the parameter (data) and return value (len). At line 8, the relationship between data and n through the function call is used to prove the safety of the buffer access at the next line. (excerpt from httptunnel-3.3)

```
1  int read_until(char** data){
2    buf = malloc(len+1);             // len > 0
3    *data = buf;
4    return len;
5  }
6
7  int http_parse_request(){
8    n = real_until(&data);
9    data[n-1] = 0;
10 }
```

- The relationship between the size of string and i established at line 10 is passed through two function calls at line 12 and 6. The relationship is used to prove the safety at line 2. (excerpt from bash-2.05a)

```
1  char* get_history_word_specifier(char* spec, int* index){
2    spec[*index] = ...
3  }
4
5  int history_expand_internal(char* string, int i){
6    get_history_word_specifier(string, &i);
7  }
8
9  int history_expand(...){
10   string = malloc(l);             // l > 0
11   for (i = 0; i < l; i++)
12     history_expand_internal(string, i);
13 }
```

8. RELATED WORK

8.1. Selective X-sensitive Approach

This work extends the selective X-sensitive approach [12], and proposes an effective solution to achieve the global relational analysis. In the previous work, we provided a general approach and applied it to develop a selective context-sensitive interval analysis and selective relational analysis with the octagon domain. This work reuses the general idea but applies it to be selective both in context-sensitivity and relational analysis in a single analysis. Also, we provide a new design of the impact pre-analysis with the summary-based context-sensitivity.

8.2. Global Octagon Analysis

Existing global octagon analyzers [16, 2, 5] are also based on packing strategies, but do not predict the efficacy of packs. The syntactic packing heuristic [2] groups variables that linearly interact together within small syntactic blocks. The syntactic packing strategy easily misses necessary variables beyond the fixed code blocks. Other approaches construct packs by collecting semantically dependent variables [16, 5]. Their approaches may be a reasonable choice to analyze moderate-sized programs. However, they can generate useless packs or large packs containing useless variables in complicated programs because they do not care for the impact of packs. Furthermore, those approaches cannot construct compact packs along different calling contexts.

8.3. Scalable Relational Analysis

Localization [1] and sparse analysis [11] for relational analysis are orthogonal to our approach. Localization safely removes irrelevant parts of abstract memories when analyzing specific code blocks such as procedure bodies. Sparse analysis skips unnecessary propagation of semantic elements on control flows, in addition to localization. The approaches are applicable with any context selector and packing configuration so that our selective analysis complements those techniques.

9. CONCLUSION

We propose a selective conjunction of context-sensitivity and the octagon relational analysis toward cost-effective global analysis. The existing selective X-sensitive approach is limited to a single sensitivity and a naive combination is prohibitively impractical. To address the issue, we propose a selective analysis that applies context-sensitive and relational reasoning only when those sensitivities help prove the desired properties. Furthermore, we provide a practical pre-analysis that handles the huge sensitivities instead of naively combining the existing pre-analyses. For 8 open-source C programs, our approach proves 8 times more queries while reducing the time consumption of the analysis by 73.1% compared to the existing syntactic heuristic-based octagon relational analysis.

ACKNOWLEDGEMENTS

This work was partly supported by Samsung Research Funding Center of Samsung Electronics under Project Number SRFC-IT1502-07 and Institute for Information & communications Technology Promotion(IITP) grant funded by the Korea government(MSIP) (No.R0190-16-2011, Development of Vulnerability Discovery Technologies for IoT Software Security), (No.B0717-16-0098, Development of homomorphic encryption for DNA analysis and biometry authentication). This research was also supported by Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Science, ICT & Future Planning(NRF-2016R1C1B2014062) and BK21 Plus for Pioneers in Innovative Computing (Dept. of Computer Science and Engineering, SNU) funded by National Research Foundation of Korea(NRF) (21A20151113068).

