Abstract Interpretation

William Schultz

August 25, 2022

If we want to analyze the behavior of a program, we typically perform some kind of *abstraction*. That is, we approximate the concrete semantics of the program in some way that is sufficient for analysis. *Abstract interpretation* provides a formal framework for defining and performing these types of program abstractions.

Background

A *lattice* $L = (S, \sqsubseteq)$ is a partially ordered set where each pair of elements has a least upper bound (i.e. *join* $\sqcup$) and a greatest lower bound (i.e. *meet*, $\sqcap$).

Abstraction Domains

An *abstraction domain* is defined as follows, where D is a *concrete domain* and $\hat{D}$ is an *abstract domain*, and elements of $\hat{D}$ form a lattice:

- $\gamma : \hat{D} \rightarrow D$: **Concretization function** that maps abstract values to sets of concrete elements.
- $\alpha : D \rightarrow \hat{D}$: **Abstraction function** that maps sets of concrete elements to the most precise value in the abstract domain.

where α and γ must form a *Galois connection*. That is, they satisfy the following condition

$$\forall x \in D, \forall \hat{x} \in \hat{D} : \alpha(x) \sqsubseteq \hat{x} \Leftrightarrow x \sqsubseteq \gamma(\hat{x})$$

Intuitively, this means that the abstraction and concretization functions respect the orderings of D and $\hat{D}$. That is, if $\alpha(x)$, the abstraction of x, is ordered before some other $\hat{x} \in \hat{D}$, then x should be ordered before $\gamma(\hat{x})$, the concretization of $\hat{x}$.