Formal Aspect of Global Sparse Analysis Framework

Jeehoon Kang

December 20, 2012

Abstract

Global sparse analysis framework aims to design precise, sound, yet scalable static analyzers systematically. The framework breaks the tradeoff between general-purpose but coarse-grained sparse analyses and fine-grained but limited-purpose sparse analyses by providing a general-purpose fine-grained sparse analysis framework. However, the formal aspect of the framework has not been presented rigorously. In this paper, we rigorously prove the framework correct and formally present a fixpoint computation strategy for realistic sparse analyzers.

1 Introduction

Global sparse analysis framework, introduced by Oh et al. [14], aims to design "precise, sound, yet scalable" static analyzers systematically. The framework achieves this goal by optimizing abstract interpretation-based static analyzer [6, 3]. The essence of the optimization is qualifying abstract locations whose abstract values should be calculated for each iteration of the worklist algorithm in the course of the fixpoint computation.

Previous sparse analysis techniques [15, 19, 8, 2, 17, 7, 11, 9, 10] achieved the same goal, but in a limited manner. Their techniques are tightly coupled with particular analyses and cannot be applied to abstract interpretation-based static analyzers in general. Global sparse analysis framework addresses this by requiring only small amounts of prerequisites for baseline analyzers to satisfy.

Oh et al. [14] presented the global sparse analysis framework for C-like languages, but the proof was omitted. In the preparation of the journal version, they generalized their framework so that it is not only for C-like languages but also for functional or object-oriented languages, and handles arbitrary trace partitioning.

This paper focuses on the formal aspect of global sparse analysis framework. We rigorously prove the framework correct and formally present a fixpoint computation strategy for realistic sparse analyzers.

This paper is organized as follows. Section 2 presents the baseline analysis based on the abstract interpretation framework and the prerequisites for it. Section 3 formally presents the sparse version of the baseline analysis. Section 4 proves it correct. Section 5 formally presents a fixpoint computation strategy for realistic sparse analyzers. Section 6 concludes.

1.1 Notations

By $\wp(A)$ is meant the powerset of a set A. By A^+ is meant the set of all finite sequences in A; A^*, the set of all (possibly infinite) sequences in A. For $a \in A$, we abuse a to mean a as an element and at the same time, the one-element sequence whose the only element is a. For a finite sequence s and a sequences s' in A, by $s \cdot s'$ is meant the juxtaposition of s and s'. By $A \to B$ is meant the set of functions from A to B.

2 Baseline Analysis

In this section, we formally present the baseline analysis based on the abstract interpretation framework. In the course of the presentation, we formally provide the prerequisites for the baseline analysis for transforming it into its sparse version.

Collecting Semantics To remain as general as possible, we do not specify the target language and its semantics. Instead, a program is specified by a transition system $(\mathbb{S}, \to, \iota)$. By $\mathbb{S}$ is meant the set of all program states; $\to \subseteq \mathbb{S} \times \mathbb{S}$, the transition relation; $\iota \in \mathbb{S}$, the initial state. A sequence $\sigma \in \mathbb{S}^*$ of states is a *trace* if the first element is the initial state and all adjacent elements are related as follows:

$$\sigma_0 = \iota \text{ and } [\sigma_i \to \sigma_{i+1} \text{ for all } i].$$

We abuse $\sigma \to \sigma'$ to mean there exists $s \in \mathbb{S}$ such that $\sigma' = \sigma \cdot s$. The collecting semantics $[\![P]\!] \in \wp(\mathbb{S}^+)$ of a program P is defined by the set of traces, i.e.

$$[\![P]\!] = \{\sigma \in \mathbb{S}^+ \mid \sigma \text{ is a trace}\}.$$

We define the semantic function $F : \wp(\mathbb{S}^+) \to \wp(\mathbb{S}^+)$ by

$$F(\phi) = \{\iota\} \cup \{\sigma' \in \mathbb{S}^+ \mid \text{there exists } \sigma \in \phi \text{ such that } \sigma \to \sigma'\}.$$

Then we have

$$[\![P]\!] = \text{lfp } F.$$

Trace Partitioning By *trace partitioning* [12, 16] is meant the set Δ of *partitioning indices* and the *partition* $p : \Delta \to \wp(\mathbb{S}^+)$ such that

1. participants are mutually disjoint, i.e. $p(i) \cap p(j) = \varnothing$ for all $i \neq j$ and
2. the union of the participants is the set of all program states, i.e. $\bigcup_{i \in \Delta} p(i) = \wp(\mathbb{S}^+)$.

A trace partitioning induces a Galois connection

$$\wp(\mathbb{S}) \underset{\alpha_p}{\overset{\gamma_p}{\leftrightarrows}} \Delta \to \wp(\mathbb{S})$$

defined by

$$\alpha_p : S \mapsto [i \mapsto S \cap p(i)]$$

and

$$\gamma_p : f \mapsto \bigcup_{i \in \Delta} f(i).$$

In fact, it is a Galois insertion and does not lose information. To see this, we define the semantic function $F_p : (\Delta \to \wp(\mathbb{S}^+)) \to (\Delta \to \wp(\mathbb{S}^+))$ by

$$F_p(\phi)(i) = \alpha_p(\{\iota\})(i) \cup f_i(\bigcup_{i \Rightarrow_\phi i'} \phi(i')),$$

where $(\Rightarrow_\phi) \in \Delta \times \Delta$ is the *partitioning transition relation* defined by

$$(\Rightarrow_\phi) = \{(i, i') \in \Delta \times \Delta \mid \text{there exists } \sigma, \sigma' \text{ such that } \sigma \in \phi(i), \sigma \to \sigma', \text{ and } \sigma' \in p(i')\},$$

and $f_i : \wp(\mathbb{S}^+) \to \wp(\mathbb{S}^+)$ is the *semantic function* defined by

$$f_i(\Sigma) = \{\sigma' \in p(i) \mid \text{there exists } \sigma \in \Sigma \text{ such that } \sigma \to \sigma'\}.$$

Then we have

$$\text{lfp } F = \gamma_p(\text{lfp } F_p).$$

State Abstraction Suppose a Galois connection for program states is provided as follows:

$$\wp(\mathbb{S}) \underset{\alpha_{\mathbb{S}}}{\overset{\gamma_{\mathbb{S}}}{\leftrightarrows}} \hat{\mathbb{S}}.$$

Then we have a Galois connection

$$\Delta \to \wp(\mathbb{S}) \underset{\alpha_S}{\overset{\gamma_S}{\leftrightarrows}} \Delta \to \hat{\mathbb{S}}$$

by the component-wise construction.

