Selective Context-Sensitivity Guided by Impact Pre-Analysis

Hakjoo Oh¹  Wonchan Lee¹  Kihong Heo¹  Hongseok Yang²  Kwangkeun Yi¹
Seoul National University¹, University of Oxford²

Abstract
We present a method for selectively applying context-sensitivity during interprocedural program analysis. Our method applies context-sensitivity only when and where doing so is likely to improve the precision that matters for resolving given queries. The idea is to use a pre-analysis to estimate the impact of context-sensitivity on the main analysis’s precision, and to use this information to find out when and where the main analysis should turn on or off its context-sensitivity. We formalize this approach and prove that the analysis always benefits from the pre-analysis-guided context-sensitivity. We implemented this selective method for an existing industrial-strength interval analyzer for full C. The method reduced the number of (false) alarms by 24.4%, while increasing the analysis cost by 27.8% on average.

The use of the selective method is not limited to context-sensitivity. We demonstrate this generality by following the same principle and developing a selective relational analysis.

Categories and Subject Descriptors F.3.2 [Semantics of Programming Languages]: Program Analysis

Keywords Static analysis, context-sensitive analysis

1. Introduction
Handling procedure calls in static analysis with a right balance between precision and cost is challenging. To precisely analyze procedure calls and returns, the analysis has to distinguish calls to the same procedure by their different calling contexts. However, a simple-minded, uniform context-sensitivity at all call sites easily makes the resulting analysis non cost-effective. For example, imagine a program analysis for proving the safety of array accesses that uses the k-callstring approach [16, 17] for abstracting calling contexts. The k-callstring approach distinguishes two calls to the same procedure whenever their k-most recent call sites are different. To make this context-sensitive analysis cost-effective, we need to tune the k values at the call sites in a way that we should increase the k value only where the increased precision contributes to the proof of array-access safety. If we simply use the same fixed k for all the call sites, the analysis would end up becoming either unnecessarily precise and costly, or not precise enough to prove the safety of many array accesses.

In this paper, we present a method for performing selective context-sensitive analysis, which applies the context-sensitivity only when and where doing so is likely to improve the precision that matters for the analysis’s ultimate goal. Our method consists of two steps. The first step is a pre-analysis that estimates the behavior of the main analysis under the full context-sensitivity (i.e. using callstrings). The pre-analysis focuses only on estimating the impact of context-sensitivity on the main analysis. Hence, it aggressively abstracts the other semantic aspects of the main analysis. The second step is the main analysis with selective context-sensitivity. This analysis uses the results of the pre-analysis, selects influential call sites for precision, and selectively applies context-sensitivity only to these call sites. Our method can be instantiated with a range of static analyses, and provides a guideline for designing impact pre-analyses for them, in particular, an efficient way of implementing those pre-analyses based on graph reachability.

One important feature of our method is that the pre-analysis-guided context-sensitivity pays off at the subsequent selective context-sensitive analysis. One way to see the subtlety of this impact realization is to note that the pre-analysis and the selective main analysis are incomparable in precision: the pre-analysis is more precise than the main analysis in terms of context sensitivity, but it is worse than the main analysis in tracking individual program statements. Despite this mismatch, our guidelines for designing an impact pre-analysis and the resulting selective context-sensitivity ensure that the selective context-sensitive main analysis is at least as precise as the fully context-sensitive pre-analysis, as far as given queries are concerned.

We have implemented our method on an existing industrial-strength interval analyzer for full C. The method led to the reduction of alarms from 6.6 to 48.3%, with average 24.4%, compared with the baseline context-insensitive analysis, while increasing the analysis cost from 9.4 to 50.5%, with average 27.8%.

The general principle behind the design and the use of our impact pre-analysis can be used for developing other types of selective analyses. We show its applicability by following the same principle and developing a selective relational analysis that keeps track of relationships between variables, only when tracking them are likely to help the main analysis answer given queries. In this case, the impact pre-analysis is fully relational while it aggressively abstracts other semantic aspects. We implemented this technique for the octagon analysis [11] and our experiments show that our selective octagon analysis achieves competitive cost-precision tradeoffs when applied to real-world benchmark programs.

Contributions
• We present a method for performing selective context-sensitive analysis that receives guidance from an impact pre-analysis.
• We show that the general idea behind our selective method is not limited to context-sensitivity. We present a selective relational analysis that is guided by an impact pre-analysis.
• We experimentally show the effectiveness of selective analyses designed according to our method, with real-world C programs.
2. Informal Description

We illustrate our approach using the interval domain and the program in Figure 1, which is adopted from make-3.76.1. Procedure xmalloc is a wrapper of the malloc function. It is called twice in procedure multi_glob, once with the argument size (line 4) and again with an input from the environment (line 6). The main routine of this program calls procedure f and g. Procedure multi_glob is called in f and g with different argument values.

The program contains two queries. The first query at line 5 asks whether p points to a buffer of size larger than 1. The other query at line 7 asks a similar question, but this time for the pointer variable q. Note that the first query always holds, but the second query is not necessarily true.

Context-insensitive analysis If we analyze the program using a context-insensitive interval analysis, we cannot prove the first query. Since the analysis is insensitive to calling contexts, it estimates the effect of xmalloc under all the possible inputs, and uses this same estimation as the result of every call. Note that an input to xmalloc at line 6 can be any integer, and the analysis concludes that xmalloc allocates a buffer of size in $[-\infty, +\infty]$.

Context-sensitive analysis A natural way to fix this precision issue is to increase the context-sensitivity. One popular approach is $k$-CFA analysis [16, 17]. It uses sequences of call sites up to length $k$ to distinguish calling contexts of a procedure, and analyzes the procedure separately for such distinguished calling contexts. For instance, 3-CFA analyzes the procedure xmalloc separately for each of the following calling contexts:

$$
\begin{array}{c}
4 \cdot 10 \cdot 14 \\
6 \cdot 10 \cdot 14
\end{array}
\begin{array}{c}
4 \cdot 10 \cdot 15 \\
6 \cdot 10 \cdot 15
\end{array}
\begin{array}{c}
4 \cdot 11 \cdot 16 \\
6 \cdot 11 \cdot 16
\end{array}
\begin{array}{c}
4 \cdot 11 \cdot 17 \\
6 \cdot 11 \cdot 17
\end{array}
(1)
$$

Here $a \cdot b \cdot c$ denotes a sequence of call sites $a$, $b$ and $c$ (we use the line numbers as call sites), with $a$ being the most recent call. Note that the 3-CFA analysis can prove the first query: the analysis analyzes the first four contexts separately and infers that a buffer of size greater than 1 gets allocated under these calling contexts.

Need of selective context-sensitivity However, using such a “uniform” context-sensitivity is not ideal. It is often too expensive to run such an analysis with high enough $k$, such as $k \geq 3$ that our example needs. More importantly, for many procedure calls, increasing context-sensitivity does not help—either it does not improve the analysis results of these calls, or the increased precision is not useful for answering queries. For instance, at the second query, for every $k \geq 0$, the $k$-CFA analysis concludes that $p$ points to a buffer of size $[-\infty, +\infty]$. Also, it is unnecessary to analyze $g$ separately for call sites 16 and 17, because those two calls have the same effect on the query.

Our selective context-sensitivity With our approach, an analysis can analyze procedures with only needed context-sensitivity. It analyzes a procedure separately for a calling context if doing so is likely to improve the precision of the analysis and reduce false alarms in its answers for given queries. For the example program, our analysis first predicts that increasing context-sensitivity is unlikely to help answer the second query (line 7) accurately, but is likely to do so for the first query (line 5). Next, the analysis finds out that we can bring the full benefit of context-sensitivity for the first query, by distinguishing only the following four types of calling contexts of xmalloc:

$$
\begin{array}{c}
4 \cdot 10 \cdot 14, \\
4 \cdot 10 \cdot 15, \\
4 \cdot 11, \\
all \ the \ other \ contexts
\end{array}
(2)
$$

Note that contexts $4 \cdot 11 \cdot 16$ and $4 \cdot 11 \cdot 17$ are merged into a single context $4 \cdot 11$. This merging happens because the analysis figures out that two callers of $g$ (line 16 and 17) do not provide any useful information for resolving the first query. Finally, the

Our selective context-sensitivity

![Figure 1. Example Program](https://example.com/figure1.png)

The second column of the table shows the results of the interval analysis with full context-sensitivity. Note that the pre-analysis in this case precisely estimates the impact of context-sensitivity: it identifies calling contexts (i.e., the first four contexts in the table) where the interval analysis accurately tracks the size of the allocated buffer in xmalloc under the full context-sensitivity. In general, our pre-analysis might lose precision and use $\top$ more often.
than in the ideal case. However, even when such approximation occurs, it does so only in a sound manner—if the pre-analysis computes $★$ for the size of a buffer, the interval analysis under full context-sensitivity is guaranteed to compute a non-negative interval.

**Use of pre-analysis results** Next, from the pre-analysis results, we select calling contexts that help improve the precision regarding given queries. We first identify queries whose expressions are assigned with $★$ in the pre-analysis run. In our example, the pre-analysis assigns $★$ to the expression $\text{sizeof}(p)$ in the first query. We regard this as a good indication that the interval analysis under full context-sensitivity is likely to estimate the value of $\text{sizeof}(p)$ accurately. Then, for each query that is judged promising, we consider the slice of the program that contributes to the query. We conclude that all the calls made in the slice should be tracked precisely. For example, if a slice for a query looks as follows:

![Diagram of program slice]

Then, we derive calling contexts $f, g, \{ f \cdot h \cdot g \}$, and $\{ i \cdot h \cdot f, i \cdot h \cdot g \}$ for procedure $f, g, h$, and $i$, respectively. However, if the slice involves a recursive call, we exclude the query since otherwise, we need infinitely many different calling contexts. In our example program, the slice for the first query includes all the execution paths from lines 11, 14, and 15 to line 5. Note that call-sites 16 and 17 are not included in the slice, and that all the calling contexts of $\text{xmalloc}$ in this slice are: $4 \cdot 10 \cdot 14, 4 \cdot 10 \cdot 15$, and $4 \cdot 11$. Our analysis decides to distinguish these contexts and their suffixes.

**Impact realization** Our method guarantees that the impact estimation under full context-sensitivity pays off at the subsequent selective context-sensitive analysis. That is, in our example program, the selective main analysis, which distinguishes only the contexts in (2), is guaranteed to assign a nonnegative interval to the expression $\text{sizeof}(p)$ at the first query. This guarantee holds because our selective context-sensitive analysis distinguishes all the calling contexts that matter for the selected queries (Section 5.2) and ensures that undistinguished contexts are isolated from the distinguished contexts (Section 4). For instance, although the call to $\text{xmalloc}$ at line 6 is analyzed in a context-insensitive way, our analysis ensures that $\text{xmalloc}$ in this case returns only to line 6, not to line 4.

**Application to relational analysis** Behind our approach lies a general principle for developing a static analysis that selectively uses precision-improving techniques, such as context-sensitivity and relational abstract domains. The principle is to develop an impact pre-analysis that finds out when and where the main static analysis under the full precision setting is likely to have an accurate result, and to choose an appropriate precision setting of the main analysis based on the results of this pre-analysis.

For instance, suppose that we want to develop a selective version of the octagon analysis, which tracks only some relationships between program variables that are likely to be tracked well by the octagon analysis and also to help the proofs of given queries. To achieve this goal, we design an impact pre-analysis that aims at finding when and where the original octagon analysis is likely to infer precise relationships between program variables. In Section 6, we describe this selective octagon analysis in detail.

### 3. Program Representation

We assume that a program $P$ is represented by a control flow graph $(\mathcal{C}, \to, F, t)$ where $\mathcal{C}$ is the finite set of nodes, $(\to) \subseteq \mathcal{C} \times \mathcal{C}$ denotes the control flow relation between nodes, $F$ is the set of procedure ids, and $\epsilon \in \mathcal{C}$ is the entry node of the main procedure.

