Abstract Interpretation and Astrée

Emmanuel Ma, Belle Pan

Abstract

Abstract interpretation is a theory of approximation, and a technique used for analyzing and reasoning about the behaviour of computer programs. In this paper, we explore the basic theory of abstract interpretation, the advantages that come with its use, and its application in Astrée. The goal of this report is to give the background to make abstract interpretation and Astrée more understandable.

1 Introduction

Abstract interpretation is a form of static analysis, in which the program is analyzed without actually executing it, in order to determine certain properties of its behavior. Abstract interpretation is a theory of approximation developed by Patrick Cousot and Radhia Cousot[2] - it uses mathematical abstractions to approximately represent the possible behavior of a program. While abstractions can take on many forms depending on the specific properties that are being analyzed, a few common ways to represent the behaviour of a program are lattices, sets, graphs, and Boolean expressions. A key application of abstract interpretation is in program verification, especially for programs that are too large or complex to be executed directly, and programs that are known to be incorrect. Furthermore, abstract interpretation can be used to automatically infer the safety and correctness properties of a program, which is useful for verifying critical systems.

The general schema of a static analyzer is $Program \longrightarrow Analyzer \longrightarrow Properties$. The absence of runtime errors (ex. division by zero, square root of negative numbers, overflows, array index out of bounds, and pointer dereference outside objects), worst-case execution time, and security properties (ex. secrecy, and authenticity of protocols) are some examples of properties proved by abstract interpretation in program verification.

As most interesting program properties are undecidable from Rice's theorem, such as the grand example of whether a program terminates, i.e. the halting problem, the abstraction of a program must be sound (but not necessarily complete) approximations. If the verifier concludes that the abstraction would execute with no errors, then the program must be error-free. This is the basis of soundness in abstractions, that the abstraction must encompass all possible behaviours of the program. However, the abstraction need not be complete in some cases: if the verifier concludes that the abstraction would encounter an error, it indicates that the program may or may not be erroneous - which is not exactly an incorrect answer but certainly not a helpful one in many situations.

These abstractions must be formalized to be able to prove the soundness of the program analyses. To do so, we first formalize the meaning of semantics (programs), and then its abstractions (approximations). After this, we delve into the application of abstract interpretation in Astrée.

2 Preliminaries

We first introduce some background information on the mathematical structures that hold the theory together. Our presentation of the syntax of lattices and Galois connections is from Bruno Blanchet's paper on abstract interpretation[1].

2.1 Order

An order is a binary relation that defines a set of values and specifies whether one value is greater than, less than, or equal to another value. A partially ordered set, a poset, is a set S with a binary relation $\leq$ such that these three properties are fulfilled:

- $\leq$ is reflexive: $\forall a \in S, a \leq a$
- $\leq$ is transitive: $\forall a, b, c \in S$, if $a \leq b$, $b \leq c$, then $a \leq c$
- $\leq$ is antisymmetric: $\forall a, b \in S$, if $a \leq b$, $b \leq a$, then $a = b$

Furthermore,

- $n \in S$ is an upper bound of some $X \subseteq S$ if and only if $\forall n' \in X, n' \leq n$
- $n \in S$ is a lower bound of some $X \subseteq S$ if and only if $\forall n' \in X, n \leq n'$
- $n \in S$ is the least upper bound of some $X \subseteq S$ if and only if $\forall n' \in X, n' \leq n$ and $\forall n'' \in S$ s.t. $\forall n' \in X, n' \leq n''$ and $n \leq n''$
- $n \in S$ is the greatest upper bound of some $X \subseteq S$ if and only if $\forall n' \in X, n \leq n'$ and $\forall n'' \in S$ s.t. $\forall n' \in X, n'' \leq n'$ and $n'' \leq n$

When it exists, the least upper bound and greatest lower bound are unique.

An ordering is total, or comparable when $a \leq b$ or $b \leq a$. However, an ordering is not always comparable; that is, we may not always have $a \leq b$ or $b \leq a$. An example would be comparing parts of S ordered by inclusion ($\subseteq$): $\{a\}$ is incomparable with $\{b\}$ if $a \neq b$.

Let $(\leq)$ be a partially ordered set. A *chain* $C = (x_n)_{n \in \mathbb{N}}$ is a monotone sequence of elements of $S : x_0 \leq x_1 \leq ... \leq x_n \leq x_{n+1} \leq ...$

A *complete partial order* (cpo) is a poset $S(\leq)$ such that S has a least element $\bot$ and every chain C has a least upper bound $\sqcup C$.

2.2 Lattices

A lattice is a partially ordered set, $L(\leq)$, in which $\forall a, b \in L$, a, b have a unique least upper bound , $a \sqcup b$, and a unique greatest lower bound, $a \sqcap b$. Such a lattice is denoted by $L(\leq, \sqcap, \sqcup)$. In a lattice, while all finite sets have least upper bounds and greatest lower bounds, infinite sets may not necessarily have them.

