

AAA616: Program Analysis

Abstract Interpretation Framework

Hakjoo Oh
2019 Fall

Abstract Interpretation Framework

A powerful framework for designing correct static analysis

- “framework”: correct static analysis comes out, reusable
- “powerful”: all static analyses are understood in this framework
- “simple”: prescription is simple
- “eye-opening”: any static analysis is an abstract interpretation

ABSTRACT INTERPRETATION: A DEVEDIC LECTURE NOTE FOR STUDENT ANALYSIS OF PROGRAMS BY COMPRESSION OR APPROXIMATION OF FEATURES

Heejoon Kim* and **Heehun Kim†**
 Laboratory of Computer Science, KAIST, Daejeon, 305-385, Korea
 heejoon@cs.kaist.ac.kr, hkim@cs.kaist.ac.kr

1. Introduction

Program abstract interpretation is one of the most abstract and powerful methods for analyzing the behavior of programs. It is a technique for analyzing the behavior of programs by approximating the values of variables and expressions in the program. The abstract interpretation framework provides a systematic way to design and implement abstract interpreters for various programming languages. In this paper, we present a framework for designing correct static analyses based on abstract interpretation. The framework is simple and powerful, and it can be used to design a wide range of static analyses. The framework consists of a concrete interpreter, an abstract interpreter, and a transfer function. The concrete interpreter is a standard interpreter for the target programming language. The abstract interpreter is a simplified version of the concrete interpreter, where the values of variables and expressions are represented by abstract values. The transfer function maps concrete values to abstract values. The abstract interpreter uses the transfer function to compute the abstract values of expressions and statements. The abstract interpreter is designed to be sound, meaning that it never reports a false alarm. The framework is implemented in the C++ programming language. The framework is available as a library for other researchers. The framework is also available as a set of lecture notes for students. The framework is a good starting point for designing correct static analyses. The framework is simple and powerful, and it can be used to design a wide range of static analyses. The framework consists of a concrete interpreter, an abstract interpreter, and a transfer function. The concrete interpreter is a standard interpreter for the target programming language. The abstract interpreter is a simplified version of the concrete interpreter, where the values of variables and expressions are represented by abstract values. The transfer function maps concrete values to abstract values. The abstract interpreter uses the transfer function to compute the abstract values of expressions and statements. The abstract interpreter is designed to be sound, meaning that it never reports a false alarm. The framework is implemented in the C++ programming language. The framework is available as a library for other researchers. The framework is also available as a set of lecture notes for students. The framework is a good starting point for designing correct static analyses.

FORMALIZING DESIGN OF PROGRAM ANALYSIS FRAMEWORKS

Heejoon Kim* and **Heehun Kim†**
 Laboratory of Computer Science, KAIST, Daejeon, 305-385, Korea

1. Introduction and Summary

Formalizing the design of program analysis frameworks is a challenging task. The design of a program analysis framework involves many decisions, and these decisions can have a significant impact on the performance and reliability of the framework. In this paper, we present a formalized design of a program analysis framework. The design is based on the abstract interpretation framework. The design is simple and powerful, and it can be used to design a wide range of program analysis frameworks. The design consists of a concrete interpreter, an abstract interpreter, and a transfer function. The concrete interpreter is a standard interpreter for the target programming language. The abstract interpreter is a simplified version of the concrete interpreter, where the values of variables and expressions are represented by abstract values. The transfer function maps concrete values to abstract values. The abstract interpreter uses the transfer function to compute the abstract values of expressions and statements. The abstract interpreter is designed to be sound, meaning that it never reports a false alarm. The design is implemented in the C++ programming language. The design is available as a library for other researchers. The design is also available as a set of lecture notes for students. The design is a good starting point for designing correct static analyses.

CC77

CC79

Step 1: Define Concrete Semantics

The concrete semantics describes the real executions of the program.
Described by semantic domain and function.