The entry node $\epsilon$ does not have predecessors. A node $c \in \mathcal{C}$ in the program is one of the five types:

\[
\begin{align*}
\mathcal{C} &= \mathcal{C}_e \quad \text{(Entry Nodes)} \cup \mathcal{C}_c \quad \text{(Call Nodes)} \cup \mathcal{C}_r \quad \text{(Return Nodes)} \\
&\quad \cup \mathcal{C}_i \quad \text{(Internal Nodes)}
\end{align*}
\]

Each procedure $f \in F$ has one entry node and one exit node. Given a node $c \in \mathcal{C}$, $\text{fid}(c)$ denotes the procedure enclosing the node. Each call-site in the program is represented by a pair of call and return nodes. Given a return node $c \in \mathcal{C}_r$, we write $\text{callof}(c)$ for the corresponding call node. We assume for simplicity that there are no indirect function calls such as calls via function pointers.

We associate a primitive command with each node $c$ of our control flow graph, and denote it by $\text{cmd}(c)$. For brevity, we consider simple primitive commands specified by the following grammar:

\[
\text{cmd} \to \text{skip } | x := e
\]

where $e$ is an arithmetic expression: $e \to \mathcal{C} \ni x \mid e + e \mid e - e$. We denote the set of all program variables by $\text{Var}$.

For simplicity, we handle parameter passing and return values of procedures via simple syntactic encoding. Recall that we represent a 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 the actual parameter $e$ is assigned to the formal parameter $p$. For return values, we assume that each procedure $f$ has a variable $r_f$ and the return value is assigned to $r_f$; that is, we represent return statement $\text{return } e$ 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 there are no global variables in the program, all parameters and local variables of procedures are distinct, and there are no recursive procedures.

### 4. Selective Context-Sensitive Analysis with Context-Sensitivity Parameter $K$

We consider selective context-sensitive analyses specified by the following data: (1) a domain $\mathcal{S}$ of abstract states, which forms a complete lattice structure $(\mathcal{S}, \subseteq, ⊔, ⊓, ⊖, ⊠, *)$; (2) an initial abstract state $σ_I \in \mathcal{S}$ at the entry of the main procedure; (3) a monotone abstract semantics of primitive commands $\text{cmd} : \mathcal{S} \to \mathcal{S}$; (4) a context selector $K$ that maps procedures to sets of calling contexts (sequences of call nodes):

\[
K \in F \to \wp(\mathcal{C}).
\]

For each procedure $f$, the set $K(f)$ specifies calling contexts that the analysis should differentiate while analyzing the procedure. We sometimes abuse the notation and denote by $K$ the entire set of calling contexts in $K$: we write $κ \in K$ for $κ \in \bigcup_{f \in F} K(f)$.

With the above data, we design a selective context-sensitive analysis as follows. First, we differentiate nodes with contexts in $K$, and define a set $C_K \subseteq \mathcal{C} \times C_K$ of context-enriched nodes:

\[
C_K = \{(c, κ) \mid c \in \mathcal{C} \land κ \in K(\text{fid}(c))\}.
\]

The control flow relation $(\cdot \to) \subseteq \mathcal{C} \times \mathcal{C}$ is extended to $\to_K$ on $C_K$:

**Definition 1** $(\cdot \to_K) \subseteq C_K \times C_K$ is the context-enriched control flow relation:

\[
\begin{align*}
(c, κ) &\to_K (c', κ') \iff \\
&\begin{cases}
\quad c \to c' \land κ' := κ \\
\quad c \to c' \land κ' := κ \land κ \\
\quad c \to c' \land κ = \text{callof}(c') :: K \land κ' \\
\quad (c' \notin \mathcal{C}_e \cup \mathcal{C}_r) \\
\quad (c \in \mathcal{C}_e \land c' \in \mathcal{C}_e) \\
\quad (c \in \mathcal{C}_i \land c' \in \mathcal{C}_i)
\end{cases}
\end{align*}
\]
where \( :K \in \mathbb{C}_c \times \mathbb{C}_c \to \mathbb{C}_c \) updates contexts according to \( K \):
\[
\varepsilon :K \kappa = \begin{cases} \varepsilon \cdot \kappa & (c \cdot \kappa \in K) \\ \varepsilon & \text{otherwise} \end{cases}
\]
where \( \varepsilon \) is the empty call sequence.

In our analysis, \( \varepsilon \) is used to represent all the other contexts not included in \( K \), and we assume that \( K \) includes \( \varepsilon \) if it is necessary. For instance, consider a program where \( J \) has three different calling contexts \( K_1, K_2, \) and \( K_3 \). When the analysis differentiates \( K_1 \) only, undistinguished contexts \( K_2 \) and \( K_3 \) are represented by \( \varepsilon \). Thus, \( K(f) = \{ K_1, \varepsilon \} \). Note that our analysis isolates undistinguished contexts from distinguished ones: \( \varepsilon \) means only \( K_2 \) or \( K_3 \), not \( K_1 \).

**Example 1.** The analysis is context-insensitive when \( K = \lambda f.\{ \varepsilon \} \) and fully context-sensitive when \( K = \lambda f.\mathbb{C}_c \). Our selective context-sensitive analysis in Section 2 uses the following context selector \( \{ f, f, \text{multi.glob} \to \{ 10-14, 10-15, 11 \}, \text{malloc} \to \{ 4 \cdot 10-15, 4-15, 4-11, \varepsilon \} \} \).

Next, we define the abstract domain \( \mathbb{D} \) of the analysis:
\[
\mathbb{D} = (\mathbb{C}_K \to \mathbb{S})
\]

The analysis keeps multiple abstract states at each program node \( c \), one for each context \( \kappa \in K (\text{fid}(c)) \). The abstract transfer function \( F \) of the analysis works on \( \mathbb{C}_K \), and it is defined as follows:

\[
F(X)(c, \kappa) = \bigcup_{(e_0, s_0) \to (c, \kappa)} X(e_0, s_0)
\]

The static analysis computes an abstract element \( X \in \mathbb{D} \) satisfying the following condition:
\[
s_1 \subseteq X(t, \varepsilon) \land \forall (c, \kappa) \in \mathbb{C}_K . F(X)(c, \kappa) \subseteq X(c, \kappa)
\]

In general, many \( X \) can satisfy the condition in (5). Some analyses compute the least \( X \) satisfying (5). Other analyses use a widening operator \( [1] \).

**Example 2 (Interval Analysis).** The interval analysis is a standard example that uses a widening operator. Let 1 be the domain of intervals: \( I = \{ [l, u] \mid l, u \in \mathbb{Z} \cup \{-\infty, +\infty\} \land l \leq u \} \). Using this domain, we specify the rest of the analysis:

1. The abstract states are \( \bot \) or functions from program variables to their interval values: \( \mathbb{S} = \{ \bot \} \cup \{ \mathbb{V} \to I \} \).
2. The initial abstract state is: \( s_1(x) = [-\infty, +\infty] \).
3. The abstract semantics of primitive commands is:
   
   \[
   \begin{align*}
   \text{skip}(s) & = s, \quad x := e(s) = \begin{cases} s[x \mapsto \text{eval}(e)](s) & (s \neq \bot) \\ \bot & (s = \bot) \end{cases} \\
   \text{if}(a, \ell, \ell', \bot) & = \begin{cases} \text{ite}(l < t, \ell, \ell', \bot, 0, 0) & (t) \\ \text{ite}(u > u, +\infty, u) & \end{cases}
   \end{align*}
   \]
   where \( \text{eval} \) is the abstract evaluation of the expression \( e \):

   \[
   \begin{align*}
   [n](s) & = [n, r], \quad [e_1 + e_2](s) = [e_1](s) + [e_2](s) \\
   [x](s) & = s(x), \quad [e_1 - e_2](s) = [e_1](s) - [e_2](s)
   \end{align*}
   \]

4. The last component of the analysis is a widening operator, which is defined as a pointwise lifting of the following widening operators \( \mathbb{V} \): \( I \times [l, u] \to \mathbb{D} \) for intervals:
   \[
   [l, u] \mathbb{V} [l', u'] = \begin{cases} \text{ite}(l' \leq l, \ell, \ell'(l' < 0, -\infty, 0), l), & \text{ite}(u' > u, +\infty, u) \end{cases}
   \]
   where \( \text{ite} \) evaluates to \( a \) if \( p \) is true and \( b \) otherwise. The above widening operator uses 0 as a threshold, which is useful when proving the absence of buffer overruns.

**Queries** Queries are triples in \( \mathbb{Q} \subseteq \mathbb{C} \times \mathbb{S} \times \mathbb{V} \), and they are given as input to our static analysis. A query \( (c, s, x) \) represents an assertion that every reachable concrete state at node \( c \) is over-approximated by the abstract state \( s \). The last component \( x \) describes that the query is concerned with the value of the variable \( x \). For instance, in the interval analysis, a typical query is
\[
(c, \mathbb{V}f, (y = x) \land [0, \infty] \text{ else } \top, x)
\]
for some variable \( x \). It asserts that at program node \( c \), the variable \( x \) should always have a non-negative value. Proving the queries or identifying those that are likely to be violated is the goal of the analysis.

5. **Impact Pre-Analysis for Finding \( K \)**

Suppose that we would like to develop a selective context-sensitive analysis in Section 4 for a given program and given queries, using one of the existing abstract domains specified by the following data:

\[
(\mathbb{S}, s_I \in \mathbb{S}, [\bot] : \mathbb{D} \to \mathbb{S})
\]

To achieve our aim, we need to construct \( K \) specifying context-sensitivity for the given program and queries. Once this construction is done, the rest is standard. The analysis can analyze the program under partial context-sensitivity, using the induced abstract domain \( \mathbb{D} \) and transfer function \( F : \mathbb{D} \to \mathbb{D} \) for this program in (5) and (4). We assume that the analysis employs the fixpoint algorithm based on widening operation \( \mathbb{V} : \mathbb{D} \times \mathbb{D} \to \mathbb{D} \).

How should we automatically choose an effective \( K \) that balances the precision and cost of the induced interprocedural analysis? In this section, we give an answer to this question. In Section 5.1, we present an impact pre-analysis, which estimates the behavior of the main analysis \( (\mathbb{S}, s_I, [\bot]) \) under full context-sensitivity. In Section 5.2, we describe how to use the results of this pre-analysis for constructing an effective context selector \( K \).

Throughout the section, we fix our main analysis to \( (\mathbb{S}, s_I, [\bot]) \).

### 5.1 Designing an Impact Pre-Analysis

An impact pre-analysis for context sensitivity aims at estimating the main analysis \( (\mathbb{S}, s_I, [\bot]) \) under full context-sensitivity. It is specified by the following data:

\[
(\mathbb{S}^\sharp, s_I^\sharp \in \mathbb{S}^\sharp, [-\ell] : \mathbb{D}^\sharp \to \mathbb{S}^\sharp, K^\infty)
\]

This specification and the way that the data are used in our pre-analysis are fairly standard. \( \mathbb{S}^\sharp \) and \( [\bot] \) are, respectively, the domain of abstract states and the abstract semantics of \( \text{cmd} \) used by the pre-analysis, and \( s_I^\sharp \) is an initial state. \( K^\infty = \lambda f.\mathbb{C}_c \) is the context selector for full context-sensitivity. The pre-analysis uses the abstract domain \( \mathbb{D}^\sharp = \mathbb{C}_K^\infty \to \mathbb{S}^\sharp \) and the following transfer function \( F^\sharp : \mathbb{D}^\sharp \to \mathbb{D}^\sharp \) for the given program:
\[
F^\sharp(X)(c, \kappa) = [\mathbb{cmd}(X)]^\sharp(\bigcup_{(e_0, s_0) \to (c, \kappa)} X(e_0, s_0))
\]

It computes the least \( X \) satisfying
\[
s_I^\sharp \subseteq X(t, \varepsilon) \land \forall (c, \kappa) \in \mathbb{C}_K . F^\sharp(X)(c, \kappa) \subseteq X(c, \kappa)
\]

What is less standard is the soundness and efficiency conditions for our pre-analysis, which provides a guideline on the design of these pre-analyses. Let us discuss these conditions separately.