Suppose there exists an *abstract transfer function* $\hat{f}_i : \hat{\mathbb{S}} \to \hat{\mathbb{S}}$ which soundly abstracts the transfer function f_i as follows:

$$\alpha_{\mathbb{S}} \circ f_i \sqsubseteq \hat{f}_i \circ \alpha_{\mathbb{S}}$$

and an *abstract transition relation* $(\hookrightarrow_{\hat{\phi}}) \in \Delta \times \Delta$ which soundly abstracts the transition relation $(\Rightarrow)$ as follows:

$$(\hookrightarrow_{\hat{\phi}}) \supseteq (\Rightarrow_{\gamma_S(\hat{\phi})}).$$

Then we have

$$\text{lfp } F_p \sqsubseteq \gamma_S(\text{lfp } \hat{F})$$

if we define the abstract semantic function $\hat{F} : (\Delta \to \hat{\mathbb{S}}) \to (\Delta \to \hat{\mathbb{S}})$ by

$$\hat{F}(\hat{\phi})(i') = (\alpha_S \circ \alpha_p)(\{\iota\})(i) \sqcup \hat{f}_{i'}(\bigsqcup_{i \hookrightarrow_{\hat{\phi}} i'} \hat{\phi}(i)).$$

Let $\alpha = \alpha_S \circ \alpha_p$ and $\gamma = \gamma_p \circ \gamma_S$. Then

$$\wp(\mathbb{S}^+) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \Delta \to \hat{\mathbb{S}}$$

is a Galois connection and

$$\text{lfp } F \sqsubseteq \gamma(\text{lfp } \hat{F}).$$

Prerequisite 1. We additionally qualify the baseline analysis to make a sparse version of it as follows:

1. $\hat{\mathbb{S}} = \hat{\mathbb{L}} \to \hat{\mathbb{V}}$ for a finite set $\hat{\mathbb{L}}$ of abstract locations and (possibly infinite) set $\hat{\mathbb{V}}$ of abstract values,
2. the abstract transfer function is monotone, i.e. for all $\hat{s}, \hat{s}' \in \hat{\mathbb{S}}$, if $\hat{s} \sqsubseteq \hat{s}'$ then $\hat{f}_i(\hat{s}) \sqsubseteq \hat{f}_i(\hat{s}')$,
3. the abstract transition relation is monotone, i.e. for all $\hat{\phi}, \hat{\phi}' \in \Delta \to \hat{\mathbb{S}}$, if $\hat{\phi} \sqsubseteq \hat{\phi}'$ then $(\hookrightarrow_{\hat{\phi}}) \subseteq (\hookrightarrow_{\hat{\phi}'})$, and
4. the abstract transition relation is component-wisely independent, i.e. for all $\hat{\phi}, \hat{\phi}' \in \Delta \to \hat{\mathbb{S}}$, $i, i' \in \Delta$, if $\hat{\phi}(i) = \hat{\phi}'(i)$ then $[i \hookrightarrow_{\hat{\phi}} i' \Leftrightarrow i \hookrightarrow_{\hat{\phi}'} i']$.

All non-relational abstract domains [6] and *packed* relational abstract domains [4, 13, 18, 1] such as octagon domains [13] and polyhedral domains [5] satisfy Prerequisite 1. In this respect, the global sparse analysis framework is sufficiently general.

The biggest difference from C-like languages to functional languages or object-oriented languages being the mutual dependency between data and control, the global sparse analysis framework can be applied not only to C-like languages but also to a broader class of languages. The dependency of control to data is addressed by the abstract transition relation $(\hookrightarrow_{\hat{\phi}})$. It depends on the data $\hat{\phi}$.

3 Global Sparse Analysis

We present a globally sparse version of a baseline analysis as presented by Oh et al. [14].

By "sparse" is meant qualifying updates of abstract values at abstract locations for each iteration of the worklist algorithm in the course of the fixpoint computation, while preserving the analysis result. As an example of the qualification, consider the program

$$⑩x := 1; \quad ⑪y := 2; \quad ⑫z := x;$$

Consider the entry for the variable x at ⑪, say $(⑪, x)$, in the fixpoint table. Semantically, if no control flows into ⑪ elsewhere, $(⑪, x)$ always equals to $(⑩, x)$ in the analysis result. Thus we may remove $(⑪, x)$ from the table and recover it from $(⑩, x)$. However, if this is the case, we have to redirect the dependency to $(⑩, x)$ because $(⑫, z)$ depended on the removed entry $(⑪, x)$.

More complex situations than simple assignments, such as pointer dereferences, pointer assignment, or non-linear control flow, make it difficult to correctly qualify updates of abstract values at abstract locations. In this respect, the technique presented by Oh et al. [14] covered a broader range of language constructs than previous ones [15, 19, 8, 2, 17, 7, 11, 9, 10].

Their technique is advantageous because it relies not on syntactic properties but on semantic properties.