- A *semantic domain* D , which is a CPO:
 - ▶ D is a partially ordered set with a least element \perp .
 - ▶ Any increasing chain $d_0 \sqsubseteq d_1 \sqsubseteq \dots$ in D has a least upper bound $\bigsqcup_{n \geq 0} d_n$ in D .
- A *semantic function* $F : D \rightarrow D$, which is continuous: for all chains $d_0 \sqsubseteq d_1 \sqsubseteq \dots$,

$$F\left(\bigsqcup_{n \geq 0} d_i\right) = \bigsqcup_{n \geq 0} F(d_n).$$

Then, the concrete semantics (or collecting semantics) is defined as the least fixed point of *semantic function* $F : D \rightarrow D$:

$$\mathit{fix} F = \bigsqcup_{i \in \mathbb{N}} F^i(\perp).$$

Step 2: Define Abstract Semantics

Define the abstract semantics of the input program.

- Define an *abstract semantic domain* CPO \hat{D} .
 - ▶ Intuition: \hat{D} is an abstraction of D
- Define an *abstract semantic function* $\hat{F} : \hat{D} \rightarrow \hat{D}$.
 - ▶ Intuition: \hat{F} is an abstraction of F .
 - ▶ \hat{F} must be monotone:

$$\forall \hat{x}, \hat{y} \in \hat{D}. \hat{x} \sqsubseteq \hat{y} \implies \hat{F}(\hat{x}) \sqsubseteq \hat{F}(\hat{y})$$

(or extensive: $\forall x \in \hat{D}. x \sqsubseteq \hat{F}(x)$)

Then, static analysis is to compute an upper bound of:

$$\bigsqcup_{i \in \mathbb{N}} \hat{F}^i(\perp)$$

How can we ensure that the result soundly approximate the concrete semantics?

Requirement 1: Galois Connection

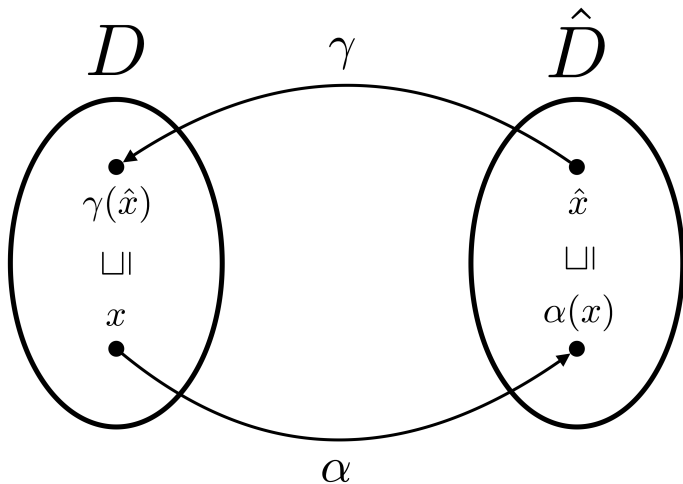
D and \hat{D} must be related with Galois-connection:

$$D \xrightleftharpoons[\alpha]{\gamma} \hat{D}$$

That is, we have

- *abstraction function*: $\alpha \in D \rightarrow \hat{D}$
 - ▶ represents elements in D as elements of \hat{D}
- *concretization function*: $\gamma \in \hat{D} \rightarrow D$
 - ▶ gives the meaning of elements of \hat{D} in terms of D
- $\forall x \in D, \hat{x} \in \hat{D}. \alpha(x) \sqsubseteq \hat{x} \iff x \sqsubseteq \gamma(\hat{x})$
 - ▶ α and γ respect the orderings of D and \hat{D}
 - ▶ If an element $x \in D$ is safely described by $\hat{x} \in \hat{D}$, i.e., $\alpha(d) \sqsubseteq \hat{d}$, then the element described by \hat{x} is also safe w.r.t. x , i.e., $x \sqsubseteq \gamma(\hat{x})$