**Soundness condition** Intuitively, our soundness condition says that all the components of the pre-analysis have to over-approximate the corresponding ones of the main analysis.\(^1\) This is identical to the standard soundness requirement of a static program analysis.

---

\(^1\)We design a pre-analysis as an over-approximation of the main analysis, because an under-approximating pre-analysis would be too optimistic in context selection and the resulting selective main analysis is hardly cost-effective.
except that the condition is stated not over the concrete semantics of a given program, but over the main analysis. The condition has the following four requirements:

1. There should be a concretization function $\gamma : S^2 \rightarrow \mathcal{P}(S)$. This function formalizes the fact that an abstract state of the main analysis means a set of abstract states of the main analysis.

2. The initial abstract state of the pre-analysis has to overapproximate the initial state of the main analysis, i.e., $s_1 \in \gamma(s_1^p)$. The next condition is for the efficiency of $\gamma$.

3. The abstract semantics of commands in the pre-analysis should be sound with respect to that of the main analysis:  
   \[
   \forall s \in S, s^p \in S^2, s \in \gamma(s^p) \implies \text{cmd}(s) \in \gamma([\text{cmd}]^p(s^2)).
   \]

4. The join operation of the pre-analysis's abstract domain over-approximates the widening operation of the main analysis: for all $X, Y \in \mathcal{D}$ and $X^2, Y^2 \in \mathcal{D}^2$,
   \[
   X \in \gamma(X^2) \land Y \in \gamma(Y^2) \implies X \uplus Y \in \gamma(X^2 \sqcup Y^2).
   \]

The purpose of our condition is that the impact pre-analysis over-approximates the fully-context-sensitive main analysis:

**Lemma 1.** Let $M \in \mathcal{D}$ be the main analysis result, i.e., a solution of (5) under full context-sensitivity ($K = K_\infty$). Let $P \in \mathcal{D}^2$ be the pre-analysis result, i.e., the least solution of (6). Then, $\forall c \in C, \kappa \in C^\kappa, M(c, \kappa) \in \gamma(P(c, \kappa))$.

**Efficiency condition** The next condition is for the efficiency of our pre-analysis. It consists of two requirements, and ensures that the pre-analysis can be computed using efficient algorithms:

1. The abstract states are $\bot$ or functions from program variables to abstract values: $S^2 = \{\bot\} \cup (\text{Var} \rightarrow \mathcal{V})$, where $\mathcal{V}$ is a finite complete lattice $(\mathcal{V}, \sqsubseteq, \bot, \top, \land, \lor)$. An initial abstract state is $s_1^p = \lambda x. \top_v$.

2. The abstract semantics of primitive commands has a simple form involving only join operation and constant abstract value, which is defined as follows:
   \[
   [[\text{skip}]]^p(s) = s, \quad [[x := e]]^p(s) = \begin{cases} s[x \mapsto [[e]]^p(s)] & (s \neq \bot) \\ \bot & (s = \bot) \end{cases}
   \]
   where $[e]^p$ has the following form: for every $s \neq \bot$,
   \[
   [e]^p(s) = s(x_1) \sqcup \ldots \sqcup s(x_n) \sqcup v
   \]
   for some variables $x_1, \ldots, x_n$ and an abstract value $v \in \mathcal{V}$, all of which are fixed for the given $e$. We denote these variables and the value by $\text{var}(e) = \{x_1, \ldots, x_n\}$, $\text{const}(e) = v$.

**Example 3** (Impact Pre-Analysis for the Interval Analysis). We design a pre-analysis for our interval analysis in Example 2, which satisfies our soundness and efficiency conditions. The pre-analysis aims at predicting which variables get associated with non-negative intervals when the program is analyzed by an interval analysis with full context-sensitivity $K_\infty$.

1. Let $\mathcal{V} = \{\bot_v, \star_v, \top_v\}$ be a lattice such that $\bot_v \sqsubseteq \star_v \sqsubseteq \top_v$. Define the function $\gamma_v : \{\bot_v, \star_v, \top_v\} \rightarrow \mathcal{P}(\mathcal{V})$ as follows:
   \[
   \gamma_v(\bot_v) = \emptyset, \quad \gamma_v(\star_v) = \{[a, b] \in \mathbb{I} \mid 0 \leq a\}, \quad \gamma_v(\top_v) = \emptyset
   \]
   This function determines the meaning of each element in $\mathcal{V}$ in terms of a collection of intervals. The only non-trivial case is $\star$, which denotes all non-negative intervals according to this function. We include such a case because non-negative intervals, not negative ones, prove buffer-overrun properties.

2. The domain of abstract states is defined as $S^2 = \{\bot\} \cup (\text{Var} \rightarrow \mathcal{V})$. The meaning of abstract states in $S^2$ is given by $\gamma$ such that $\gamma(\bot) = \{\bot\}$ and, for $s^2 \neq \bot$,
   \[
   \gamma(s^2) = \{s \in S \mid s = \bot \lor \forall x \in \text{Var}. s(x) \in \gamma_v(s^2(x))\}.
   \]

3. Initial abstract state: $s_1^p = \lambda x. \top_v$.

4. Abstract evaluation $[e]^p$ of expression $e$: for every $s \neq \bot$,
   \[
   [[e]](s) = \text{ite}(\mathbb{I} \geq 0, \star_v, \top_v), \quad [[e_1 + e_2]](s) = [[e_1]](s) \uplus [[e_2]](s), \quad [[e_1 - e_2]](s) = \top_v.
   \]
   The analysis approximately tracks numbers, but distinguishes the non-negative cases from general ones: non-negative numbers get abstracted to $\star$ by the analysis, but negative numbers are represented by $\top_v$. Observe that the $+$ operator is interpreted as the least upper bound $\uplus_v$, so that $e_1 + e_2$ evaluates to $\star$ only when both $e_1$ and $e_2$ evaluates to $\star$. This implements the intuitive fact that the addition of two non-negative intervals gives another non-negative interval. For expressions involving subtractions, the analysis simply produces $\top_v$.

**Running the pre-analysis via reachability-based algorithm** The class of our pre-analyses enjoys efficient algorithms (e.g., [3, 15]) for computing the least solution $X$ that satisfies (6), even though it is fully context-sensitive. For our purpose, we provide a variant of the graph reachability-based algorithm in [15]. Our algorithm is specialized for our pre-analysis and is more efficient than the algorithm in [15]. Next, we go through each step of our algorithm while introducing concepts necessary to understand it. In the rest of this section, we interchangeably write $K$ for $K_\infty$.

First, our algorithm constructs the value-flow graph of the given program, which is a finite graph $(\Theta, \rightarrow)$ defined as follows:

\[
\Theta = \text{Var} \times \text{Var}, \quad (\rightarrow) \subseteq \Theta \times \Theta
\]

The node set consists of pairs of program nodes and variables, and $(\rightarrow)$ is the edge relation between the nodes.

**Definition 2** $(\rightarrow)$. The value-flow relation $(\rightarrow) \subseteq (\text{Var} \times \text{Var})$ links the vertices in $\Theta$ based on how values of variables flow to other variables in each primitive command:

\[
\begin{align*}
(c, x) &\rightarrow (c', x') \quad \text{iff} \\
&\begin{cases}
(c \rightarrow c' \land x = x') & \quad \text{cmd}(c') = \text{skip} \\
(c \rightarrow c' \land x = x') & \quad \text{cmd}(c') = y := e \land y \neq x' \\
(c \rightarrow c' \land x = \text{var}(e)) & \quad \text{cmd}(c') = y := e \land y = x' 
\end{cases}
\end{align*}
\]

We can extend the $\rightarrow$ to its context-enriched version $\rightarrow_K$:

**Definition 3** $(\rightarrow_K)$. The context-enriched value-flow relation $(\rightarrow_K) \subseteq (\text{Var} \times \text{Var}) \cap (\text{Var} \times \text{Var})$ links the vertices in $\Theta \times \text{Var}$ according to the specification below:

\[
\begin{align*}
((c, \kappa), x) &\rightarrow_K ((c', \kappa'), x') \quad \text{iff} \\
&\begin{cases}
((c, \kappa) \rightarrow_K (c', \kappa') \land x = x') & \quad \text{cmd}(c') = \text{skip} \\
((c, \kappa) \rightarrow_K (c', \kappa') \land x = x') & \quad y \neq x' \\
((c, \kappa) \rightarrow_K (c', \kappa') \land x = \text{var}(e)) & \quad y = x'
\end{cases}
\end{align*}
\]

(where $\text{cmd}(c')$ in the last two cases is $y := e$)

Second, the algorithm computes the interprocedurally-valid reachability relation $(\rightarrow_K) \subseteq \Theta \times \Theta$:

**Definition 4** $(\rightarrow_K)$. The reachability relation $(\rightarrow_K) \subseteq \Theta \times \Theta$ connects two vertices when one node can reach the other via an interprocedurally-valid path:

\[
\begin{align*}
(c, x) &\rightarrow_K (c', x') \quad \text{iff} \\
&\exists \kappa, \kappa'. (c, \kappa) \rightarrow_K (c, \kappa) \land ((c, \kappa), x) \rightarrow_K ((c', \kappa'), x').
\end{align*}
\]
We use the tabulation algorithm in \cite{15} for computing $(\rightarrow^\circ K)$. While computing $(\rightarrow^\circ K)$, the algorithm also collects the set $C'$ of reachable nodes:

$$C' = \{ e \mid \exists k \in \Theta, (e, s) \rightarrow^\circ K (k, r) \}.$$  

(7)

Third, our algorithm computes a set $\Theta_e$ of generators for each abstract value $e$ in $\mathbb{V}$. Generators for $e$ are vertices in $\Theta$ whose commands join $e$ in their abstract semantics:

$$\Theta_e = \{ (c, x) \mid \text{cmd}(c) = x : e \land \text{abstract}(e) = \text{Var} \}$$

$$\cup \{ \text{if } (v = v_0) \text{ then } (i, x) \mid x \in \text{Var} \} \cup \{ \text{else } \} \}

(8)

Finally, using $(\rightarrow K)$ and $\Theta_e$, the algorithm constructs $PA_K$:

**Definition 5 (PA\_K).** $PA_K \in C \rightarrow \mathbb{S}$ is defined as follows:

$$PA_K(c) = \text{if } (c \notin C) \text{ then } \perp$$

$$\text{else } \lambda x. \bigcup \{ v \in \mathbb{V} \mid c(s, x) \rightarrow K \}.$$

Then, $PA_K$ is the solution of our pre-analysis:

**Lemma 2.** Let $X$ be the least solution satisfying (6). Then, $PA_K(c) = \bigcup_{k \in \mathbb{C}^\circ} X(c, k)$.

Our reachability-based algorithm is $|\mathbb{V}|^3$-times faster in the worst case than the RHS algorithm \cite{15}. The algorithm in \cite{15} works on a graph with the following set of vertices:

$$\Theta = \{ (c, s) \mid c \in C \land s \neq \perp \land (\exists x \forall y. y \neq x \implies s(y) = \perp) \}$$

Note that the set $\Theta$ is $|\mathbb{V}|$-times larger than set $\Theta$ used in our algorithm and the worst-case time complexity is cubic on the size of the underlying graph \cite{15}.