Design 2. We design a sparse analysis as follows:

1. Let $\mathcal{S} = \mathrm{lfp}\ \hat{F}$.
2. Fix the *definition locations* $\hat{D} : \Delta \to \hat{\mathbb{L}}$ which includes all the abstract locations which can be actually defined in the course of the fixpoint computation, i.e. for all $i' \in \Delta$,

   $$\hat{D}(i') \supseteq D(i')$$

   where

   $$D(i') \triangleq \{\hat{l} \in \hat{\mathbb{L}} \mid \text{there exists } \hat{s} \sqsubseteq \bigsqcup_{i \hookrightarrow_{\mathcal{S}} i'} \mathcal{S}(i) \text{ such that } \alpha(\{\iota\})(\hat{l}) \sqcup \hat{s}(\hat{l}) \neq \hat{s}(\hat{l})\}.$$

3. Fix the *use locations* $\hat{U} : \Delta \to \hat{\mathbb{L}}$ which includes all the abstract locations which is actually used to generate the abstract values at definition locations (data) and the abstract transition relation (control), i.e. for all $i' \in \Delta$,

   $$\hat{U}(i') \supseteq U_d(i') \cup U_c(i')$$

   where

   $$U_d(i') \triangleq \{\hat{l} \in \hat{\mathbb{L}} \mid \text{there exists } \hat{s} \sqsubseteq \bigsqcup_{i \hookrightarrow_{\mathcal{S}} i'} \mathcal{S}(i) \text{ such that } \hat{f}_{i'}(\hat{s})|_{\hat{D}(i')} \neq \hat{f}_{i'}(\hat{s} \setminus \hat{l})|_{\hat{D}(i')}\}$$

   and

   $$U_c(i') \triangleq \{\hat{l} \in \hat{\mathbb{L}} \mid \text{there exists } \hat{s} \sqsubseteq \bigsqcup_{i \hookrightarrow_{\mathcal{S}} i'} \mathcal{S}(i) \text{ such that } (\hookrightarrow_{\hat{s}}) \neq (\hookrightarrow_{\hat{s} \setminus \hat{l}})\}.$$

4. Define the *data dependency* $i \overset{\hat{l}}{\rightsquigarrow}_{\hat{\phi}} i'$ based on definition locations and use locations. Intuitively, it means the abstract value at the abstract location $\hat{l}$ *flows* from i to i' on the

abstract transition relation of $\hat{\phi}$. By "flows" is meant $\hat{l}$ *may* be defined at i, and *must* not be defined until the control reaches to i' for use, i.e. the data dependency is defined by

$$\begin{aligned} i \overset{\hat{l}}{\sim}_{\mathcal{S}} i'' &\overset{\triangle}{\Leftrightarrow} \text{there exists } i_0, \cdots, i_n \in \Delta \text{ such that } i = i_0, i'' = i_n, i_0 \hookrightarrow_{\mathcal{S}} i_1 \hookrightarrow_{\mathcal{S}} \cdots \hookrightarrow_{\mathcal{S}} i_n, \\ &\qquad \hat{l} \in \hat{D}(i_0), \text{ and for all } k \in (0,n), \hat{l} \notin \hat{D}(i_k) \text{ and} \\ i \overset{\hat{l}}{\rightsquigarrow}_{\mathcal{S}} i'' &\overset{\triangle}{\Leftrightarrow} i \overset{\hat{l}}{\sim}_{\mathcal{S}} i'' \text{ and } \hat{l} \in \hat{U}(i_n). \end{aligned}$$

5. Define the *sparse abstract semantic function* $\hat{F}_s : (\Delta \to \hat{\mathbb{S}}) \to (\Delta \to \hat{\mathbb{S}})$ so as to consider the data dependency rather than the abstract transition relation, i.e.

$$\hat{F}_s(\hat{\phi})(i') = \alpha(\{\iota\})(i) \sqcup \hat{f}_{i'}(\bigsqcup_{i \overset{\hat{l}}{\rightsquigarrow}_{\hat{\phi}} i'} \hat{\phi}(i)|_{\hat{l}}).$$

Prerequisite 3. These are required for the use locations to be valid.

1. The abstract transfer function should depend only on the use locations, i.e. for all $i \in \Delta$ and $\hat{s} \in \hat{\mathbb{S}}$,
$$\alpha(\{\iota\})(i) \sqcup \hat{f}_i(\hat{s})|_{\hat{D}(i)} = \alpha(\{\iota\})(i) \sqcup \hat{f}(\hat{s}|_{U_d(i)})|_{\hat{D}(i)}.$$
2. The abstract transition relation should depend only on the use locations, i.e. for all $i \in \Delta$ and $\hat{s} \in \hat{\mathbb{S}}$,
$$(\hookrightarrow_{\mathcal{S}}) = (\hookrightarrow_{\mathcal{S}[i \mapsto \mathcal{S}(i)|_{U_c(i)}]}).$$

The correctness theorem means the original analysis result lfp $\hat{F}$ can be recovered easily from the sparse analysis result lfp $\hat{F}_s$.

Theorem 4 (Correctness)**.** *Let $i \in \Delta$ and $\hat{l} \in \hat{\mathbb{L}}$. If $\hat{l} \in \hat{D}(i)$ then*

$$(lfp\ \hat{F})(i)(\hat{l}) = (lfp\ \hat{F}_s)(i)(\hat{l}).$$

Otherwise,

$$(lfp\ \hat{F})(i)(\hat{l}) = \bigsqcup_{i' \overset{\hat{l}}{\sim}_{\mathcal{S}} i} \mathcal{S}(i')(\hat{l})$$

where

$$\mathcal{S} = lfp\ \hat{F}_s.$$

4 Correctness Proof

The correctness theorem essentially relates lfp $\hat{F}$ and lfp $\hat{F}_s$. There are two main differences between $\hat{F}$ and $\hat{F}_s$. First, $\hat{F}_s$ updates not through the abstract transition relation but through the data dependency. Second, $\hat{F}_s$ updates abstract values only for qualified abstract locations. To prove the correctness, we introduce the *helper abstract semantic function* $\hat{F}_h$ as an intermediate step which updates through the data dependency but does not qualify abstract locations. Formally, we define $\hat{F}_h : (\Delta \to \hat{\mathbb{S}}) \to (\Delta \to \hat{\mathbb{S}})$ by

$$\hat{F}_h(\hat{\phi})(i') = \alpha(\{\iota\})(i) \sqcup \hat{f}_{i'}(\bigsqcup_{i \overset{\hat{l}}{\rightsquigarrow}_{\hat{\phi}} i'} \hat{\phi}(i)|_{\hat{l}}).$$

Since helper abstract semantic function $\hat{F}_s$ qualifies abstract locations, lfp $\hat{F}_s$ does not equal to lfp $\hat{F}$ in general. To address their relevance, we define the *equivalence* of tables based on the equality of qualified abstract locations.