REFERENCES

1. Eva Beckschulze, Stefan Kowalewski, and Jörg Brauer. Access-Based Localization for Octagons. *Electronic Notes in Theoretical Computer Science*, 287(C):29–40, November 2012.
2. Bruno Blanchet, Patrick Cousot, Radhia Cousot, Jérôme Feret, Laurent Mauborgne, Antoine Miné, David Monniaux, and Xavier Rival. A static analyzer for large safety-critical software. In *PLDI*, 2003.
3. Patrick Cousot and Radhia Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *POPL*, 1977.
4. Patrick Cousot and Radhia Cousot. Abstract interpretation frameworks. *J. Log. Comput.*, 1992.
5. Azadeh Farzan and Zachary Kincaid. Verification of parameterized concurrent programs by modular reasoning about data and control. In *POPL*, pages 297–308, 2012.
6. Yongin Jhee, Minsik Jin, Yungbum Jung, Deokhwan Kim, Soonho Kong, Heejong Lee, Hakjoo Oh, Daejun Park, and Kwangkeun Yi. Abstract interpretation + impure catalysts: Our Sparrow experience. Presentation at the Workshop of the 30 Years of Abstract Interpretation, San Francisco, `http://ropas.snu.ac.kr/~kwang/paper/30yai-08.pdf`, 2008.
7. Ravi Mangal, Mayur Naik, and Hongseok Yang. A correspondence between two approaches to interprocedural analysis in the presence of join. In *ESOP*, pages 513–533, 2014.
8. Zohar Manna, Stephen Ness, and Jean Vuillemin. Inductive methods for proving properties of programs. *SIGACT News*, (14):27–50, January 1972.
9. Antoine Miné. The octagon abstract domain. *Higher-Order and Symbolic Computation*, 19(1):31–100, 2006.
10. Hakjoo Oh, Lucas Brutschy, and Kwangkeun Yi. Access analysis-based tight localization of abstract memories. In *VMCAI*, 2011.
11. Hakjoo Oh, Kihong Heo, Wonchan Lee, Woosuk Lee, and Kwangkeun Yi. Design and implementation of sparse global analyses for C-like languages. In *PLDI*, 2012.
12. Hakjoo Oh, Wonchan Lee, Kihong Heo, Hongseok Yang, and Kwangkeun Yi. Selective context-sensitivity guided by impact pre-analysis. In *PLDI*, 2014.
13. Micha Sharir and Amir Pnueli. Two approaches to interprocedural data flow analysis. In *Program Flow Analysis: Theory and Applications*, pages 189–234. Prentice-Hall, Englewood Cliffs, NJ, 1981.
14. Olin Grigsby Shivers. *Control-flow analysis of higher-order languages -or- taming lambda*. PhD thesis, CMU, 1991.
15. Sparrow. http://ropas.snu.ac.kr/sparrow.
16. Arnaud Venet and Guillaume P. Brat. Precise and efficient static array bound checking for large embedded c programs. In *PLDI*, 2004.

A. PROOF

A.1. *Proof of Proposition 1*

Proof
We show that

$$\forall \kappa_0 \in K_\infty(\mathsf{fid}(c)).\ \big(\mathsf{lfp} F^\sharp(c,\kappa_0)\big)_{xy} = \bigstar \implies \forall (\mathsf{K},\pi) \in \mathsf{S}, \kappa \in \mathsf{K}(\mathsf{fid}(c)).\big(\mathsf{lfp} F(c,\kappa)(\pi)\big)_{xy} \neq +\infty.$$

It is proved by Lemma 4 and 7:

$$\begin{array}{lll} & \forall \kappa_0 \in K_\infty(\mathsf{fid}(c)).\big(\mathsf{lfp} F^\sharp(c,\kappa_0)\big)_{xy} = \bigstar & \\ \implies & \forall (\mathsf{K},\pi) \in \mathsf{S}, \kappa \in \mathsf{K}(\mathsf{fid}(c)).\big(\mathsf{lfp} F^\flat(c,\kappa)(\pi)\big)_{xy} = \bigstar & \text{(Lemma 4)} \\ \implies & \forall (\mathsf{K},\pi) \in \mathsf{S}, \kappa \in \mathsf{K}(\mathsf{fid}(c)).\big(\mathsf{lfp} F(c,\kappa)(\pi)\big)_{xy} \neq +\infty. & \text{(Lemma 7)} \end{array}$$