Similarly, a complete lattice is a partially ordered set, $L(\leq)$. However, every subset $X \in L$ has a least upper bound $\sqcup X$ and a greatest lower bound $\sqcap X$. L also has a least element $\bot = \sqcup \emptyset$ and a greatest element $\top = \sqcup L$. This lattice is denoted $L(\leq, \bot, \top, \sqcup, \sqcap)$.

2.3 Galois Connections

A Galois connection is a mathematical relationship between two partially ordered sets, $L_1(\leq_1)$ and $L_2(\leq_2)$. It consists of two functions, (α, γ), where α maps elements of one set to the other, and γ maps elements of the other set back to the first. (α, γ) is a Galois connection between L_1 and L_2 if and only if they satisfy the following properties:

$$\alpha \in L_1 \longrightarrow L_2, \gamma \in L_2 \longrightarrow L_1$$

and

$$\forall x \in L_1, \forall y \in L_2, \alpha(x) \leq_2 y \Leftrightarrow x \leq_1 \gamma(y)$$

which can be rewritten as:

$$(L_1, \leq_1) \underset{\gamma}{\overset{\alpha}{\rightleftarrows}} (L_2, \leq_2)$$

Some relevant properties of Galois connections are:

Proposition 1. *$\gamma \circ \alpha$ is extensive:*

$$\forall x \in L_1 : x \leq_1 \gamma \circ \alpha(x)$$

since $\alpha(x) \leq_1 \alpha(x)$ by reflexivity, hence $x \leq_1 \gamma \circ \alpha(x)$ with $y = \alpha(x)$.

Proposition 2. *The same way, $\alpha \circ \gamma$ is reductive:*

$$\forall y \in L_2 : \alpha \circ \gamma(y) \leq_2 y$$

since $\gamma(y) \leq_1 \gamma(y)$ by reflexivity, hence $\alpha \circ \gamma(y) \leq_2 y$ with $x = \gamma(y)$.

Proposition 3. *(α, γ) is a Galois connection if and only if α and γ are monotone, $(\alpha \circ \gamma)(y) \leq_2 y$, and $x \leq_1 (\gamma \circ \alpha)(x)$.*

Proof. Assuming (α, γ) is a Galois connection, we know the following:

- By definition, $\alpha(x) \leq_2 y$ and by reflexivity $\gamma(y) \leq_1 \gamma(y)$, thus $(\alpha \circ \gamma)(y) \leq_2 y$.
- By definition, $x \leq_1 \gamma(y)$, and by reflexivity $\alpha(x) \leq_2 \alpha(x)$, thus $x \leq_1 (\gamma \circ \alpha)$.
- Let $x \leq_1 x'$. Because $x' \leq_1 (\gamma \circ \alpha)(x')$, by transitivity $x \leq_1 (\gamma \circ \alpha)(x')$, and thus $\alpha(x) \leq_2 \alpha(x')$ and α is monotone.
- Let $y \leq_2 y'$. Because $(\alpha \circ \gamma)(y) \leq_2 y$, by transitivity $(\alpha \circ \gamma)(y) \leq_2 y'$, and thus $\gamma(y) \leq_1 \gamma(y')$ and γ is monotone.

To prove the converse, assume α and γ are monotone, i.e. $(\alpha \circ \gamma)(y) \leq_2 y$, and $x \leq_1 (\gamma \circ \alpha)(x)$

- Assume that $\alpha(x) \leq_2 y$. Since γ is monotone, then $\gamma(\alpha(x)) \leq_1 \gamma(y)$. By assumption, $x \leq_1 \gamma(\alpha(x))$, and by transitivity, $x \leq_1 \gamma(y)$.
- Assume that $x \leq_1 \gamma(y)$. Since α is monotone, then $\alpha(x) \leq_2 \alpha(\gamma(y))$. By assumption, $\alpha(\gamma(y)) \leq_2 y$, and by transitivity $\alpha(x) \leq_2 y$.

□

3 Defining Semantics and Their Domains

The *semantics* of a program describes the set of all possible behaviours given all possible input data of the program when it is executed. These behaviours can be non-terminating, terminating with an error, correct termination with some output, and many other examples.

Our presentation of the syntax of the following semantics is from the paper by Patrick Cousot and Radhia Cousot on abstract interpretation frameworks, published in 1992[3].

3.1 Standard and Collecting Semantics

The correctness of abstract interpretation relies on the existence of *standard semantics* that describe the possible behaviours of programs during runtime, as the abstract interpretation of a program is an approximation of these semantics. Then, from the standard semantics, semantics relevant to the runtime behaviour of the program are collected. This set, called the *collecting semantics*, can be thought of as a more refined version of the standard semantics, containing only the properties relevant to the program execution. Its purpose is to provide a sound basis upon which we can approximate semantics from.

The collecting semantics can be formalized as the fixpoint of possible program traces. Let the standard semantics be the set of program execution traces (usually referred to as trace semantics). We then define a function $F_t(T) = \{s|s \in\} \bigcup \{t \to s' | t \in T, T \text{ ends in } s, s \to s'\}$, where S_0 is the set of initial execution states and T is the set of traces. Then the set of possible traces is $\text{lfp}(F_t)$.