Galois-Connection



Example: Sign Abstraction

$$\wp(\mathbb{Z}) \xrightleftharpoons[\alpha]{\gamma} (\{\perp, +, \mathbf{0}, -, \top\}, \sqsubseteq)$$

$$\alpha(Z) = \begin{cases} \perp & Z = \emptyset \\ + & \forall z \in Z. z > 0 \\ \mathbf{0} & Z = \{0\} \\ - & \forall z \in Z. z < 0 \\ \top & \text{otherwise} \end{cases}$$

$$\gamma(\perp) = \emptyset$$

$$\gamma(\top) = \mathbb{Z}$$

$$\gamma(+)= \{z \in \mathbb{Z} \mid z > 0\}$$

$$\gamma(\mathbf{0}) = \{0\}$$

$$\gamma(-) = \{z \in \mathbb{Z} \mid z < 0\}$$

Example: Interval Abstraction

$$\wp(\mathbb{Z}) \xrightleftharpoons[\alpha]{\gamma} \{\perp\} \cup \{[a, b] \mid a \in \mathbb{Z} \cup \{-\infty\}, b \in \mathbb{Z} \cup \{+\infty\}\}$$

$$\begin{aligned}\gamma(\perp) &= \emptyset \\ \gamma([a, b]) &= \{z \in \mathbb{Z} \mid a \leq z \leq b\}\end{aligned}$$

$$\begin{aligned}\alpha(\emptyset) &= \perp \\ \alpha(X) &= [\min X, \max X]\end{aligned}$$

cf) Alternate Formulation

D and \hat{D} are related with Galois-connection:

$$D \xrightleftharpoons[\alpha]{\gamma} \hat{D}$$

iff (α, γ) satisfies the following conditions:

- α and γ are monotone functions
- $\gamma \circ \alpha$ is extensive, i.e., $\gamma \circ \alpha \sqsupseteq \lambda x.x$
 - ▶ abstraction typically loses precision
 - ▶ $(\gamma \circ \alpha)(\{1, 3\}) = \{1, 2, 3\}$
- $\alpha \circ \gamma$ is reductive: i.e., $\alpha \circ \gamma \sqsubseteq \lambda x.x$
 - ▶ If $\alpha \circ \gamma = \lambda x.x$, Galois-insertion.
 - ▶ With Galois-insertion, no two abstract elements describe the same concrete element, which may be true with Galois-connection.

Proof (\Rightarrow)

If we have a Galois-connection:

$$\forall x \in D, \hat{x} \in \hat{D}. \alpha(x) \sqsubseteq \hat{x} \iff x \sqsubseteq \gamma(\hat{x})$$

then

- $\lambda x.x \sqsubseteq \gamma \circ \alpha$: $\alpha(x) \sqsubseteq \alpha(x)$ and hence $x \sqsubseteq \gamma(\alpha(x))$ by Galois-connection.
- $\alpha \circ \gamma \sqsubseteq \lambda x.x$: $\gamma(\hat{x}) \sqsubseteq \gamma(\hat{x})$ and hence $\alpha(\gamma(\hat{x})) \sqsubseteq \hat{x}$ by Galois-connection.
- γ is monotone: if $\hat{x} \sqsubseteq \hat{y}$, then $\alpha(\gamma(\hat{x})) \sqsubseteq \hat{y}$. Hence $\gamma(\hat{x}) \sqsubseteq \gamma(\hat{y})$ by Galois-connection.
- α is monotone: if $x \sqsubseteq y$, then $x \sqsubseteq \gamma(\alpha(y))$. Hence $\alpha(x) \sqsubseteq \alpha(y)$ by Galois-connection.

Proof (\Leftarrow)

- Assume $\alpha(x) \sqsubseteq \hat{x}$. Since γ is monotone, $\gamma(\alpha(x)) \sqsubseteq \gamma(\hat{x})$.
Because $\gamma \circ \alpha$ is extensive, we have $x \sqsubseteq \gamma(\hat{x})$.
- Assume $x \sqsubseteq \gamma(\hat{x})$. Since α is monotone, $\alpha(x) \sqsubseteq \alpha(\gamma(\hat{x}))$.
Because $\alpha \circ \gamma$ is reductive, we have $\alpha(x) \sqsubseteq \hat{x}$.