### 5.2 Use of the Pre-analysis Results

Using the pre-analysis results, we select queries that are likely to benefit from the increased context-sensitivity of the main analysis. Also, we collect calling contexts that are worth being distinguished during the main analysis. The collected contexts are used to construct a context selector $K$ (Definition 10), which instructs how much context-sensitivity the main analysis should use for each procedure call. This main analysis with $K$ is guaranteed to benefit from the increased context-sensitivity (Proposition 1).

**Query selection** We first select queries that can benefit from increased context-sensitivity. Among given queries $Q \subseteq C \times \mathbb{S} \times \text{Var}$ about the given program, we select the following ones:

$$Q^3 = \{ (c, x) \in (C \times \mathbb{S}) \mid \exists s \in \mathbb{S}, (c, s, x) \in Q \land \forall s' \in (\text{PA}_K(c)), s \cup s' \neq T \}$$

(8)

where $\text{PA}_K(c) : C \rightarrow \mathbb{S}$ is the pre-analysis result. The first conjunct says that $(c, x) \in Q^3$ comes from some query $(c, s, x) \in Q$, and the second conjunct expresses that according to the pre-analysis result, the main analysis does not lose too much information regarding this query. For instance, consider the case of interval analysis. In this case, we are usually interested in checking an assertion like $1 \leq x \leq 10$, which corresponds to a query $(c, s, x)$ with the abstract state $s = (\lambda x. \text{if } (x = 10) \text{ then } 1 \text{ else } \perp)$. Then, the second conjunct in (8) becomes equivalent to $\text{PA}_K(c)(x) \subseteq \mathbb{S}$. That is, we select the query only when the pre-analysis estimates that the variable $x$ will have at least a non-negative interval in the main analysis. In the rest of this section, we assume for brevity that there is only one selected query $(c_q, x_q) \in Q^3$ in the program.

**Building a context selector** Next, we construct a context selector $K : F \rightarrow (C \mathbb{Q})$. $K$ is to answer which calling contexts the main analysis should distinguish in order to achieve most of the benefits of context sensitivity on the given query $(c_q, x_q)$. Our construction considers the following proxy of this goal: which contexts should

the pre-analysis distinguish to achieve the same precision on the selected query $(c_q, x_q)$ as in the case of the full context-sensitivity? In this subsection, we will define a context selector $K$ (Definition 10) that answers this question (Proposition 1).

We construct $K$ in two steps. Before giving our construction, we remind the reader that the impact pre-analysis works on the value-flow graph $(\Theta, \rightarrow)$ of the program and computes the reachability relation $(\rightarrow^K) \subseteq \Theta \times \Theta$ over the interprocedurally-valid paths. The first step is to build a program slice that includes all the dependencies of the query $(c_q, x_q)$. A query $(c_q, x_q)$ depends on a vertex $(c, x)$ in the value-flow graph if there exists an interprocedurally-valid path between $(c, x)$ and $(c_q, x_q)$ on the graph (i.e., $(c, x) \rightarrow^K (c_q, x_q)$). Tracing the dependency backwards from the query eventually hits vertices with no predecessors. We call such vertices *sources* and denote their set by $\Phi$:

**Definition 6 (\Phi).** Sources $\Phi$ are vertices in $\Theta$ where dependencies begin: $\Phi = \{ (c_0, x_0) \in \Theta \mid \exists (c, x) \in \Theta, (c, x) \rightarrow (c_0, x_0) \}$. We compute the set $\Phi(q_c, x_q)$ of sources on which the query $(c_q, x_q)$ depends:

**Definition 7 (\Phi(q_c, x_q)).** Sources on which the query $(c_q, x_q)$ depends: $\Phi(q_c, x_q) = \{ (c_0, x_0) \in \Theta \mid (c_0, x_0) \rightarrow^K (c_q, x_q) \}$. We compute the set $\Phi(q_c, x_q)$ of sources on which the query $(c_q, x_q)$ depends:

**Definition 8 (\text{Paths}(q_c, x_q)).** The set of all dependency paths for the query $(c_q, x_q)$ is defined as follows:

$$\text{Paths}(q_c, x_q) = \{ \{(c_0, x_0), (c_0, x_0) \rightarrow^K (c_q, x_q) \} \mid (c_0, x_0) \in \Phi(q_c, x_q) \land \{ (t, e) \rightarrow^* K \} \land (c_0, x_0) \}

\text{Paths}(q_c, x_q) \text{ is the program slice we intend to construct in this step.}

**Example 5.** In Figure 2, suppose that $k_0$ and $k_1$ are the initial contexts of procedures $m$ and $b$, respectively. For source $(1, x)$, we find the following dependency path to the query $(6, z)$:

$$p_1 = ((1, k_0), y) \rightarrow^K ((2, k_0), x) \rightarrow^K ((5, 2, k_0), y) \rightarrow^K ((6, 4, 2, k_0), z) \rightarrow^K ((6, 8, k_0), z) \rightarrow^K ((6, 8, k_0), z)\}

Then, $\text{Paths}(6, x) = \{ p_1, p_2 \}$. The next step is to compute calling contexts that should be treated precisely. Consider a dependency path from $\text{Paths}(q_c, x_q)$:

$$\{(c_0, k_0), x_0) \rightarrow^K (c_q, k_q), x_q \}

(10)

where $K_0$, $k_1$, $k_2$, $k_3$ are the calling contexts appeared in the (fully context-sensitive) pre-analysis. Instead, we are interested in partial contexts that represent the “difference” between $k_0$ and $k_0$. Intuitively, if $k_0$ is a suffix of $k_1$, i.e., $k_1 = k_0 \cdot k_0$, the partial context for $k_0$ is defined as $k_0$. Formally, we define the partial calling contexts of $k_0$ as $k_0 \odot k_0 = k_0 - \text{suffix}(k_0, k_0)$ where $\text{suffix}(k_1, k_2)$ is the longest common suffix of $k_1$ and $k_2$. For example, when $k_1$ is a suffix of $k_0$, we use $e$ as the partial context for $k_0$ if $k_0 = e \cdot c_1$ and $k_2 = c_1$, then $k_1 \odot k_0 = e$. Suppose that $k_1$ and $k_2$ are not a
suffix of each other, for instance κ₀ = c₂ · c₁ and κ₁ = c₃ · c₁. In this case, κ₁ ⊙ κ₀ = c₃.

Assumption 1. In general, the above definition of partial contexts requires that the input program should be well-formed with respect to the query: for a path (10) from a source to the query, every call site, cᵢ ∈ C_c, in that path should not be included in the initial context κ₀. We require this condition because our selective context-sensitive analysis aims at distinguishing only the calls after passing the sources of dependency and analyzing context-insensitively those encountered before reaching those sources, which do not contribute to the query. This well-formedness assumption is not a strong restriction and its violation nearly never happens in practice. We did not observe any violation of the assumption in our benchmark programs (Section 7). If the program is not well-formed to a query, then we simply ignore it.

Let us explain the condition with an example. Suppose that κ₀ = c₃ · c₂ · c₁ is the initial context at c₀ and κᵢ = c₁ is the context at cᵢ. Suppose further that c₁ is a call node. Then, our condition requires that c₁ should not be one of call site c₁, c₂, and c₃. Formally, the condition is defined as follows:

We say the given program is well-formed with respect to the query (cₐ, xₐ) iff for every (c₀, x₀) ∈ Φ(cₐ, xₐ) and its valid value-path flow

\[(c₀, κ₀), x₀) \ltarrow K_∞ \cdot \ldots \ltarrow K_∞ \cdot (cₐ, κₐ), xₐ)\]

for all 0 ≤ i ≤ n such that cᵢ ∈ C_c, c₁ is not included in the initial calling context κ₀; i.e.

\[c₁ \notin κ₀\]  

where we write c ∈ κ when there exists some κ’ such that c · κ’ is a suffix of κ.

In summary, for the path in (10), collecting contexts

\[\{κ₀ \ominus κ₀, \ldots, κₗ \ominus κ₀\}\]

give all the necessary partial calling contexts, where each κₗ \ominus κ₀ belongs to the calling contexts of procedure fid(cₗ). Thus, we define the context selector for the dependency path (10) as follows:

Definition 9 (Kₚ, Context Selector for Path p). Let p be a dependency path from a source (c₀, x₀) to query (cₐ, xₐ):

\[p = ((c₀, κ₀), x₀) \ltarrow K_∞ \cdot \ldots \ltarrow K_∞ \cdot ((cₐ, κₐ), xₐ),\]

where κ₀ is an initial context at c₀ such that (ℓ, ε) →_K_∞ (c₀, x₀). The context selector Kₚ for the path is defined as:

\[Kₚ = λf. \{κ₁ \ominus κ₀ | fid(cₐ) = f \wedge \{(cᵢ, κᵢ), \ldots \} \in p\}\]

Example 6. From the path p₁ in Example 5, the collection of κᵢ is \{κ₀, 2 · κ₀, 4 · 2 · κ₀\} (see Figure 2). Hence, the collection of κ₁ ⊙ κ₀ is \{ε, 2, 4 · 2\}, where ε belongs to procedure m 2 to f, and 4 · 2 to g. Similar for path p₂. Thus, Kₚ₁ and Kₚ₂ are:

\[Kₚ₁ = \begin{bmatrix} m & \{ε\} \\ f & \{2\} \\ g & \{4 · 2\} \end{bmatrix} \quad Kₚ₂ = \begin{bmatrix} h & \{ε\} \\ g & \{8\} \end{bmatrix}\]

Then, the final context selector K is the union of Kₚ’s:

Definition 10 (K, Context Selector). Let (cₐ, xₐ) be a query. The context selector \(K \in \mathcal{P} \rightarrow \mathcal{P}(\mathcal{C}_c)\) for our selective analysis is:

\[K(f) = \mathcal{E}(f) \cup \bigcup \{K_p(f) | p \in \text{Paths}(cₐ, xₐ)\} \]  

where \(\mathcal{E}(f) = \{ε\} \) if \(f \neq \text{fid}(cₐ)\); and otherwise, \(\mathcal{E}(f) = \emptyset\).

Running selective context-sensitive main analysis Finally, we run the main analysis with selective context-sensitivity K defined by the result of the impact pre-analysis. The following proposition states that the pre-analysis-guided context-sensitivity (K) manages to pay off at the selective main analysis, although the pre-analysis is fully context-sensitive and the main analysis is not.

Proposition 1 (Impact Realization). Let \(\text{PA}_K \subseteq \mathcal{P} \rightarrow \mathcal{P}(\mathcal{C}_c)\) be the result of the impact pre-analysis (Definition 5). Let q ∈ \(\mathcal{P}(\mathcal{C}_c)\) be a selected query (8). Let K be the context selector for q (Definition 10) defined using the pre-analysis result \(\text{PA}_K\). Let \(\text{MA}_K \subseteq \mathcal{P} \rightarrow \mathcal{P}(\mathcal{C}_c)\) be the main analysis result with the context selector K. Then, the selective main analysis is at least as precise as the fully context-sensitive pre-analysis for the selected query q:

\[\text{MA}_K \subseteq \mathcal{P} \rightarrow \mathcal{P}_K\]

where \(\text{MA}_K \subseteq \mathcal{P} \rightarrow \mathcal{P}_K\) iff \((q \overset{((cₐ, xₐ))}{\rightarrow} (cₐ, xₐ)) \in \gamma(T[x \mapsto \text{PA}_K(cₐ, xₐ)]))\).

This impact realization holds thanks to two key properties. First, our selective context-sensitivity K (Definition 10) distinguishes all the calling contexts that matter for the queries selected by the pre-analysis. Second, the main analysis designed in Section 4 isolates these distinguished contexts from other undistinguished contexts (ε), ensuring that spurious flows caused by merging contexts never adversely affect the precision of the selected query.

6. Application to Selective Relational Analysis

A general principle behind our method is that we can selectively improve the precision of the analysis by using an impact pre-analysis that estimates the main static analysis of the maximal precision. In this section, we use the same principle to develop a selective relational analysis with the octagon domain [11].