□

We define auxiliary semantic function $F^\flat$ that is a selective context-sensitive and relational pre-analysis with K and Π:

$$F^\flat(X) = \lambda X.X_0 \sqcup \mathit{Next}^\flat(X)$$

$\mathit{Next}^\flat$ is defined with $[\![c]\!]^\flat$ and $\rightarrow_K$:

$$\mathit{Next}^\flat(X)(c,\kappa) = [\![c]\!]^\flat\big(\bigsqcup\{X(c_0,\kappa_0) \mid (c_0,\kappa_0) \rightarrow_K (c,\kappa)\}\big)$$

$[\![c]\!]^\flat \in \mathbb{O}^\flat \rightarrow \mathbb{O}^\flat$ is defined on $\mathbb{O}^\flat = \Pi \rightarrow \mathbb{O}^\sharp$ as the same extension as $[\![c]\!]^\Pi$. We extends the definition of use set U and value-flow relation $(\hookrightarrow)$ to $U^\flat$ and $(\rightsquigarrow)$ as follows:.

$$U^\flat(\Pi)(c,x,y) = \{(x_0,y_0) \mid \pi \in \Pi \ \wedge\ o^\flat \in \mathbb{O}^\flat \ \wedge\ \big([\![c]\!]^\flat(o^\flat(\pi))\big)_{ij} = \bigstar \ \wedge\ \big([\![c]\!]^\flat(o^\flat(\pi)\backslash_{i_0 j_0})\big)_{ij} = \top_{\mathbb{V}}\}$$

$$\begin{array}{l} ((c_0,\kappa_0),x_0,y_0) \rightsquigarrow ((c,\kappa),x,y) \quad \textit{iff} \\ \quad (c_0,\kappa_0) \rightarrow_K (c,\kappa) \ \wedge\ (x_0,y_0) \in U^\flat(x,y) \end{array}$$

DOI: 10.1002/spe

Lemma 4 (Pre-analysis Coincidence). Let (c, x, y) be a query. Let $F^\sharp$ be the semantic function of the pre-analysis. Let S be a set of sensitivity for context-selector and packing configuration derived from the pre-analysis result. Let $F^\flat$ be the semantic function of the auxiliary pre-analysis.

$$\begin{aligned}&\forall \kappa_0 \in K_\infty(\mathsf{fid}(c)).\big(\mathsf{lfp}F^\sharp(c,\kappa_0)\big)_{xy} = \bigstar\\ &\implies \forall(\mathsf{K},\pi) \in \mathsf{S}, \kappa \in \mathsf{K}(\mathsf{fid}(c)).\big(\mathsf{lfp}F^\flat\big)(c,\kappa)(\pi)_{xy} = \bigstar\end{aligned}$$

Proof
It is sufficient to show that, in the value-flow graph, query (c, x, y) is reachable from source (c_0, x_0) under the full context-sensitivity if and only if (c_q, x_q) is reachable from (c_0, x_0) under the selective context-sensitivity with K:

$$\begin{aligned}&\forall((c_0,\kappa_0),x_0,y_0) \in \Phi_{(c,x,y)}, \exists \kappa,\kappa'.\\ &\quad((c_0,\kappa_0),x_0,y_0) \hookrightarrow^* ((c,\kappa),x,y) \iff ((c_0,\kappa_0'),x_0,y_0)\leadsto^*((c,\kappa'),x,y)\end{aligned}$$

- ($\Longrightarrow$) By Lemma 5.
- ($\Longleftarrow$) When $((c_0,\kappa_0),x_0,y_0) \in \Phi_{(c,x,y)}$, by the definition of $\Phi_{(c,x,y)}$. When $((c_0,\kappa_0),x_0,y_0) \notin \Phi_{(c,x,y)}$, it is proven by the previous work [12].