Definition 5. Let $\mathcal{S}, \mathcal{S}' : \Delta \to \hat{\mathbb{S}}$. We define the equivalence $\mathcal{S} \equiv \mathcal{S}'$ by

- for all $\hat{l} \in \hat{D}(i)$, $\mathcal{S}(i)(\hat{l}) = \mathcal{S}'(i)(\hat{l})$ and
- $(\hookrightarrow_{\mathcal{S}}) = (\hookrightarrow_{\mathcal{S}'})$.

We present the overview of the correctness proof before the details.

Proof of Theorem 4. We have

$$\begin{aligned} \text{lfp } \hat{F} &= \text{lfp } \hat{F}_h && \text{(by Lemma 6 and 7)} \\ &= \hat{F}_h(\text{lfp } \hat{F}_s). && \text{(by Lemma 11)} \end{aligned}$$

We prove by a case analysis on $\hat{l} \in \hat{D}(i)$.

- Suppose $\hat{l} \in \hat{D}(i)$. By Definition 5, it is sufficient to prove that

$$\begin{aligned} \hat{F}_h(\text{lfp } \hat{F}_s) &\equiv \hat{F}_s(\text{lfp } \hat{F}_s) && \text{(by Lemma 8, 10, and 6)} \\ &= \text{lfp } \hat{F}_s. \end{aligned}$$

- Suppose $\hat{l} \notin \hat{D}(i)$. We have

$$\begin{aligned} (\hat{F}_h(\text{lfp } \hat{F}_s))(i)(\hat{l}) &= \alpha(\{\iota\})(i)(\hat{l}) \sqcup \hat{f}_i(\bigsqcup_{i' \overset{\hat{l}'}{\sim}_{\mathcal{S}} i} \mathcal{S}(i')|_{\hat{l}'})(\hat{l}) && \text{(by definition)} \\ &= (\bigsqcup_{i' \overset{\hat{l}'}{\sim}_{\mathcal{S}} i} \mathcal{S}(i')|_{\hat{l}'})(\hat{l}) && \text{(since } \hat{l} \notin \hat{D}(i)) \\ &= \bigsqcup_{i' \overset{\hat{l}}{\sim}_{\mathcal{S}} i} \mathcal{S}(i')(\hat{l}) \end{aligned}$$

  where

$$\mathcal{S} = \text{lfp } \hat{F}_s.$$

□

Now details. The following lemmas collectively prove $\text{lfp } \hat{F} = \text{lfp } \hat{F}_h$.

Lemma 6. *lfp* $\hat{F}_h \sqsubseteq$ *lfp* $\hat{F}$.

Proof. Let $\mathcal{S} = \text{lfp } \hat{F}$. Sufficient to prove $\hat{F}_h(\mathcal{S}) \sqsubseteq \mathcal{S}$.

$$\begin{aligned} \hat{F}_h(\mathcal{S})(i) &= \alpha(\{\iota\})(i) \sqcup \hat{f}_i(\bigsqcup_{i' \overset{\hat{l}}{\sim}_{\mathcal{S}} i} \mathcal{S}(i')|_{\hat{l}}) && \text{(by definition)} \\ &\sqsubseteq \alpha(\{\iota\})(i) \sqcup \hat{f}_i(\bigsqcup_{i' \hookrightarrow_{\mathcal{S}} i} \mathcal{S}(i')) && \text{(since } \hat{f}_i \text{ is monotone and } \bigsqcup_{i' \overset{\hat{l}}{\sim}_{\mathcal{S}} i} \mathcal{S}(i')|_{\hat{l}} \sqsubseteq \bigsqcup_{i' \hookrightarrow_{\mathcal{S}} i} \mathcal{S}(i')) \\ &= \hat{F}(\mathcal{S})(i) && \text{(by definition)} \\ &= \mathcal{S}(i). && \text{(since } \mathcal{S} = \text{lfp } \hat{F}) \end{aligned}$$

To see

$$\bigsqcup_{i' \overset{\hat{l}}{\sim}_{\mathcal{S}} i} \mathcal{S}(i')|_{\hat{l}} \sqsubseteq \bigsqcup_{i' \hookrightarrow_{\mathcal{S}} i} \mathcal{S}(i'),$$

it is sufficient to prove that for all $i'' \overset{\hat{l}}{\sim}_{\mathcal{S}} i$,

$$\mathcal{S}(i'')|_{\hat{l}} \sqsubseteq \bigsqcup_{i' \hookrightarrow_{\mathcal{S}} i} \mathcal{S}(i').$$