Overview Consider the following code snippet:

```c
int a = b;  
int c = input(); // User input 
for (i = 0; i < b; i++) {
    assert (i < a); // Query 1 
    assert (i < c); // Query 2 
}
```

The first query at line 4 always holds but the second one at line 5 is not necessarily true.

A fully relational octagon analysis, which tracks contraints of the form ±x ± y ≤ c (where c ∈ \(\mathbb{Z} \cup \{∞\}\)) between all variables...
Consider the positive form $x$ and negative form $\bar{x}$ for each variable $x$ and represent an octagon domain element $o \in O$ by a $2|\mathcal{V}| \times 2|\mathcal{V}|$ matrix where each entry $\alpha_{xy} \in [\infty \to \infty]$ stores the upper bound of $y - x$. The definition of $\mathit{cmd}$ for our commands can be found at [11].

With $O$ and $\mathit{cmd}$, we define the domain of packed octagons that assign an octagon to a subset of variables, which we call $\mathit{pack}$.

An octagon of a pack expresses only the constraints of the variables in that pack. We call $P \subseteq \wp(\mathcal{V})$ of sets of packing configuration, such that $\bigcup P = \mathcal{V}$. The packed octagon domain $\mathcal{P}^O(P)$ parameterized by packing configuration $P$ is then defined as $\mathcal{P}^O(P) = \mathcal{P} \to P \to \mathcal{P}$. We extend the abstract semantics $\mathit{cmd}$ : $\mathcal{O} \to \mathcal{O}$ of command $\mathit{cmd}$ to $\mathit{cmd}^P$ : $\mathcal{P}^O(P) \to \mathcal{P}^O(P)$ as follows:

$$[[\mathit{cmd}]^P(o) = \lambda \pi . \begin{cases} \mathit{cmd} = \mathit{cmd}_o & \text{if } o \in O \text{ and } \pi = \emptyset \\ \mathit{cmd} = \mathit{cmd}_o & \text{if } o \in O \text{ and } \pi \neq \emptyset \\ \mathit{cmd} = \mathit{cmd}_o & \text{otherwise} \end{cases} \quad \mathit{cmd} = \mathit{cmd}_o \in \mathcal{O}^2$$

The extended abstract semantics is essentially the same except it forgets all the relationships of the assignee $x$ (the second case) when the pack is missing one variable involved in the octagonal constraint. The abstract semantics of program in $\mathcal{D} = \mathcal{C} \to \mathcal{P}^O(P)$ is defined as the least fixpoint of abstract transfer function $\mathit{lfp}^P : \mathcal{D} \to \mathcal{D}$, which is defined as usual.

The selectivity of the analysis is governed by the configuration $P$. For instance, with $P = \{ \{x\} \mid x \in \mathcal{V} \}$, the analysis degenerates to a non-relational analysis. With $P = \{\mathcal{V}\}$, the analysis becomes a fully relational analysis. Our goal is to find a cost-effective $P$ by using an impact pre-analysis.

### Impact pre-analysis

Second, we formally define the impact pre-analysis. The meaning of our abstract values ($\mathcal{V} = \{\#, \star, \top, \bot\}$) is described by $\gamma_\mathcal{V}$ such that $\gamma_\mathcal{V}(\#) = \mathcal{V}$ and $\gamma_\mathcal{V}(\star) = \mathcal{Z} \cup \{\infty\}$. The abstract state $O^2 = \{\bot\} \cup \mathcal{V}^2|\mathcal{V} \to 2|\mathcal{V}|$ of our pre-analysis is the set of matrices whose entries are in $\mathcal{V}$. An abstract state $o^2 \in O^2$ denotes a set of octagons: we define $\gamma : O^2 \to \wp(\mathcal{O})$ as follows:

$$\gamma(o^2) = \{ o \in \mathcal{O} \mid \forall i, j, o_{ij} \in \gamma_\mathcal{V}(o^2) \}$$

The abstract semantics $\mathit{cmd}^P$ : $O^2 \to O^2$ of each primitive command $\mathit{cmd}$ of the pre-analysis is defined as an over-approximation of the abstract semantics of the main analyses: e.g.,

$$\{[x := y](o^2)\}_{ij} = \begin{cases} \star & (i = j \land x = y) \\ \top & (i \not\in \{x, \bar{x}\} \land j \not\in \{x, \bar{x}\}) \\ o^2_{ij} \land \neg (i = j \land x = y) & \text{otherwise} \end{cases}$$

The abstract domain of the pre-analysis is $\mathcal{D}^P = \mathcal{C} \to \mathcal{O}^2$ and the pre-analysis result is defined as the least fixpoint of semantic function $\mathit{lfp}^P : \mathcal{D}^P \to \mathcal{D}^P$, which is defined as usual.

### Use of pre-analysis results

From the pre-analysis results ($\mathit{lfp}^P \sharp$), we construct $P$ as follows. Assume that a set $\mathcal{Q} \subseteq \mathcal{C} \times \mathcal{V} \times \mathcal{V}$ of relational queries is given in the program. A query $(c, x, y) \in \mathcal{Q}$ represents a predicate $y - x < 0$ at program point $c$ and we say that $c \in \mathcal{Q}$ proves the query when $o_{xy} < -1$. We first select a set $Q^2$ of queries that are judged promising by the pre-analysis:

$$Q^2 = \{ (c, x, y) \in \mathcal{Q} \mid (\mathit{lfp}^P\sharp)(c) \not\perp \perp \wedge (\mathit{lfp}^P\sharp)(c)_{xy} = \# \}.$$

Next, for each selected query $(c, x, y) \in Q^2$, we compute the pack $\mathit{pack}_{(c,x,y)} \subseteq \mathcal{V}$ of necessary variables using dependency analysis, which is simultaneously done with the pre-analysis as follows: let $\mathcal{V}^2 = \mathcal{V} \times \wp(\mathcal{V})$ and $O^2$ be the set of $2|\mathcal{V}| \times 2|\mathcal{V}|$ matrices over $\mathcal{V}^2$. The idea is to over-approximate the involved variables for each octagon constraint in the second component of $\mathcal{V}^2$. The abstract
chains is not uncommon in the interval analysis of C programs. Analysis time from 707.1s to 903.6s (27.8% increase). In doing so, our technique increases the total number down to 9,598 (24.4% reduction). In total, the context-insensitive analysis precision improves. In total, the context-insensitive analysis precision improves.

We have implemented our selective method on top of the octave-analysis version of our baseline analyzer. We compare the performance of our selective analysis with an existing context-sensitive approach based on the syntactic variable packing [11, 13]. The syntactic packing approach relates variables together if they are involved in the same syntactic block [11]. We limited the maximum pack size by 10 in the syntactic packing strategy, since otherwise the analysis did not scale.

Table 2 shows our benchmark programs. Note, that although a relational analysis is a key to proving important numerical properties, it is useful only for specific target programs and queries [4, 11]. Thus, we first identified a set of benchmark programs and their buffer-overrun queries whose proofs require relational information, and compared the performance of the two analyses on these programs and queries. Column #Query shows the number of relational queries that we consider in our experiments. In the experiments, we manually in-lined the functions that are involved in the proofs of the target queries, so that our selective relational analysis and the syntactic packing approach are run under context-sensitivity.

The results show that our selective octave analysis has a competitive precision-cost balance. Among 135 queries in total, our analysis is able to prove 132 (97.8%) queries in 3,632.7s. On the other hand, the octave analysis with syntactic packing proved 44 (32.6%) queries in 35,840s: the syntactic packing heuristic often fails to prescribe variable relationships necessary to prove queries. Our analysis is even faster than the counterpart in most cases because it selectively turns on relational analysis.

One thing to note is that running our pre-analysis is feasible in practice even though it is fully relational. The bottlenecks of a fully relational octave analysis are the memory costs for representing 2|Var| × 2|Var| matrices and the expensive strong closure operation [11] whose time complexity is cubic in the number of variables. Thanks to the simplicity of the abstract domain (★ or □), we can reduce the memory cost using a sparse representation for the matrices. For the closure operation, we use Dijkstra’s algorithm and compute the shortest-path closure [11] instead of the strong closure. In our experiments, using the shortest-path closure made no difference in the pre-analysis precision.

### Related Work

Most of the previous context-sensitive analysis techniques assign contexts to calls in a uniform manner. The k-callstring approach (or k-CFA) [16, 17] and its flexible variants [6], k-object sensitivity [10], and type sensitivity [18] are such cases. All these techniques generate calling contexts according to a single fixed policy and do not explore how to tune their parameters (for example, different k values at each call site) for target queries. The hybrid context-sensitivity [8], which employs multiple policies of assigning contexts in a single analysis, still does not tailor those policies to the program to analyze. There are also other approaches to context-sensitivity based on function summaries like [15], but here we do not discuss them as it is by itself a challenge to design a summary-based analysis with abstract domains of infinite height.

While refinement-based analyses [5, 14, 20] are similar to our approach (in that they use a “pre-analysis” to adjust the main analysis precision), there is a fundamental difference in their techniques. Refinement-based approaches (e.g., client-driven analysis [5]) start with an imprecise analysis and refines the abstraction in response to client queries. On the other hand, our approach starts with a pre-analysis that estimates the impact of the most precise main
Our Selective Context-Sensitive Analysis

<table>
<thead>
<tr>
<th>Program</th>
<th>LOC</th>
<th>Proc</th>
<th>Context-Insensitive</th>
<th>Our Selective Context-Sensitive Analysis</th>
<th>Alarm</th>
<th>Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>#alarm</td>
<td>time</td>
<td>#alarm</td>
</tr>
<tr>
<td>spell-1.0</td>
<td>2,113</td>
<td>31</td>
<td>58</td>
<td>0.6</td>
<td>30</td>
<td>0.1</td>
</tr>
<tr>
<td>bc-1.06</td>
<td>13,093</td>
<td>134</td>
<td>606</td>
<td>14.0</td>
<td>483</td>
<td>1.9</td>
</tr>
<tr>
<td>tar-1.17</td>
<td>20,258</td>
<td>222</td>
<td>940</td>
<td>42.1</td>
<td>799</td>
<td>5.9</td>
</tr>
<tr>
<td>less-382</td>
<td>23,822</td>
<td>382</td>
<td>654</td>
<td>123.0</td>
<td>563</td>
<td>3.3</td>
</tr>
<tr>
<td>sed-4.08</td>
<td>26,807</td>
<td>294</td>
<td>1,325</td>
<td>107.5</td>
<td>1,238</td>
<td>7.4</td>
</tr>
<tr>
<td>make-3.76</td>
<td>27,304</td>
<td>191</td>
<td>1,500</td>
<td>84.4</td>
<td>1,028</td>
<td>7.1</td>
</tr>
<tr>
<td>grep-2.5</td>
<td>31,495</td>
<td>153</td>
<td>713</td>
<td>12.1</td>
<td>653</td>
<td>2.4</td>
</tr>
<tr>
<td>wget-1.9</td>
<td>35,018</td>
<td>434</td>
<td>1,307</td>
<td>69.0</td>
<td>942</td>
<td>12.5</td>
</tr>
<tr>
<td>a2ps-4.14</td>
<td>64,900</td>
<td>980</td>
<td>3,682</td>
<td>118.1</td>
<td>2,121</td>
<td>29.5</td>
</tr>
<tr>
<td>bison-2.5</td>
<td>101,807</td>
<td>1,427</td>
<td>1,894</td>
<td>136.3</td>
<td>1,742</td>
<td>34.6</td>
</tr>
<tr>
<td>Total</td>
<td>346,407</td>
<td>4,248</td>
<td>12,701</td>
<td>707.1</td>
<td>9,598</td>
<td>173.4</td>
</tr>
</tbody>
</table>

Table 1. Performance comparison between context-insensitive analysis and our selective context-sensitive analysis. LOC reports lines of code before pre-processing. Proc shows the number of procedures in the programs. #alarm reports the number of buffer-overrun alarms raised by the analyses. pre reports the time spent for running the pre-analysis (including query selection and building context selector) and main reports the time spent by the main analysis of our approach. Each entry a/b (c%) in column selected call-sites means that, among a b call-sites in the program, a c call-sites are selected for context-sensitivity by our pre-analysis and the selection ratio is c%, ~ reports the maximum call-depth prescribed by the pre-analysis. Overhead. pre shows the time overhead and main reports the cost increase in the main analysis due to increased context-sensitivity, compared to the context-insensitive analysis.