□

Lemma 5.

$$\begin{aligned}&\forall((c_0,\kappa_0),x_0,y_0) \in \Phi_{(c,x,y)}, \exists \kappa,\kappa'.\\ &\quad((c_0,\kappa_0),x_0,y_0) \hookrightarrow^* ((c,\kappa),x,y) \implies ((c_0,\kappa_0'),x_0,y_0)\leadsto^*((c,\kappa'),x,y)\end{aligned}$$

Proof
This simply amount to showing that:

$$\begin{aligned}&\forall 0 \le i < n.\ ((c_i,\kappa_i),x_i,y_i) \hookrightarrow ((c_{i+1},\kappa_{i+1}),x_{i+1},y_{i+1})\\ &\implies ((c_i,\kappa_i'),x_i,y_i) \leadsto ((c_{i+1},\kappa_{i+1}'),x_{i+1},y_{i+1})\end{aligned}$$

where $c_n = c$, $\kappa_n = \kappa$, $x_n = x$, and $y_n = y$. It is proved by the previous work [12],

$$\forall 0 \le i < n.\ (c_i,\kappa_i) \rightarrow_{K_\infty} (c_{i+1},\kappa_{i+1}) \implies (c_i,\kappa_i') \rightarrow_K (c_{i+1},\kappa_{i+1}')$$

and by Lemma 6,

$$\forall 0 \le i < n.\ (x_i,y_i) \in U(c_{i+1},x_{i+1},y_{i+1}) \implies (x_i,y_i) \in U^\flat(\Pi)(c_{i+1},x_{i+1},y_{i+1})$$

□

Lemma 6.

$$\forall 0 \le i < n.\ (x_i,y_i) \in U(c_{i+1},x_{i+1},y_{i+1}) \implies (x_i,y_i) \in U^\flat(\Pi)(c_{i+1},x_{i+1},y_{i+1})$$

Proof
We consider the following cases:

1. When c_{i+1} is $v := k$:
   - If $v = x_{i+1}$ then, $x_i = y_i = y_{i+1}$ by definition of U and $[\![v := k]\!]^\sharp$. By definition of Π, there exists $\pi \in \Pi$ such that $x_i, y_i, x_{i+1}, y_{i+1} \in \pi$. Then, $(x_i, y_i) \in U^\flat(\Pi)(c_{i+1}, x_{i+1}, y_{i+1})$ by definition of $U^\flat$.
   - if $v = y_{i+1}$ then, similar to the first case.
   - Otherwise, $x_i = x_{i+1}$ and $y_i = y_{i+1}$ by the definition of $[\![v := k]\!]^\sharp$. By definition of Π, there exists $\pi \in \Pi$ such that $x_i, y_i, x_{i+1}, y_{i+1} \in \pi$. Then, $(x_i, y_i) \in U^\flat(\Pi)(c_{i+1}, x_{i+1}, y_{i+1})$ by definition of $U^\flat$.
2. When c_{i+1} is $v := w + k$:
   - If $v = x_{i+1}$ and $w = y_{i+1}$ then, contradiction by the definition of U and $[\![v := w + k]\!]^\sharp$.
   - If $v = y_{i+1}$ and $w = x_{i+1}$ then, similar to the first case.

- If $v = x_{i+1}$ and $w \neq y_{i+1}$ then, $w = x_i$ and $y_{i+1} = y_i$ by definition of the $[\![v := w + k]\!]^\sharp$. By the definition of Π, there exists $\pi \in \Pi$ such that $x_i, y_i, x_{i+1}, y_{i+1} \in \pi$. Then, $(x_i, y_i) \in U^\flat(\Pi)(c_{i+1}, x_{i+1}, y_{i+1})$ by definition of $U^\flat$.
- If $v = y_{i+1}$ and $w \neq x_{i+1}$ then, similar to the third case.
- Otherwise, $x_i = x_{i+1}$ and $y_i = y_{i+1}$ by the definition of $[\![v := w + k]\!]^\sharp$. By the definition of Π, there exists $\pi \in \Pi$ such that $x_i, y_i, x_{i+1}, y_{i+1} \in \pi$. Then, $(x_i, y_i) \in U^\flat(\Pi)(c_{i+1}, x_{i+1}, y_{i+1})$ by definition of $U^\flat$.