Properties of Galois-Connection

Given $D \xleftrightarrow[\alpha]{\gamma} \hat{D}$, we have:

- $\gamma \circ \alpha \circ \gamma = \gamma$
 - ▶ From $\alpha \circ \gamma \sqsubseteq \lambda x.x$ and monotonicity of γ , we have $\gamma \circ \alpha \circ \gamma \sqsubseteq \gamma$. We have $\gamma \circ \alpha \circ \gamma \sqsupseteq \gamma$ from $\gamma \circ \alpha \sqsupseteq \lambda x.x$.
- $\alpha \circ \gamma \circ \alpha = \alpha$
- $\alpha \circ \gamma$ and $\gamma \circ \alpha$ are idempotent:

$$(\alpha \circ \gamma)^2 = \alpha \circ \gamma, (\gamma \circ \alpha)^2 = \gamma \circ \alpha$$

- γ uniquely determines α (D, \hat{D} complete lattices):

$$\alpha(d) = \bigsqcap \{\hat{d} \mid d \sqsubseteq \gamma(\hat{d})\}$$

which implies that $\alpha(d)$ is the best abstraction of d .

- α uniquely determines γ :

$$\gamma(\hat{d}) = \bigsqcup \{d \mid \alpha(d) \sqsubseteq \hat{d}\}$$

Deriving Galois-Connections

- Pointwise lifting: Given $D \xleftrightarrow[\alpha]{\gamma} \hat{D}$ and a set S , then

$$S \rightarrow D \xleftrightarrow[\alpha']{\gamma'} S \rightarrow \hat{D}$$

with $\alpha'(f) = \lambda s \in S. \alpha(f(s))$ and $\gamma(f) = \lambda s \in S. \gamma(f(s))$.

- Composition: Given $X_1 \xleftrightarrow[\alpha_1]{\gamma_1} X_2 \xleftrightarrow[\alpha_2]{\gamma_2} X_3$, we have

$$X_1 \xleftrightarrow[\alpha_2 \circ \alpha_1]{\gamma_1 \circ \gamma_2} X_3$$

Requirement 2: \hat{F} and F

- \hat{F} is a sound abstraction of F :

$$F \circ \gamma \sqsubseteq \gamma \circ \hat{F} \quad (\alpha \circ F \sqsubseteq \hat{F} \circ \alpha)$$

- or, alternatively,

$$\alpha(x) \sqsubseteq \hat{x} \implies \alpha(F(x)) \sqsubseteq \hat{F}(\hat{x})$$

Best Abstract Semantics

From $D \xleftrightarrow[\alpha]{\gamma} \hat{D}$ and $F \circ \gamma \sqsubseteq \gamma \circ \hat{F}$, we have

$$\begin{aligned} \alpha \circ F \circ \gamma &\sqsubseteq \alpha \circ \gamma \circ \hat{F} && \alpha \text{ is monotone} \\ &\sqsubseteq \hat{F} && \alpha \circ \gamma \sqsubseteq \lambda x.x \end{aligned}$$

The result means that $\alpha \circ F \circ \gamma$ is the best abstraction of F and any sound abstraction \hat{F} of F is greater than $\alpha \circ F \circ \gamma$.

Composition

When F, F' are concrete operators and \hat{F}, \hat{F}' are abstract operators, if \hat{F} and \hat{F}' are sound abstractions of F and F' , respectively, then $\hat{F} \circ \hat{F}'$ is a sound abstraction of $F \circ F'$.

Fixpoint Transfer Theorems

Theorem (Fixpoint Transfer)

Let D and \hat{D} be related by Galois-connection $D \stackrel{\gamma}{\longleftarrow} \hat{D}$. Let $F : D \rightarrow D$ be a continuous function and $\hat{F} : \hat{D} \rightarrow \hat{D}$ be a monotone function such that $\alpha \circ F \sqsubseteq \hat{F} \circ \alpha$. Then,

$$\alpha(\text{fix } F) \sqsubseteq \bigsqcup_{i \in \mathbb{N}} \hat{F}^i(\hat{\perp}).$$

Theorem (Fixpoint Transfer2)

Let D and \hat{D} be related by Galois-connection $D \stackrel{\gamma}{\longleftarrow} \hat{D}$. Let $F : D \rightarrow D$ be a continuous function and $\hat{F} : \hat{D} \rightarrow \hat{D}$ be a monotone function such that $\alpha(x) \sqsubseteq \hat{x} \implies \alpha(F(x)) \sqsubseteq \hat{F}(\hat{x})$. Then,

$$\alpha(\text{fix } F) \sqsubseteq \bigsqcup_{i \in \mathbb{N}} \hat{F}^i(\hat{\perp}).$$