Our Selective Relational Analysis

<table>
<thead>
<tr>
<th>Program</th>
<th>LOC</th>
<th>#Variable</th>
<th>#Query</th>
<th>Syntactic Packing Approach</th>
<th>Our Selective Relational Analysis</th>
<th>Comparison</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>proven</td>
<td>time</td>
<td>mem</td>
</tr>
<tr>
<td>calculator-1.0</td>
<td>298</td>
<td>197</td>
<td>10</td>
<td>2</td>
<td>0.3</td>
<td>63</td>
</tr>
<tr>
<td>spell-1.0</td>
<td>2,213</td>
<td>531</td>
<td>16</td>
<td>1</td>
<td>4.8</td>
<td>109</td>
</tr>
<tr>
<td>barcodes-0.96</td>
<td>4,460</td>
<td>2,002</td>
<td>37</td>
<td>16</td>
<td>11.8</td>
<td>221</td>
</tr>
<tr>
<td>bit Humphrey-3.3</td>
<td>6,174</td>
<td>1,908</td>
<td>28</td>
<td>16</td>
<td>26.0</td>
<td>220</td>
</tr>
<tr>
<td>bc-1.06</td>
<td>13,093</td>
<td>2,194</td>
<td>10</td>
<td>2</td>
<td>247.1</td>
<td>945</td>
</tr>
<tr>
<td>tar-1.17</td>
<td>20,258</td>
<td>5,332</td>
<td>17</td>
<td>7</td>
<td>1,043.2</td>
<td>1,131</td>
</tr>
<tr>
<td>less-382</td>
<td>23,822</td>
<td>4,482</td>
<td>13</td>
<td>0</td>
<td>3,031.5</td>
<td>1,439</td>
</tr>
<tr>
<td>a2ps-4.14</td>
<td>64,900</td>
<td>16,531</td>
<td>17</td>
<td>0</td>
<td>29,479</td>
<td>3,304</td>
</tr>
<tr>
<td>Total</td>
<td>135,008</td>
<td>33,177</td>
<td>14</td>
<td>44</td>
<td>33,840</td>
<td>6,611</td>
</tr>
</tbody>
</table>

Table 2. Performance comparison between an octagon analysis with an existing syntactic packing strategy and our selective relational analysis. #Variable denotes the number of variables (abstract locations) in the program. #Query denotes the number of buffer-overrun queries whose proofs require relational reasoning. proven reports the number of queries that are proven by each octagon analysis. mem reports the peak memory consumption in megabytes. Each X/Y in column pack represents the number of non-singleton packs (X) and the average size (Y) of the packs used in each relational analysis. Precision and Time shows additionally proven queries and time consumption by our selective relational analysis compared to the syntactic packing approach.

Conclusion

We proposed a method of designing a selective “X-sensitive” analysis, where the selection is guided by an impact pre-analysis. We followed this approach, presented two program analyses that selectively apply precision-improving techniques, and demonstrated their effectiveness with experiments in a realistic setting. The first was a selective context-sensitive analysis that receives guidance from an impact pre-analysis. Our experiments with realistic benchmarks showed that the method reduces the number of false alarms of a context-insensitive interval analysis by 24.4%, while increasing the analysis cost by 27.8%. The second example was a selective relational analysis with octagons using the same idea of impact pre-analysis, and our experiments showed that our selective octagon analysis proves 88 more queries than the existing one based on the syntactic variable packing and reduces the analysis cost by 81%. We believe that our approach can be used for developing other selective analyses as well, e.g., selective flow-sensitive analysis, selective loop-unrolling, etc.

Acknowledgements This work was partially supported by the Engineering Research Center of Excellence Program of Korea Ministry of Science, ICT & Future Planning (MSIP) / National Research Foundation of Korea (NRF) (Grant NRF-2008-0062609) and Samsung Electronics. Yang was supported by EPSRC.
A. Proofs

In this appendix, we prove the impact guarantee (Proposition 1) of our selective context-sensitive analysis as well as the correctness of our reachability-based pre-analysis algorithm (Lemma 2).

Remark 1. In the rest of this appendix, we generalize the notion of the impact pre-analysis, and write $\text{PA}_K : C \rightarrow \mathcal{S}^3$ for the solution of the pre-analysis under context-sensitivity $K$. In the body of this paper, we have discussed our pre-analysis and its reachability-based algorithm only under full context-sensitivity ($K = K_\infty$). However, the correctness of our pre-analysis (Lemma 2) holds with arbitrary context selector $K$, as its proof does not assume a particular instance of $K$ (we prove this generalized lemma in A.2).

This implies that, regardless of the underlying context-sensitivity, we can compute the least solution of the pre-analysis (6) via our reachability algorithm.

### A.1 Proof of Proposition 1

**Proof.** To show:

$$\forall \kappa \in K(\text{fid}(c)), \text{MA}_K(\kappa, c) \in \gamma(\mathcal{T}[x \mapsto \text{PA}_{K_\infty}(c)(x)])$$

It is proved by Lemma 3 and 6:

$$\gamma(\mathcal{T}[x \mapsto \text{PA}_{K_\infty}(c)(x)])$$

(Lemma 3)

$$\subseteq \gamma(\text{PA}_K(c))$$

(Soundness of $\mathcal{T}$)

$$\subseteq \text{MA}_K(\kappa, c)$$

(Lemma 6)

$\square$

### A.2 Proofs of Lemmas

#### Lemma 3 (Pre-analysis Coincidence)

Let $(c_q, x_q)$ be a query. Let $\text{PA}_{K_\infty}$ be the pre-analysis result with full context-sensitivity. Let $K$ be the selective context-sensitivity for query $(c_q, x_q)$ defined using $\text{PA}_{K_\infty}$ (Definition 10). Let $\text{PA}_K$ be the result of the pre-analysis under the selective context-sensitivity $K$. Then, $\text{PA}_{K_\infty}(c_q) = \text{PA}_K(c_q)$ if and only if $\text{PA}_{K_\infty}(c_q)(x_q) = \text{PA}_K(c_q)(x_q)$.

**Proof.** It is sufficient to show that, in the value-flow graph, the query $(c_q, x_q)$ is reachable from a 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 $K$.

$$\forall (c_0, x_0) \in \Phi, (c_0, x_0) \xrightarrow{t} 1_{K_\infty} (c_q, x_q) \iff (c_0, x_0) \xrightarrow{t} K (c_q, x_q).$$

$\bullet (\implies)$ By Lemma 4.

$\bullet (\impliedby)$ When $(c_0, x_0) \notin \Phi_{(c_q, x_q)}$, by the definition of $\Phi_{(c_q, x_q)}$.

We prove this lemma in two steps. We first show that for every $(c, x) \in \Phi_{(c_q, x_q)}$, we have $(c, x) \xrightarrow{t} 1_{K_\infty} (c_q, x_q)$ (Lemma 4). Then, we prove that there is no $(c', x') \notin \Phi_{(c_q, x_q)}$ such that $(c', x') \xrightarrow{t} K (c_q, x_q)$ (Lemma 5). Comprising these two lemmas, we can prove Lemma 3.

The following lemma shows that, for every $K_\infty$-valid value-flow path from sources to the query, we can always find the corresponding $K$-valid value-flow path.

#### Lemma 4. Suppose $(c_0, x_0) \in \Phi_{(c_q, x_q)}$ and consider $c_1, \kappa_1, x_1$ such that

$$(s, e) \rightarrow 1_{K_\infty} (c_0, x_0) \land ((c_0, \kappa_0), x_0) \rightarrow_{K_\infty} ((c_1, \kappa_1), x_1) \rightarrow_{K_\infty} \cdots \rightarrow_{K_\infty} ((c_q, \kappa_q), x_q).$$

Then, we have

$$(c_0, \kappa_0), x_0) \rightarrow_{K} ((c_1, \kappa_1'), x_1) \rightarrow_{K} \cdots \rightarrow_{K} ((c_q, \kappa_q'), x_q).$$

where $\kappa_i' = \kappa_i \ominus \kappa_0$.

**Proof.** By the definition of $K$,

$$\forall i, \kappa_i' = \kappa_i \ominus \kappa_0 \in K.$$

We prove this lemma in two steps. We first show that for every $(c, x) \in \Phi_{(c_q, x_q)}$, we have $(c, x) \xrightarrow{t} 1_{K_\infty} (c_q, x_q)$ (Lemma 4). Then, we prove that there is no $(c', x') \notin \Phi_{(c_q, x_q)}$ such that $(c', x') \xrightarrow{t} K (c_q, x_q)$ (Lemma 5). Comprising these two lemmas, we can prove Lemma 3.

The following lemma shows that, for every $K_\infty$-valid value-flow path from sources to the query, we can always find the corresponding $K$-valid value-flow path.

#### Lemma 5. Suppose $(c_0, x_0) \in \Phi_{(c_q, x_q)}$ and consider $c_1, \kappa_1, x_1$ such that

$$(s, e) \rightarrow 1_{K_\infty} (c_0, x_0) \land ((c_0, \kappa_0), x_0) \rightarrow_{K_\infty} ((c_1, \kappa_1), x_1) \rightarrow_{K_\infty} \cdots \rightarrow_{K_\infty} ((c_q, \kappa_q), x_q).$$

Then, we have

$$(c_0, \kappa_0), x_0) \rightarrow_{K} ((c_1, \kappa_1'), x_1) \rightarrow_{K} \cdots \rightarrow_{K} ((c_q, \kappa_q'), x_q).$$

where $\kappa_i' = \kappa_i \ominus \kappa_0$.

**Proof.** By the definition of $K$,

$$\forall i, \kappa_i' = \kappa_i \ominus \kappa_0 \in K.$$
We show that
\[
\forall 0 \leq i < n, \quad ((c_i, \kappa_i), x_i) \leadsto_{K_{\infty}} ((c_{i+1}, \kappa_{i+1}), x_{i+1}) \\
\implies ((c_i, \kappa_i), x_i) \leadsto (c_{i+1}, \kappa_{i+1}), x_{i+1})
\]
where \( c_0 = c_q, \kappa_0 = \kappa_q \), and \( x_0 = x_q \). This simply amounts to showing the following:
\[
\forall 0 \leq i < n, \quad ((c_i, \kappa_i) \leadsto_{K_{\infty}} (c_{i+1}, \kappa_{i+1}) \implies ((c_i, \kappa_i') \leadsto_{K} (c_{i+1}, \kappa_{i+1}')).
\]

1. \( c_i \not\in C_c \cup C_r \):
   By the definition of \( \leadsto_{K_{\infty}} \),
   \[
   \kappa_i = \kappa_{i+1}, \quad \kappa_i \circ \kappa_0 = \kappa_{i+1} \circ \kappa_0.
   \]
   By the definition of \( \leadsto_{K} \),
   \[
   (c_i, \kappa_i') \leadsto_{K} (c_{i+1}, \kappa_{i+1}').
   \]

2. \( c_i \in C_c \):
   By the definition of \( \leadsto_{K_{\infty}} \),
   \[
   \kappa_{i+1} = c_i \cdot \kappa_i, \quad \kappa_{i+1} \circ \kappa_0 = (c_i \cdot \kappa_i) \circ \kappa_0.
   \]
   By the definition of \( K \),
   \[
   (c_i, \kappa_i) \circ \kappa_0 \notin \epsilon.
   \]
   and hence
   \[
   (c_i, \kappa_i) \circ \kappa_0 = c_i \cdot (k_i \circ \kappa_0) \in K.
   \]
   Therefore, by the definition of \( \leadsto_{K} \) and \( \leadsto_{K_{\infty}} \), we have
   \[
   (c_i, \kappa_i) \leadsto_{K} (c_{i+1}, \kappa_i, c_i : (\kappa_i \circ \kappa_0)).
   \]

3. \( c_i \in C_r \):
   By the definition of \( \leadsto_{K_{\infty}} \),
   \[
   \kappa_i = \text{callof}(c_{i+1}) \cdot \kappa_{i+1}
   \]
   and hence
   \[
   \kappa_i \circ \kappa_0 = (\text{callof}(c_{i+1}) \cdot \kappa_{i+1}) \circ \kappa_0.
   \]

Now we split into two cases.