Suppose $i'' \overset{\hat{l}}{\sim}_{\mathcal{S}} i$. By definition, there exists $i_0, \cdots, i_n \in \Delta$ such that $i'' = i_0$, $i = i_n$, $i_0 \hookrightarrow_{\mathcal{S}} i_1 \hookrightarrow_{\mathcal{S}} \cdots \hookrightarrow_{\mathcal{S}} i_n$, $\hat{l} \in \hat{D}(i_0)$, and for all $k \in (0, n)$, $\hat{l} \notin \hat{D}(i_k)$. For all $k \in (0, n)$,

$$\begin{aligned}
\mathcal{S}(i_k)(\hat{l}) &= \alpha(\{\iota\}) \sqcup \hat{f}_{i_k}(\bigsqcup_{i' \hookrightarrow_{\mathcal{S}} i_k} \mathcal{S}(i'))(\hat{l}) && \text{(by definition)} \\
&= (\bigsqcup_{i' \hookrightarrow_{\mathcal{S}} i_k} \mathcal{S}(i'))(\hat{l}) && \text{(since } \hat{l} \notin \hat{D}(i_k)\text{)} \\
&\sqsupseteq \mathcal{S}(i_{k-1})(\hat{l}).
\end{aligned}$$

Hence

$$\begin{aligned}
\mathcal{S}(i'')|_{\hat{l}} &= \mathcal{S}(i_0)|_{\hat{l}} \\
&\sqsubseteq \mathcal{S}(i_1)|_{\hat{l}} \\
&\sqsubseteq \cdots \\
&\sqsubseteq \mathcal{S}(i_{n-1})|_{\hat{l}} \\
&\sqsubseteq \bigsqcup_{i' \hookrightarrow_{\mathcal{S}} i} \mathcal{S}(i').
\end{aligned}$$

□

Lemma 7. *lfp* $\hat{F} \sqsubseteq$ *lfp* $\hat{F}_h$.

Proof. Let $\mathcal{S} = \text{lfp } \hat{F}_h$. Sufficient to prove $\hat{F}(\mathcal{S}) \sqsubseteq \mathcal{S}$.

$$\begin{aligned}
\hat{F}(\mathcal{S})(i) &= \alpha(\{\iota\})(i) \sqcup \hat{f}_i(\bigsqcup_{i' \hookrightarrow_{\mathcal{S}} i} \mathcal{S}(i')) && \text{(by definition)} \\
&\sqsubseteq \alpha(\{\iota\})(i) \sqcup \hat{f}_i(\bigsqcup_{i' \overset{\hat{l}}{\sim}_{\mathcal{S}} i} \mathcal{S}(i')|_{\hat{l}}) && (\hat{f}_i \text{ is monotone and } \bigsqcup_{i' \hookrightarrow_{\mathcal{S}} i} \mathcal{S}(i') \sqsubseteq \bigsqcup_{i' \overset{\hat{l}}{\sim}_{\mathcal{S}} i} \mathcal{S}(i')|_{\hat{l}}) \\
&= \hat{F}_h(\mathcal{S})(i) && \text{(by definition)} \\
&= \mathcal{S}(i). && \text{(since } \mathcal{S} = \text{lfp } \hat{F}_h\text{)}
\end{aligned}$$

To see

$$\bigsqcup_{i' \hookrightarrow_{\mathcal{S}} i} \mathcal{S}(i') \sqsubseteq \bigsqcup_{i' \overset{\hat{l}}{\sim}_{\mathcal{S}} i} \mathcal{S}(i')|_{\hat{l}},$$

it is sufficient to prove that for all $i' \hookrightarrow_{\mathcal{S}} i$,

$$\mathcal{S}(i') \sqsubseteq \bigsqcup_{i'' \overset{\hat{l}}{\sim}_{\mathcal{S}} i} \mathcal{S}(i'')|_{\hat{l}}.$$

We prove by a case analysis on $\hat{l} \in \hat{D}(i')$.

- Suppose $\hat{l} \in \hat{D}(i')$. We have $i' \overset{\hat{l}}{\sim}_{\mathcal{S}} i$. Hence

$$\mathcal{S}(i')(\hat{l}) \sqsubseteq \bigsqcup_{i'' \overset{\hat{l}}{\sim}_{\mathcal{S}} i} \mathcal{S}(i'')(\hat{l}).$$

- Suppose $\hat{l} \notin \hat{D}(i')$.

$$\begin{aligned}
\mathcal{S}(i')(\hat{l}) &= \hat{F}_h(\mathcal{S})(i')(\hat{l}) && \text{(since } \mathcal{S} = \text{lfp } \hat{F}_h\text{)} \\
&= \alpha(\{\iota\})(i') \sqcup \hat{f}_{i'}(\bigsqcup_{i'' \overset{\hat{l}}{\sim}_{\mathcal{S}} i'} \mathcal{S}(i'')|_{\hat{l}})(\hat{l}) && \text{(by definition)} \\
&= (\bigsqcup_{i'' \overset{\hat{l}}{\sim}_{\mathcal{S}} i'} \mathcal{S}(i'')|_{\hat{l}})(\hat{l}) && (\hat{l} \notin D(i') \text{ and Lemma 6}) \\
&= (\bigsqcup_{i'' \overset{\hat{l}}{\sim}_{\mathcal{S}} i'} \mathcal{S}(i'')(\hat{l})) \\
&\sqsubseteq (\bigsqcup_{i'' \overset{\hat{l}}{\sim}_{\mathcal{S}} i} \mathcal{S}(i'')(\hat{l})). && (i'' \overset{\hat{l}}{\sim}_{\mathcal{S}} i', i' \hookrightarrow_{\mathcal{S}} i, \text{ and } \hat{l} \notin D(i') \text{ implies } i'' \overset{\hat{l}}{\sim}_{\mathcal{S}} i)
\end{aligned}$$

□

Abstract semantic functions are related as follows:

Lemma 8. *If $\hat{\phi} \equiv \hat{\phi}'$ and $\hat{\phi}, \hat{\phi}' \sqsubseteq$ lfp $\hat{F}$ then $\hat{F}_h(\hat{\phi}) \equiv \hat{F}_s(\hat{\phi}')$.*

Proof. We prove the two conditions of Definition 5.