Computing $\bigsqcup_{i \in \mathbb{N}} \hat{F}^i(\hat{\perp})$

- If the abstract domain \hat{D} has finite height (i.e., all chains are finite), we can directly calculate

$$\bigsqcup_{i \in \mathbb{N}} \hat{F}^i(\hat{\perp}).$$

- If the domain \hat{D} has infinite height, the computation may not terminate. In this case, we find a finite chain $\hat{X}_0 \sqsubseteq \hat{X}_1 \sqsubseteq \hat{X}_2 \sqsubseteq \dots$ such that

$$\bigsqcup_{i \in \mathbb{N}} \hat{F}^i(\hat{\perp}) \sqsubseteq \lim_{i \in \mathbb{N}} \hat{X}_i$$

Fixpoint Accerlation with Widening

Define finite chain \hat{X}_i by an widening operator $\nabla : \hat{D} \times \hat{D} \rightarrow \hat{D}$:

$$\begin{aligned}\hat{X}_0 &= \perp \\ \hat{X}_i &= \hat{X}_{i-1} && \text{if } \hat{F}(\hat{X}_{i-1}) \sqsubseteq \hat{X}_{i-1} \\ &= \hat{X}_{i-1} \nabla \hat{F}(\hat{X}_{i-1}) && \text{otherwise}\end{aligned} \quad (1)$$

Conditions on ∇ :

- $\forall a, b \in \hat{D}. (a \sqsubseteq a \nabla b) \wedge (b \sqsubseteq a \nabla b)$
- For all increasing chains $(x_i)_i$, the increasing chain $(y_i)_i$ defined as

$$y_i = \begin{cases} x_0 & \text{if } i = 0 \\ y_{i-1} \nabla x_i & \text{if } i > 0 \end{cases}$$

eventually stabilizes (i.e., the chain is finite).

Decreasing Iterations with Narrowing

- We can refine the widening result $\lim_{i \in \mathbb{N}} \hat{X}_i$ by a narrowing operator $\Delta : \hat{D} \times \hat{D} \rightarrow \hat{D}$.
- Compute chain $(\hat{Y}_i)_i$

$$\hat{Y}_i = \begin{cases} \lim_{i \in \mathbb{N}} \hat{X}_i & \text{if } i = 0 \\ \hat{Y}_{i-1} \Delta \hat{F}(\hat{Y}_{i-1}) & \text{if } i > 0 \end{cases} \quad (2)$$

- Conditions on Δ
 - ▶ $\forall a, b \in \hat{D}. a \sqsubseteq b \implies a \sqsubseteq a \Delta b \sqsubseteq b$
 - ▶ For all decreasing chain $(x_i)_i$, the decreasing chain $(y_i)_i$ defined as

$$y_i = \begin{cases} x_i & \text{if } i = 0 \\ y_{i-1} \Delta x_i & \text{if } i > 0 \end{cases}$$

eventually stabilizes.

Safety of Widening and Narrowing

Theorem (Widening's Safety)

Let \hat{D} be a CPO, $\hat{F} : \hat{D} \rightarrow \hat{D}$ a monotone function, $\nabla : \hat{D} \times \hat{D} \rightarrow \hat{D}$ a widening operator. Then, chain $(\hat{X}_i)_i$ defined as (1) eventually stabilizes and

$$\bigsqcup_{i \in \mathbb{N}} \hat{F}^i(\hat{\perp}) \sqsubseteq \lim_{i \in \mathbb{N}} \hat{X}_i.$$

Theorem (Narrowing's Safety)

Let \hat{D} be a CPO, $\hat{F} : \hat{D} \rightarrow \hat{D}$ a monotone function, $\Delta : \hat{D} \times \hat{D} \rightarrow \hat{D}$ a narrowing operator. Then, chain $(\hat{Y}_i)_i$ defined as (2) eventually stabilizes and

$$\bigsqcup_{i \in \mathbb{N}} \hat{F}^i(\hat{\perp}) \sqsubseteq \lim_{i \in \mathbb{N}} \hat{Y}_i.$$