(a) When \( \text{callof}(c_{i+1}) \cdot \kappa_{i+1} \circ \kappa_0 \neq \epsilon \),
   we have
   \[
   \kappa_i' = (\text{callof}(c_{i+1}) \cdot \kappa_{i+1}) \circ \kappa_0 = \text{callof}(c_{i+1}) \cdot (\kappa_{i+1} \circ \kappa_0) = \text{callof}(c_{i+1}) \cdot \kappa_{i+1}.
   \]
   From the definition of \( K \), we have \( \kappa_i', \kappa_{i+1} \in K \). Therefore, by the definition of \( \leadsto_{K} \), we have
   \[
   \kappa_i' = \text{callof}(c_{i+1}), \kappa_{i+1} \in K.
   \]
   By the definition of \( \leadsto_{K} \),
   \[
   (c_i, \kappa_i') \leadsto_{K} (c_{i+1}, \kappa_{i+1}).
   \]

(b) When \( \text{callof}(c_{i+1}) \cdot \kappa_{i+1} \circ \kappa_0 = \epsilon \),
   we have
   \[
   \kappa_i \circ \kappa_0 = \kappa_{i+1} \circ \kappa_0 = \epsilon.
   \]
   which means that \( \kappa_i' = \kappa_{i+1}' = \epsilon \). Also, note that we have
   \[
   \text{callof}(c_{i+1}) \in K_0
   \]
   from \( \text{callof}(c_{i+1}) \cdot \kappa_{i+1} \circ \kappa_0 = \epsilon \). If we assume that \( \text{callof}(c_{i+1}) \cdot \kappa_{i+1} \not\in K \), we can conclude that
   \[
   (c_i, \kappa_i') \leadsto_{K} (c_{i+1}, \kappa_{i+1}').
   \]

Now we prove that the assumption is actually true:
\[
\text{callof}(c_{i+1}) \cdot \kappa_{i+1} = \text{callof}(c_{i+1}) \not\in K.
\]

Suppose \( \text{callof}(c_{i+1}) \in K \). Then, there should exist \( c_j = \text{callof}(c_{i+1}) \in C_c \) such that
\[
((c_0, \kappa_0), x_0) \leadsto_{K_{\infty}} ((c_j, \kappa_j), x_j) \leadsto_{K_{\infty}^+} ((c_n, \kappa_n), x_n)
\]
and
\[
(c_j \cdot \kappa_j) \circ \kappa_0 = c_j \in K.
\]
By the Assumption 1, \( c_j = \text{callof}(c_{i+1}) \not\in K_0 \), which contradicts (15).

\[\square\]

The following lemma formalizes the fact that our selective context-sensitive analysis designed in Section 4 isolates undistinguished contexts from distinguished contexts: if a source does not reach the query in the fully context-sensitive pre-analysis, then the source does not reach the query in the selective context-sensitive pre-analysis as well.

**Lemma 5 (Isolation).** For all \( (c_0, x_0) \in \Phi \setminus \Phi(c_q, x_q) \),
\[
(c_0, x_0) \not\leadsto_{K} (c_q, x_q).
\]

**Proof.** Suppose we have \( (c_0, x_0) \in \Phi \setminus \Phi(c_q, x_q) \) such that
\[
(c_0, x_0) \leadsto_{K} (c_q, x_q).
\]
Then, by the definition of \( \leadsto_{K_0} \), there exists a \( \leadsto_{K} \)-path
\[
((c_0, \kappa_0), x_0) \leadsto_{K_{\infty}^+} ((c_q, \kappa_q), x_q),
\]
for some \( \kappa_0 \) and \( \kappa_q \), which means that we have a \( \leadsto_{K} \)-path
\[
(c_0, \kappa_0) \leadsto_{K} \leadsto_{K_{\infty}^+} (c_q, \kappa_q).
\]
Because \( \kappa_q \in K(\text{fid}(c_q)) \), by the definition of \( K \) we have a \( \leadsto_{K_{\infty}^+} \)-path from some source \( (c_s, x_s) \in \Phi(c_{l_s}, x_{l_s}) \)
\[
((c_s, \kappa_s), x_s) \leadsto_{K_{\infty}^+} \leadsto_{K_{\infty}^+} \leadsto_{K_{\infty}^+} ((c_q, \kappa_q'), x_q),
\]
where \( \kappa_q = \kappa_q' \circ \kappa_s \). Then, by Lemma 4, we have a \( \leadsto_{K_{\infty}^+} \)-path
\[
((c_s, \epsilon), x_s) \leadsto_{K_{\infty}^+} \leadsto_{K_{\infty}^+} \leadsto_{K_{\infty}^+} ((c_q, \kappa_q'), x_q)
\]
from which we can derive a \( \leadsto_{K_{\infty}^+} \)-path
\[
((c_s, \epsilon), x_s) \leadsto_{K_{\infty}^+} \leadsto_{K_{\infty}^+} \leadsto_{K_{\infty}^+} (c_q, \kappa_q).
\]
This path should be distinct from the path (16). Let \( (c_i, \kappa_i) \) be the farthest point from the query that (16) and (18) agree. We further assume that we have chosen the path (17) such that among all the \( \leadsto_{K} \)-paths from \( (c_s, \epsilon) \) to \( (c_q, \kappa_q) \), (18) has the longest common suffix with (16). Let \( (c_{i-1}, \kappa_{i-1}) \) and \( (c'_{i-1}, \kappa'_{i-1}) \) be the first diverging point from the query such that
\[
(c_0, \kappa_0) \leadsto_{K} \leadsto_{K} (c_{i-1}, \kappa_{i-1}) \leadsto_{K} (c_{i-1}, \kappa_{i-1}) \leadsto_{K} \leadsto_{K} (c_q, \kappa_q)
\]
and
\[
(c_{i-1}, \kappa_{i-1}) \leadsto_{K} \leadsto_{K} (c_{i-1}, \kappa_{i-1}) \leadsto_{K} \leadsto_{K} (c_q, \kappa_q)
\]
where \( (c_{i-1}, \kappa_{i-1}) \neq (c'_{i-1}, \kappa'_{i-1}) \). Note that (16) and (18) agree each other at least at the query point. We now show that this is not possible.

1. When \( c_i \not\in C_c \cup C_r \):
   By the definition of \( \leadsto_{K} \), we have
   \[
   \kappa_{i-1} = \kappa_i = \kappa_{i-1}.
   \]
Thus, we should have
\[ c_{i-1} \to c_i \land c'_{i-1} \to c_i \]
where \( c_{i-1} \neq c'_{i-1} \), which basically means that \( c_i \) is a join point and \( c_{i-1} \) and \( c'_{i-1} \) exercise different branches. However, because we have considered all possible valid paths from sources in \( \Phi(c_i, x_0) \) to \( (c_i, x_0) \), we always find another path
\[ (c_s, \epsilon) \to_K \cdots \to_K (c_{i-1}, c''_{i-1}) \to_K (c_i, c_i) \to_K \cdots \to_K (c_q, c_q) \]
whenever we have path (20). Therefore, \( (c_i, c_i) \) cannot be the farthest point.

2. When \( c_i \in C_c \):
   By the definition of \( \to_K \),
   \[ c_{i-1} : \kappa_i = c_i = c'_{i-1} : \kappa_i'' \]
   It should be either \( \kappa_i = \epsilon \) and \( c_{i-1} \cdot \kappa_i = c'_{i-1} \cdot \kappa_i'' \notin K \), or \( \kappa_i = \epsilon \) and \( (c_{i-1}, \kappa_i'' - 1) = (c'_{i-1}, \kappa_i'' - 1) \). From (18) and the definition of \( K \), we have
   \[ \kappa_i = \kappa_i' \oplus \kappa_s = (c'_{i-1} \cdot \kappa_i'' - 1) \oplus \kappa_s \]
   for some \( \kappa_i' \) and \( \kappa_i'' - 1 \) such that
   \[ (c_s, \kappa_s) \to \cdots \to (c''_{i-1}, \kappa_i'' - 1) \to (c_i, \kappa_i') \to (c_q, c_q') \].
   From the Assumption 1, we have
   \[ (c''_{i-1}, \kappa_i'' - 1) \oplus \kappa_s = \epsilon. \]
   We can deduce from this \( (c_{i-1}, \kappa_i'' - 1) = (c'_{i-1}, \kappa_i'' - 1) \). Therefore, \( (c_i, c_i) \) cannot be the farthest point.

3. When \( c_i \in C_c \):
   By the definition of \( \to_K \),
   \[ \kappa_{i-1} = \text{callof}(c_i) : \kappa_i = \kappa_i'' \]
   Also, by the definition of return edge \( \to_e \), we can deduce
   \[ c_{i-1} = c'_{i-1} \] from \( c_{i-1} \to_e c_i \) and \( c_{i-1} \to_e c_i \). Therefore,
   \( (c_i, c_i) \) cannot be the farthest point.

The following lemma shows that our pre-analysis algorithm correctly estimates the behavior of the main analysis if they use the same context selector.

**Lemma 6.** Let \( K \) be an arbitrary context selector. Let \( MA_K \) be the main analysis result, i.e., a solution of (5), under the \( K \). Let \( PA_K \in C \to S^3 \) be the result of the reachability-based algorithm (Definition 3) under the \( K \). Then,
\[ \forall c \in C, \kappa \in C^2, MA_K(c, \kappa) \in \gamma(\Phi(c, \kappa)). \]

**Proof.** This lemma is proved by Lemma 1 and Lemma 2, where the proof of Lemma 1 is immediate from the abstract model transformation framework [1, 2] and we omit the proof. We prove Lemma 2 in A.2.

### A.2 Proof of Lemma 2

Let \( PA_K \) be the result of our pre-analysis under context-sensitivity \( K \). We show that \( PA_K \) is equivalent to the least such \( X(6) \) when the underlying context selector is \( K \).

To show the equivalence, we first define a new graph and use it to construct an element \( X' \in C_K \to S^3 \) based on the reachability over this graph. Then, we prove that \( X \) is the least solution of (6) (Lemmas 7 and 8) and \( PA_K \) is equivalent to \( X \) (Lemma 9).

In the below, we spell out the details of constructing \( X' \):

1. We define a context-enriched value-flow graph \( (\Omega, \to_K) \) with the node set \( \Omega = C_K \times \mathbf{Var} \) and the edge set \( \to_K \subseteq \Omega \times \Omega \) in Definition 3.