1. for all $\hat{l} \in \hat{D}(i)$, $\hat{F}_h(\hat{\phi})(i)(\hat{l}) = \hat{F}_s(\hat{\phi}')(i)(\hat{l})$:

$$\begin{aligned}
\hat{F}_h(\hat{\phi})(i)(\hat{l}) &= \alpha(\{\iota\})(i)(\hat{l}) \sqcup \hat{f}_i(\bigsqcup_{i' \overset{\hat{l}'}{\sim}_{\hat{\phi}} i} \hat{\phi}(i')|_{\hat{l}'})(\hat{l}) && \text{(by definition)} \\
&= \alpha(\{\iota\})(i)(\hat{l}) \sqcup \hat{f}_i(\bigsqcup_{i' \overset{\hat{l}'}{\sim}_{\hat{\phi}'} i} \hat{\phi}'(i')|_{\hat{l}'})(\hat{l}) && \text{(since } \hat{\phi} \equiv \hat{\phi}'\text{)} \\
&= \alpha(\{\iota\})(i)(\hat{l}) \sqcup \hat{f}_i(\bigsqcup_{i' \overset{\hat{l}'}{\sim}_{\hat{\phi}'} i} \hat{\phi}'(i')|_{\hat{l}' \cap \hat{U}(i)})(\hat{l}) && \text{(since } \hat{l} \in \hat{D}(i) \text{ and } \hat{\phi}' \sqsubseteq \text{lfp } \hat{F}\text{)} \\
&= \alpha(\{\iota\})(i)(\hat{l}) \sqcup \hat{f}_i(\bigsqcup_{i' \overset{\hat{l}'}{\rightsquigarrow}_{\hat{\phi}'} i} \hat{\phi}'(i')|_{\hat{l}'})(\hat{l}) && \text{(by definitions of } \sim \text{ and } \rightsquigarrow\text{)} \\
&= \hat{F}_s(\hat{\phi}')(i)(\hat{l}). && \text{(by definition)}
\end{aligned}$$

   Here we deduce

$$\begin{aligned}
&\alpha(\{\iota\})(i)(\hat{l}) \sqcup \hat{f}_i(\bigsqcup_{i' \overset{\hat{l}'}{\rightsquigarrow}_{\hat{\phi}'} i} \hat{\phi}'(i')|_{\hat{l}'})(\hat{l}) \\
= \alpha(\{\iota\})(i)(\hat{l}) &\sqcup \hat{f}_i((\bigsqcup_{i' \overset{\hat{l}'}{\rightsquigarrow}_{\hat{\phi}'} i} \hat{\phi}'(i')|_{\{\hat{l}'\}})|_{\hat{U}(i)})(\hat{l}) && \text{(by Prerequisite 3 (1))} \\
= \alpha(\{\iota\})(i)(\hat{l}) &\sqcup \hat{f}_i(\bigsqcup_{i' \overset{\hat{l}'}{\rightsquigarrow}_{\hat{\phi}'} i} \hat{\phi}'(i')|_{\{\hat{l}'\} \cap U(i)})(\hat{l}).
\end{aligned}$$

2. $(\hookrightarrow_{\hat{F}_h(\hat{\phi})}) = (\hookrightarrow_{\hat{F}_s(\hat{\phi}')})$: by Prerequisite 1 (4) and 3 (2), it is sufficient to prove for all $i \in \Delta$,

$$\hat{F}_h(\hat{\phi})(i)|_{U_c(i)} = \hat{F}_s(\hat{\phi}')(i)|_{U_c(i)}.$$

   Let $\hat{l} \in U_c(i)$. We prove by a case analysis on $\hat{l} \in \hat{D}(i)$.

- Suppose $\hat{l} \in \hat{D}(i)$. By (1), $\hat{F}_h(\hat{\phi})(i)(\hat{l}) = \hat{F}_s(\hat{\phi}')(i)(\hat{l})$.
- Suppose $\hat{l} \notin \hat{D}(i)$. We have

$$\begin{aligned}
\hat{F}(\hat{\phi})(i)(\hat{l}) &= \alpha(\{\iota\}) \sqcup \hat{f}_i(\bigsqcup_{i' \overset{\hat{l}'}{\sim}_{\hat{\phi}} i} \hat{\phi}(i')|_{\hat{l}'})(\hat{l}) && \text{(by definition)} \\
&= (\bigsqcup_{i' \overset{\hat{l}'}{\sim}_{\hat{\phi}} i} \hat{\phi}(i')|_{\hat{l}'})(\hat{l}) && \text{(since } \hat{l} \notin \hat{D}(i) \text{ and } \hat{\phi} \sqsubseteq \text{lfp } \hat{F}) \\
&= \bigsqcup_{i' \overset{\hat{l}}{\sim}_{\hat{\phi}} i} \hat{\phi}(i')(\hat{l}) \\
&= \bigsqcup_{i' \overset{\hat{l}}{\sim}_{\hat{\phi}'} i} \hat{\phi}'(i')(\hat{l}) && \text{(since } \hat{l} \in \hat{D}(i') \text{ and } \hat{\phi} \equiv \hat{\phi}') \\
&= \bigsqcup_{i' \overset{\hat{l}}{\rightsquigarrow}_{\hat{\phi}} i} \hat{\phi}'(i')(\hat{l}) && \text{(since } \hat{l} \in U_c(i) \subseteq \hat{U}(i)) \\
&= \alpha(\{\iota\}) \sqcup \hat{f}_i(\bigsqcup_{i' \overset{\hat{l}'}{\rightsquigarrow}_{\hat{\phi}'} i} \hat{\phi}'(i')|_{\hat{l}'})(\hat{l}) && \text{(since } \hat{l} \notin \hat{D}(i) \text{ and } \hat{\phi}' \sqsubseteq \text{lfp } \hat{F}) \\
&= \hat{F}_s(\hat{\phi}')(i)(\hat{l}). && \text{(by definition)}
\end{aligned}$$

□

Lemma 9. *If $\hat{\phi} \equiv \hat{\phi}'$ then $\hat{F}_h(\hat{\phi}) = \hat{F}_h(\hat{\phi}')$.*