3. When c_{i+1} is $v := v + k$:
   $x_i = x_{i+1}$ and $y_i = y_{i+1}$ by the definition of $[\![v := v + k]\!]^\sharp$ and U. By the definition of Π, there exists $\pi \in \Pi$ such that $x_i, y_i, x_{i+1}, y_{i+1} \in \pi$. Then, $(x_i, y_i) \in U^\flat(\Pi)(c_{i+1}, x_{i+1}, y_{i+1})$ by definition of $U^\flat$.
4. When c_{i+1} is $v :=?$:
   - If $v = x_{i+1}$ or $v = y_{i+1}$ then, contradiction by the definition of U and $[\![v :=?]\!]^\sharp$.
   - Otherwise, $x_i = x_{i+1}$ and $y_i = y_{i+1}$ by the definition of $[\![v :=?]\!]^\sharp$. By the definition of Π, there exists $\pi \in \Pi$ such that $x_i, y_i, x_{i+1}, y_{i+1} \in \pi$. Then, $(x_i, y_i) \in U^\flat(\Pi)(c_{i+1}, x_{i+1}, y_{i+1})$ by definition of $U^\flat$.
5. When c_{i+1} is a guard the proofs are simply subsumed by the one of assignment. For example, the semantics of $[\![x \leq k]\!]^\sharp$ is a sub-part of the one of $[\![x := k]\!]^\sharp$.

□

Lemma 7.

$$\forall (\mathsf{K}, \pi) \in \mathsf{S}, \kappa \in \mathsf{K}(\mathsf{fid}(c)). \big(\mathsf{lfp} F^\flat(c,\kappa)(\pi)\big)_{xy} = \bigstar \implies \big(\mathsf{lfp} F(c,\kappa)(\pi)\big)_{xy} \neq +\infty.$$

Proof

$$\begin{array}{llll} \big(\mathsf{lfp} F^\flat(c,\kappa)(\pi)\big)_{xy} = \bigstar & \implies & \big(\mathsf{lfp} F^\sharp(c)\big)_{xy} = \bigstar & (F^\flat \text{ is an over-approximation of } F^\sharp) \\ & \implies & \big(\mathsf{lfp} F_\infty(c)\big)_{xy} \neq \infty & (\text{by Lemma 2}) \\ & \implies & \big(\mathsf{lfp} F(c,\kappa)(\pi)\big)_{xy} \neq +\infty & (F \text{ is an over-approximation of } F_\infty) \end{array}$$