2. Let \( V \) be the set of \( \kappa, (c) \)'s reachable from \( (c, \epsilon) \):
   \[ V = \{(c, \epsilon) | (c, \epsilon) \to_K (c, \kappa)\} \]

3. We define a set \( \Omega_v \) of generators for each abstract value \( v \in V : \Omega_v = \{(v = T_v) \} \cup \{(v, x) | x \in \mathbf{Var}\} \}

4. Finally, using what we have defined so far, we construct \( X' \in D = C_K \to S^3 : \)
   \[ X(c, \kappa) = \left\{ \begin{array}{ll}
   (c, \kappa) & \text{if } (c, \kappa) \notin V \text{ then } \bot, \\
   \exists x \in \varnothing((c_0, \kappa_0), x_0) 
   \end{array} \right. \]

**Lemma 7.** The \( X \) is a solution of (6). That is,
\[ s_j^f \subseteq X'(c) \land F^3(X'(c)) \subseteq X'(c). \]

**Proof.** The first condition holds because, for all \( x, ((c, \epsilon), x) \) belongs to \( \Omega_T \) and \( (c, \epsilon) \in V \). Hence \( X'(c, \epsilon) = (Ax, \Omega_T) = \bot \).

Next we show that
\[ \forall (c, \epsilon) \in C_K, F^3(X'(c, \epsilon)) \subseteq X'(c, \epsilon). \]

Pick \( (c, \epsilon) \in C_K \). Suppose that
\[ (c, \epsilon) \notin V. \]

Then, \( X(c, \epsilon) = \bot \) by the definition of \( X \). Also, for every \( (c_0, \kappa_0) \in C_K, (c_0, \kappa_0) \to_K (c, \epsilon) \), then \( (c_0, \kappa_0) \notin V \), which implies that
\[ X(c_0, \kappa_0) = \bot. \]

Using these observations, we derive the desired relationship as follows:
\[ F^3(X'(c, \epsilon)) = \left\{ \begin{array}{cl}
   [\text{cmd}(c)](\bot) & \text{if } \text{cmd}(c) \text{ is a command of the form } x := e \text{ for some expression } e; \\
   \bot & \text{otherwise, let } w = \bot. \end{array} \right. \]

Let us now consider the case that
\[ (c, \epsilon) \in V. \]

In this case,
\[ X(c, \epsilon) = \bot. \]

If \( F^3(X'(c, \epsilon)) = \bot \), the desired relationship follows immediately from the fact that \( \bot \) is the least abstract state. Suppose
\[ F^3(X'(c, \epsilon)) \neq \bot. \]

We need to show that
\[ \forall x \in \mathbf{Var}. F^3(X'(c, \epsilon)(x)) \subseteq X'(c, \epsilon)(x). \]

Pick \( x \in \mathbf{Var} \). Let \( v = X'(c, \epsilon)(x) \). Also, define \( w = \text{const}(e) \) if \( \text{cmd}(c) \) is a command of the form \( x := e \) for some expression \( e \); otherwise, let \( w = \bot \). By the definition of \( X \),
\[ \forall v'. \forall (c_0, \kappa_0, x_0) \in \Omega_v. ((c_0, \kappa_0) \in V \land ((c_0, \kappa_0), x_0) \to_K ((c, \kappa), x)) \Longrightarrow v' \subseteq v. \]

This implies two important facts. First,
\[ w \subseteq v \]
Lemma 8. The $X$ is a lower bound for every solution of (6). That is, for every $X \in D$, 
\[ (s^I_\tau \subseteq X(\iota, \epsilon) \land F^I(X) \subseteq X) \iff X \subseteq X. \]

Proof. Consider $X \in D$ such that $s^I_\tau \subseteq X(\iota, \epsilon) \land F^I(X) \subseteq X$. We have to show that 
\[ \forall (c, \kappa) \in C_K. X(c, \kappa) \subseteq X(c, \kappa). \]

First, we show that 
\[ \forall (c, \kappa) \in V. X(c, \kappa) \neq \bot. \]  
(24)

Pick $(c, \kappa) \in V$. By the definition of $V$, 
\[ (\iota, \epsilon) \rightarrow^K (c, \kappa) \]
for some $n \geq 0$. Our proof is by induction on $n$.

- Base case: $n = 0$ in this case. Hence, $(\iota, \epsilon) = (c, \kappa)$. Since $s^I_\tau \subseteq X(\iota, \epsilon)$ by assumption, 
\[ X(\iota, \epsilon) = \top \neq \bot, \]
as desired.

- Inductive case: $n > 0$ in this case. Hence, there exists $(c_0, \kappa_0)$ such that 
\[ (\iota, \epsilon) \rightarrow^{K-1} (c_0, \kappa_0) \rightarrow^K (c, \kappa). \]
This implies that $(c_0, \kappa_0) \in V$, so by the induction hypothesis, 
\[ X(c_0, \kappa_0) \neq \bot. \]

Let $s_0 = X(c_0, \kappa_0)$. Since $F^I(X) \subseteq X$ and $(c_0, \kappa_0) \rightarrow^K (c, \kappa)$, 
\[ \llbracket \text{cmd}(c) \rrbracket(s_0) \]
\[ = \llbracket \text{cmd}(c) \rrbracket(\{ (X(c_1, \kappa_1) \mid (c_1, \kappa_1) \rightarrow^K (c, \kappa) \}) \]
\[ = F^I(X(c, \kappa)) \subseteq X(c, \kappa). \]

But $\llbracket \text{cmd}(c) \rrbracket(s') = \bot$ only if $s' = \bot$. Thus, $X(c, \kappa) \neq \bot$, as desired.

Next, using what we have just proved (i.e., (24)), we prove (23).

Pick $(c, \kappa) \in C_K$. If $(c, \kappa) \not\in V$, then $X(c, \kappa) = \bot$, so the desired inequality above follows immediately. Otherwise, 
\[ X(c, \kappa) \neq \bot \land X(c, \kappa) \neq \bot, \]
where the first disequality comes from the definition of $X$ and the second from (24). Now pick $x \in \text{Var}$. Our proof obligation is now reduced to showing 
\[ X(c, \kappa)(x) \subseteq X(c, \kappa)(x). \]

This inequality is immediate if $X(c, \kappa)(x) = \bot$. Suppose that $X(c, \kappa)(x) \neq \bot$. Let $v = X(c, \kappa)(x)$. Since $(c, \kappa) \in V$ and the domain of abstract values $V$ is totally ordered, there exist 
\[ ((c_0, \kappa_0), x_0), \ldots, ((c_n, \kappa_n), x_n) \]
such that 
\[ ((c_0, \kappa_0), x_0) \in \Omega_v \land (c_0, \kappa_0) \in V \land (\forall 0 \leq i < n. ((c_i, \kappa_i), x_i) \rightarrow^K (c_{i+1}, \kappa_{i+1}, x_{i+1})) \]
(25) 
\[ \land ((c_n, \kappa_n), x_n) = (c, \kappa, x). \]

Note that every $(c_i, \kappa_i)$ is in $V$. So, by (24), 
\[ \forall 0 \leq i \leq n. X(c_i, \kappa_i) \neq \bot. \]  
(26)

We will show that 
\[ v \subseteq X(c_0, \kappa_0)(x_0) \land (\forall 0 \leq i < n. X(c_i, \kappa_i)(x_i) \subseteq X(c_{i+1}, \kappa_{i+1})(x_{i+1})). \]  
(27)

Note that this gives the desired relationship $v \subseteq X(c, \kappa)(x)$ because of the transitivity of $\subseteq$.

The key to show the first conjunct in (27) is to notice that 
\[ ((c_0, \kappa_0) = (\iota, \epsilon) \land v = T_v) \land (\exists e. \text{cmd}(c_0) = (x_0 := \epsilon) \land \text{const}(e) = v). \]

If the first disjunct holds, we can use our assumption that 
\[ s^I_\tau \subseteq X(\iota, \epsilon) \]
and derive that 
\[ v = T_v = s^I_\tau(x_0) \subseteq X(c_0, \kappa_0)(x_0). \]
Assume that the disjunct holds. Since $X(c_0, \kappa_0) \neq \bot$, $(c_0, \kappa_0) = (\iota, \epsilon)$ or there exists some $(c_0', \kappa_0')$ such that 
\[ (c_0', \kappa_0') \rightarrow^K (c_0, \kappa_0) \land X(c_0', \kappa_0') \neq \bot. \]

Since $s^I_\tau \subseteq X(\iota, \epsilon)$ and $F^I(X) \subseteq X$, in both cases, we have that 
\[ v \subseteq X(c_0, \kappa_0)(x_0). \]

We now move on to the second conjunct of (27). In this case, we use a general fact that if 
\[ ((c', \kappa'), x') \rightarrow^K ((c'', \kappa''), x'') \land X(c', \kappa') \neq \bot, \]  
(28)
then 
\[ F^I(X(c'', \kappa'')) \neq \bot \land X(c', \kappa')(x') \subseteq F^I(X(c'', \kappa'')(x''). \]
Since $F^I(X) \subseteq X$, the second conjunct above implies that 
\[ X(c', \kappa')(x') \subseteq X(c'', \kappa'')(x''). \]
Hence, the second conjunct of (27) follows if we discharge the condition (28) for consecutive elements in the sequence 
\[ ((c_0, \kappa_0), x_0), \ldots, ((c_n, \kappa_n), x_n). \]
This condition holds because of (25) and (26).

Lemma 9. For every $c \in C$, 
\[ \text{PA}_K(c) = \bigcup_{\kappa \in C^*_K} X(c, \kappa). \]

Proof. Pick $c \in C$. Recall the definition of the set of reachable nodes $C$ in (7): 
\[ C = \{ c \mid \exists \kappa. (\iota, \epsilon) \rightarrow^*_K (c, \kappa) \}. \]
If $c \not\in C$, then 
\[ \forall \kappa \in C^*_K. (c, \kappa) \not\in V. \]
Hence, in this case,
\[ \bigcup_{\kappa \in C^*} X(c, \kappa) = \bot. \]

But \( \text{PA}_K(c) \) is also \( \bot \) by the definition of \( \text{PA}_K \).

Suppose that \( c \in C \).

Let \( K_0 = \{ \kappa \mid (c, \kappa) \in V \} \). Pick \( x \in \text{Var} \). We will show
\[ \text{PA}_K(c)(x) = \bigcup_{\kappa \in K_0} X(c, \kappa)(x). \tag{29} \]

The left hand side of this equation is the join of the set
\[ V_L = \{ v \in V \mid \exists (c_0, x_0) \in \Theta_v. \ (c_0, x_0) \hookrightarrow_K (c, x) \}. \tag{30} \]

The right hand side of the equation in (29) is the join of the set
\[ V_R = \{ v \in V \mid \exists \kappa \in K_0. \exists ((c_0, \kappa_0), x_0) \in \Omega_v. \ (c_0, \kappa_0) \in V \land ((c_0, \kappa_0), x_0) \hookrightarrow_K ((c, \kappa), x) \}. \tag{31} \]

It suffices to prove that \( V_L = V_R \). By the definitions of \( \Omega_v \) and \( \Theta_v \),
\[ (c_0, x_0) \in \Theta_v \iff ((c_0, \kappa_0), x_0) \in \Omega_v. \]

Hence,
\[ V_R = \{ v \in V \mid \exists (c_0, x_0) \in \Theta_v. \exists \kappa \in K_0. \exists \kappa_0 \in K_0. (c_0, \kappa_0) \in V \land ((c_0, \kappa_0), x_0) \hookrightarrow_K ((c, \kappa), x) \}. \]

Also, by the definitions of \( V, K_0 \) and \( \hookrightarrow_K \),
\[ (c_0, x_0) \hookrightarrow_K (c, x) \]

if and only if
\[ \exists \kappa_0. \exists \kappa \in K_0. (c_0, \kappa_0) \in V \land ((c_0, \kappa_0), x_0) \hookrightarrow_K ((c, \kappa), x). \]

Thus,
\[ V_R = \{ v \in V \mid \exists (c_0, x_0) \in \Theta_v. \ (c_0, x_0) \hookrightarrow_K (c, x) \}. \]

We have just shown that \( V_R = V_L \), as desired. \( \Box \)