Proof. For all $\iota \in \Delta$ and $\hat{l} \in \hat{\mathbb{L}}$,

$$\begin{aligned}
\hat{F}_h(\hat{\phi})(i) &= \alpha(\{\iota\}) \sqcup \hat{f}_i(\bigsqcup_{i' \overset{\hat{l}}{\sim}_{\hat{\phi}} i} \hat{\phi}(i')|_{\hat{l}}) && \text{(by definition)} \\
&= \alpha(\{\iota\}) \sqcup \hat{f}_i(\bigsqcup_{i' \overset{\hat{l}}{\sim}_{\hat{\phi}'} i} \hat{\phi}'(i')|_{\hat{l}}) && \text{(since } \hat{\phi} \equiv \hat{\phi}') \\
&= \hat{F}_h(\hat{\phi}')(i). && \text{(by definition)}
\end{aligned}$$

□

Helper abstract semantic function $\hat{F}_h$ is greater than or equal to sparse abstract semantic function $\hat{F}_s$.

Lemma 10. *lfp $\hat{F}_s \sqsubseteq$ lfp $\hat{F}_h$.*

Proof. Immediate from $(\rightsquigarrow_{\hat{\phi}}) \subseteq (\sim_{\hat{\phi}})$. □

Lemma 11. *$\hat{F}_h$(lfp $\hat{F}_s$) = lfp $\hat{F}_s$.*

Proof. We prove $\hat{F}_h(\text{lfp } \hat{F}_s) \sqsubseteq \text{lfp } \hat{F}_s$ and $\text{lfp } \hat{F}_s \sqsubseteq \hat{F}_h(\text{lfp } \hat{F}_s)$.

- $\hat{F}_h(\text{lfp } \hat{F}_s) \sqsubseteq \text{lfp } \hat{F}_s$: by Lemma 10, $\text{lfp } \hat{F}_s \sqsubseteq \text{lfp } \hat{F}_h$. By applying a monotone function $\hat{F}_h$, we are done.

- lfp $\hat{F}_s \sqsubseteq \hat{F}_h(\text{lfp } \hat{F}_s)$: it is sufficient to prove that

$$\hat{F}_h^2(\text{lfp } \hat{F}_s) \sqsubseteq \hat{F}_h(\text{lfp } \hat{F}_s).$$

$$\begin{aligned}
&\text{lfp } \hat{F}_s = \text{lfp } \hat{F}_s \\
&\Rightarrow \hat{F}_h(\text{lfp } \hat{F}_s) \equiv \hat{F}_s(\text{lfp } \hat{F}_s) && \text{(by Lemma 8, 10, and 6)} \\
&\Rightarrow \hat{F}_h(\text{lfp } \hat{F}_s) \equiv \text{lfp } \hat{F}_s \\
&\Rightarrow \hat{F}_h^2(\text{lfp } \hat{F}_s) = \hat{F}_h(\text{lfp } \hat{F}_s). && \text{(by Lemma 9)}
\end{aligned}$$

□

All required lemmas are proved.

5 Fixpoint Computation Strategy

Sparse abstract semantic function $\hat{F}_s$ computes the abstract transition relation ($\hookrightarrow_{\hat{\phi}}$) (and $\overset{\hat{l}}{\rightsquigarrow}_{\hat{\phi}}$ as well) based on its argument $\hat{\phi}$. However, drawing the abstract transition relation is expensive for functional or object-oriented languages. Thus it is wise to avoid drawing the transition relation every time. For this, we repeat the following procedures until we reach an actual fixpoint:

1. Compute the fixpoint as if the abstract transition relation is fixed.
2. Draw the abstract transition relation based on the computed fixpoint.

Formally, we define $\hat{F}_m : (\Delta \to \hat{\mathbb{S}}) \to (\Delta \to \hat{\mathbb{S}})$ by

$$\hat{F}_m(\hat{X}) \triangleq \text{lfp } \lambda \hat{Y}.\hat{F}_l(\hat{X}, \hat{X} \sqcup \hat{Y})$$

where we define $\hat{F}_l : (\Delta \to \hat{\mathbb{S}}) \times (\Delta \to \hat{\mathbb{S}}) \to (\Delta \to \hat{\mathbb{S}})$ by

$$\hat{F}_l(\hat{X}, \hat{Y})(i) \triangleq \alpha(\{\iota\})(i) \sqcup \hat{f}_i(\bigsqcup_{i' \overset{\hat{l}}{\rightsquigarrow}_{\hat{X}} i} \hat{Y}(i')|_{\hat{l}}).$$

Note that the first argument of $\hat{F}_l$ is for the abstract transition relation (control) and the second is for abstract values (data). Function $\hat{F}_m$ fixes the first argument (as if the abstract transition relation is fixed) and computes the fixpoint.

The fixpoint computation strategy $\hat{F}_m$ is correct with respect to $\hat{F}_s$ as follows:

Theorem 12 (Correctness of Strategy)**.** *lfp* $\hat{F}_m$ = *lfp* $\hat{F}_s$.

Proof. We prove lfp $\hat{F}_m \sqsubseteq$ lfp $\hat{F}_s$ and lfp $\hat{F}_s \sqsubseteq$ lfp $\hat{F}_m$. Note that $\hat{F}_m$ and $\hat{F}_s$ are monotone in their arguments and $\hat{F}_s(\hat{X}) = \hat{F}_l(\hat{X}, \hat{X})$ for all $\hat{X}$.