□

$$\begin{array}{rcl}
\big(\llbracket x := k \rrbracket^{\sharp}_{\uparrow}(o^{\sharp})\big)_{ij} & = & ({o^{\sharp}}')^{\bullet} \text{ where } {o^{\sharp}}'_{ij} = \begin{cases} \min({o^{\sharp}}^{\bullet}_{ij}, o^{\sharp}_{ix}, o^{\sharp}_{i\bar{x}}) & (i = \bar{j}, i \notin \{x, \bar{x}\}) \\ \top_{\mathbb{V}} & (i \in \{x, \bar{x}\} \text{ or } j \in \{x, \bar{x}\}) \\ {o^{\sharp}}^{\bullet}_{ij} & \textit{otherwise} \end{cases} \\
\big(\llbracket x := y + k \rrbracket^{\sharp}_{\uparrow}(o^{\sharp})\big)_{ij} & = & ({o^{\sharp}}')^{\bullet} \text{ where } {o^{\sharp}}'_{ij} = \begin{cases} \min({o^{\sharp}}^{\bullet}_{ij}, {o^{\sharp}}^{\bullet}_{xj}) & (i = y, j \notin \{x, \bar{x}, y, \bar{y}\}) \\ \min({o^{\sharp}}^{\bullet}_{ij}, {o^{\sharp}}^{\bullet}_{i\bar{x}}) & (i \notin \{x, \bar{x}, y, \bar{y}\}, j = \bar{y}) \\ \min({o^{\sharp}}^{\bullet}_{ij}, {o^{\sharp}}^{\bullet}_{\bar{x}j}) & (i = \bar{y}, j \notin \{x, \bar{x}, y, \bar{y}\}) \\ \min({o^{\sharp}}^{\bullet}_{ij}, {o^{\sharp}}^{\bullet}_{ix}) & (i \notin \{x, \bar{x}, y, \bar{y}\}, j = y) \\ \min({o^{\sharp}}^{\bullet}_{ij}, {o^{\sharp}}^{\bullet}_{\bar{x}x}) & (i = \bar{y}, j = y) \\ \min({o^{\sharp}}^{\bullet}_{ij}, {o^{\sharp}}^{\bullet}_{x\bar{x}}) & (i = y, j = \bar{y}) \\ \top_{\mathbb{V}} & (i \in \{x, \bar{x}\} \text{ or } j \in \{x, \bar{x}\}) \\ {o^{\sharp}}^{\bullet}_{ij} & \textit{otherwise} \end{cases} \\
\llbracket x := x + k \rrbracket^{\sharp}_{\uparrow} & = & \llbracket x := x - k \rrbracket^{\sharp} \\
\llbracket x := -x + k \rrbracket^{\sharp}_{\uparrow} & = & \llbracket x := -x + k \rrbracket^{\sharp} \\
\llbracket x := -y + k \rrbracket^{\sharp}_{\uparrow} & = & \llbracket x := y \rrbracket^{\sharp}_{\uparrow} \circ \llbracket x := -x + k \rrbracket^{\sharp}_{\uparrow} \\
\llbracket x :=? \rrbracket^{\sharp}_{\uparrow}(o^{\sharp}) & = & \text{if } o^{\sharp}_{ij} = \top_{\mathbb{V}} \text{ where } i \in \{x, \bar{x}\}, j \notin \{x, \bar{x}\} \text{ or } i \notin \{x, \bar{x}\}, j \in \{x, \bar{x}\} \text{ then} \\
 & & {o^{\sharp}}' \text{ where } {o^{\sharp}}'_{ij} = \begin{cases} \top_{\mathbb{V}} & (i \in \{x, \bar{x}\}, j \in \{x, \bar{x}\}) \\ o^{\sharp}_{ij} & \textit{otherwise} \end{cases} \\
 & & \text{else } \bot^{\sharp} \\
\llbracket x \leq k \rrbracket^{\sharp}_{\uparrow} & = & ({o^{\sharp}}')^{\bullet} \text{ where } {o^{\sharp}}'_{ij} = \begin{cases} \min({o^{\sharp}}^{\bullet}_{ij}, o^{\sharp}_{ix}) & (i = \bar{j}, i \notin \{x, \bar{x}\}) \\ \top_{\mathbb{V}} & (i = \bar{x} \text{ or } j = x) \\ {o^{\sharp}}^{\bullet}_{ij} & \textit{otherwise} \end{cases} \\