- lfp $\hat{F}_m \sqsubseteq$ lfp $\hat{F}_s$:
  Sufficient to prove $\hat{F}_m(\text{lfp } \hat{F}_s) \sqsubseteq \text{lfp } \hat{F}_s$.

$$\begin{aligned}
\hat{F}_m(\text{lfp } \hat{F}_s) &= \text{lfp } \lambda \hat{Y}.\hat{F}_l(\text{lfp } \hat{F}_s, \text{lfp } \hat{F}_s \sqcup \hat{Y}) \\
&\sqsubseteq \text{lfp } \hat{F}_s. && \text{(since } \hat{F}_l(\text{lfp } \hat{F}_s, \text{lfp } \hat{F}_s \sqcup \text{lfp } \hat{F}_s) = \text{lfp } \hat{F}_s)
\end{aligned}$$

- $\text{lfp}\ \hat{F}_s \sqsubseteq \text{lfp}\ \hat{F}_m$:
  Sufficient to prove $\hat{F}_s(\text{lfp}\ \hat{F}_m) \sqsubseteq \text{lfp}\ \hat{F}_m$.

$$\begin{aligned}\text{lfp}\ \hat{F}_m &= \hat{F}_m(\text{lfp}\ \hat{F}_m)\\ &= \text{lfp}\ \lambda\hat{Y}.\hat{F}_l(\text{lfp}\ \hat{F}_m, \text{lfp}\ \hat{F}_m \sqcup \hat{Y}).\end{aligned}$$

  Hence

$$\begin{aligned}\text{lfp}\ \hat{F}_m &= \hat{F}_l(\text{lfp}\ \hat{F}_m, \text{lfp}\ \hat{F}_m \sqcup \text{lfp}\ \hat{F}_m)\\ &= \hat{F}_s(\text{lfp}\ \hat{F}_m).\end{aligned}$$

□

6 Conclusion

The global sparse analysis framework, presented by Oh et al. [14], aims to design "precise, sound, yet scalable" abstract interpretation-based static analyzers systematically. It requires only small amounts of prerequisites (Prerequisites 1 and 3) for the baseline analysis. It provides a well-defined design procedure (Design 2) to make a sparse version. It broke the tradeoff between general-purpose but coarse-grained sparse analyses and fine-grained but limited-purpose sparse analyses by providing a general-purpose fine-grained sparse analysis framework. However, its correctness has not been proved rigorously.

This paper focuses on the formal aspect of global sparse analysis framework. We rigorously prove the framework correct and formally present a fixpoint computation strategy for realistic sparse analyzers.

Bibliography

[1] B. Blanchet, P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival. A static analyzer for large safety-critical software. In *Proceedings of the ACM SIGPLAN-SIGACT Conference on Programming Language Design and Implementation*, pages 196–207, 2003.

[2] David R. Chase, Mark Wegman, and F. Kenneth Zadeck. Analysis of pointers and structures. In *Proceedings of the ACM SIGPLAN conference on Programming language design and implementation*, pages 296–310, 1990.

[3] P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *Conference Record of the Sixth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 269–282, San Antonio, Texas, 1979. ACM Press, New York, NY.

[4] P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, and X. Rival. Why does astrée scale up? *Formal Methods in System Design*, 35(3):229–264, December 2009.

[5] P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In *Conference Record of the Fifth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 84–97, Tucson, Arizona, 1978. ACM Press, New York, NY.

[6] Patrick Cousot and Radhia Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of The ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 238–252, 1977.

[7] Ron K. Cytron and Jeanne Ferrante. Efficiently computing ϕ-nodes on-the-fly. *ACM Trans on Programming Languages and Systems*, 17:487–506, May 1995.

[8] Dhananjay M. Dhamdhere, Barry K. Rosen, and F. Kenneth Zadeck. How to analyze large programs efficiently and informatively. In *Proceedings of the ACM SIGPLAN conference on Programming language design and implementation*, PLDI '92, pages 212–223, New York, NY, USA, 1992. ACM.

[9] Ben Hardekopf and Calvin Lin. Semi-sparse flow-sensitive pointer analysis. In *Proceedings of The ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 226–238, 2009.

[10] Ben Hardekopf and Calvin Lin. Flow-sensitive pointer analysis for millions of lines of code. In *Proceedings of the 9th Annual IEEE/ACM International Symposium on Code Generation and Optimization*, pages 289–298, 2011.

[11] Richard Johnson and Keshav Pingali. Dependence-based program analysis. In *Proceedings of the ACM SIGPLAN conference on Programming language design and implementation*, pages 78–89, 1993.

[12] Laurent Mauborgne and Xavier Rival. Trace partitioning in abstract interpretation based static analyzers. In M. Sagiv, editor, *Proceedings of European Symposium on Programming*, volume 3444 of *Lecture Notes in Computer Science*, pages 5–20. Springer-Verlag, 2005.

[13] A. Miné. The Octagon Abstract Domain. *Higher-Order and Symbolic Computation*, 19(1):31–100, 2006.

[14] Hakjoo Oh, Kihong Heo, Wonchan Lee, Woosuk Lee, and Kwangkeun Yi. Design and implementation of sparse global analyses for C-like languages. In *Proceedings of The ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2012.

[15] John H. Reif and Harry R. Lewis. Symbolic evaluation and the global value graph. In *Proceedings of the 4th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 104–118, 1977.

[16] Xavier Rival and Laurent Mauborgne. The trace partitioning abstract domain. *ACM Trans on Programming Languages and System*, 29(5):26–51, 2007.

[17] Teck Bok Tok, Samuel Z. Guyer, and Calvin Lin. Efficient flow-sensitive interprocedural data-flow analysis in the presence of pointers. In *Proceedings of the International Conference on Compiler Construction*, pages 17–31, 2006.

[18] Arnaud Venet and Guillaume Brat. Precise and efficient static array bound checking for large embedded c programs. In *Proceedings of the ACM SIGPLAN 2004 conference on Programming language design and implementation*, PLDI '04, pages 231–242, New York, NY, USA, 2004. ACM.

[19] Mark N. Wegman and F. Kenneth Zadeck. Constant propagation with conditional branches. *ACM Trans on Programming Languages and Systems*, 13:181–210, April 1991.