\llbracket -x \leq k \rrbracket^{\sharp} & = & ({o^{\sharp}}')^{\bullet} \text{ where } {o^{\sharp}}'_{ij} = \begin{cases} \min({o^{\sharp}}^{\bullet}_{ij}, o^{\sharp}_{i\bar{x}}) & (i = \bar{j}, i \notin \{x, \bar{x}\}) \\ \top_{\mathbb{V}} & (i = x \text{ or } j = \bar{x}) \\ {o^{\sharp}}^{\bullet}_{ij} & \textit{otherwise} \end{cases} \\
\big(\llbracket x - y \leq k \rrbracket^{\sharp}_{\uparrow}(o^{\sharp})\big)_{ij} & = & ({o^{\sharp}}')^{\bullet} \text{ where } {o^{\sharp}}'_{ij} = \begin{cases} \min({o^{\sharp}}^{\bullet}_{ij}, {o^{\sharp}}^{\bullet}_{ix}) & (i \notin \{x, \bar{x}, y, \bar{y}\}, j = y) \\ \min({o^{\sharp}}^{\bullet}_{ij}, {o^{\sharp}}^{\bullet}_{yj}) & (i = x, j \notin \{x, \bar{x}, y, \bar{y}\}) \\ \min({o^{\sharp}}^{\bullet}_{ij}, {o^{\sharp}}^{\bullet}_{\bar{x}x}) & (i = \bar{y}, j = y) \\ \min({o^{\sharp}}^{\bullet}_{ij}, {o^{\sharp}}^{\bullet}_{y\bar{y}}) & (i = x, j = \bar{x}) \\ \top_{\mathbb{V}} & (i = y, j = x \;\text{ or }\; i = \bar{x}, j = \bar{y}) \\ {o^{\sharp}}^{\bullet}_{ij} & \textit{otherwise} \end{cases} \\
\big(\llbracket x + y \leq k \rrbracket^{\sharp}_{\uparrow}(o^{\sharp})\big)_{ij} & = & ({o^{\sharp}}')^{\bullet} \text{ where } {o^{\sharp}}'_{ij} = \begin{cases} \min({o^{\sharp}}^{\bullet}_{ij}, {o^{\sharp}}^{\bullet}_{\bar{y}j}) & (i = x, j \notin \{x, \bar{x}, y, \bar{y}\}) \\ \min({o^{\sharp}}^{\bullet}_{ij}, {o^{\sharp}}^{\bullet}_{ix}) & (i \notin \{x, \bar{x}, y, \bar{y}\}, j = \bar{y}) \\ \min({o^{\sharp}}^{\bullet}_{ij}, {o^{\sharp}}^{\bullet}_{\bar{x}x}) & (i = y, j = \bar{y}) \\ \min({o^{\sharp}}^{\bullet}_{ij}, {o^{\sharp}}^{\bullet}_{\bar{y}y}) & (i = x, j = \bar{x}) \\ \top_{\mathbb{V}} & (i = \bar{y}, j = x \;\text{ or }\; i = \bar{x}, j = y) \\ {o^{\sharp}}^{\bullet}_{ij} & \textit{otherwise} \end{cases} \\
\big(\llbracket -x - y \leq k \rrbracket^{\sharp}_{\uparrow}(o^{\sharp})\big)_{ij} & = & ({o^{\sharp}}')^{\bullet} \text{ where } {o^{\sharp}}'_{ij} = \begin{cases} \min({o^{\sharp}}^{\bullet}_{ij}, {o^{\sharp}}^{\bullet}_{yj}) & (i = \bar{x}, j \notin \{x, \bar{x}, y, \bar{y}\}) \\ \min({o^{\sharp}}^{\bullet}_{ij}, {o^{\sharp}}^{\bullet}_{i\bar{x}}) & (i \notin \{x, \bar{x}, y, \bar{y}\}, j = y) \\ \min({o^{\sharp}}^{\bullet}_{ij}, {o^{\sharp}}^{\bullet}_{x\bar{x}}) & (i = \bar{y}, j = y) \\ \min({o^{\sharp}}^{\bullet}_{ij}, {o^{\sharp}}^{\bullet}_{y\bar{y}}) & (i = \bar{x}, j = x) \\ \top_{\mathbb{V}} & (i = y, j = \bar{x} \;\text{ or }\; i = x, j = \bar{y}) \\ {o^{\sharp}}^{\bullet}_{ij} & \textit{otherwise} \end{cases} \\
\llbracket x \leq ? \rrbracket^{\sharp}_{\uparrow}(o^{\sharp}) & = & o^{\sharp}
\end{array}$$

Figure 6. Backward abstract semantics of primitive commands for pre-analysis.

DOI: 10.1002/spe