(Tarski) *The set of fixpoints of a monotone operator f on a complete lattice $L(\leq, \bot, \top, \sqcup, \sqcap,)$ is a non-empty complete lattice, ordered by $\leq$. Then,*

$$lfp(f) = \sqcap\{x \in L | f(x) \leq x\}$$

and

$$gfp(f) = \sqcup\{x \in L | x \leq f(x)\}$$

Moving forward, to understand the connection between standard and collecting semantics, and collecting and abstract semantics, we study the connection between concrete and abstract semantics. Note that the terms concrete and abstract semantics are relative: for example, the collecting semantics are abstract with respect to the standard semantics, yet also concrete with respect to further abstraction.

3.2 Concrete Domain

Before we can establish a correspondence between concrete and abstract semantics, we must first define their respective domains. The *concrete semantic domain* can be expressed as a set $\mathcal{P}^\natural$ whose elements c represent concrete semantic properties of some program of interest. Examples include the set of maximal traces from a program execution or the set of possible program states. The concrete semantic domain can include both finite and infinite sets of values.

3.3 Concrete Semantics

Concrete semantics are properties c chosen from $\mathcal{P}^\natural$ that represents some characteristic of a program's possible executions. The concrete semantics of a program is usually represented as the limit of the

iteration of a *concrete semantic function*, a partial map denoted $F^\natural \in \mathcal{P}^\natural \hookrightarrow \mathcal{P}^\natural$ where $S \hookrightarrow T$ is the set of partial functions of the set S into the set T. The limit of the concrete semantic function is $F^{\natural\epsilon}$. The concrete semantics of a program can be inductively constructed from the concrete semantic function starting from some basis $\bot^\natural$ to limit ordinals using an inductive join, $\coprod^\natural \in \wp(\mathcal{P}^\natural \hookrightarrow \mathcal{P}^\natural)$; we can denote this as $F^{\natural\lambda}$ for all ordinals $\lambda \in Ord$.

$$\begin{cases} F^{\natural 0} = \bot^\natural \\ F^{\natural\lambda+1} = F^\natural(F^{\natural\lambda}) \\ F^{\natural\lambda} = \coprod_{\beta<\lambda}^\natural F^{\natural\beta} \quad \textit{when } \lambda > 0 \textit{ is a limit ordinal} \end{cases} \tag{1}$$

Note that ordinals are often used in abstract interpretation to represent the relative positions of elements in a set of concrete or abstract values, such as integers, real numbers, or other mathematical objects. Each element of the $F^{\natural\lambda}$ sequence is called a *concrete iterate*, and the sequence itself is called the *concrete iteration*.

We say that the iteration is total when all its iterates are well-defined; otherwise, the iteration is partial. The iteration is said to be convergent with limit $F^{\natural\epsilon}$ whenever it is total and ultimately stationary, i.e. $\exists \epsilon \in Ord : \forall \lambda \geq \epsilon : F^{\natural\lambda} = F^{\natural\epsilon}$.

Another interesting property of concrete iterates is that it may be in increasing order for a partial order $\sqsubseteq^\natural \in \wp(\mathcal{P}^\natural \times \mathcal{P}^\natural)$. This partial order relation may induce a complete partial order or even a complete lattice structure on $\mathcal{P}^\natural$. This implies that $\coprod^\natural$ may be the corresponding least upper bound and $\bot^\natural$ the least element when such an order exists, ensuring that the concrete iteration is convergent. However, the least upper bounds $\sqcup_{\beta<\lambda}^\natural F^{\natural\beta}$ are needed for the iteration sequence $F^{\natural\lambda}, \lambda \leq \epsilon$ only, not for all directed sets of $\mathcal{P}^\natural$. Hence, the concrete domain, $\mathcal{P}^\natural$, does not necessarily need to be a complete partial order.

3.4 Abstract Domain

The design of the *abstract domain*, $\mathcal{P}^\sharp$ is usually the first decision made when constructing the abstraction. The abstract domain, $\mathcal{P}^\sharp$, is a set of objects that is an approximate representation of the concrete domain, $\mathcal{P}^\natural$. An example could be expressing invariance properties as a set of program states.

3.5 Abstract Semantics

To find an *abstract semantic* $a \in \mathcal{P}^\sharp$ that is a correct approximation of the concrete semantics $c \in \mathcal{P}^\natural$ of a program, one must design some method for associating $a \in \mathcal{P}^\sharp$ to the program. An abstract interpretation of the concrete iteration, $F^{\natural\lambda}$, can be made via an *abstract iteration*, $F^{\sharp\lambda}, \lambda \in Ord$ with limit $F^{\sharp\epsilon}$. The abstract iteration is also a stationary partial map denoted $F^\sharp \in \mathcal{P}^\sharp \hookrightarrow \mathcal{P}^\sharp$ with the limit $F^{\sharp\epsilon}$.

Similar to the concrete semantics of a program, abstract semantics can be inductively constructed from the *abstract semantics function*, $F^\sharp$, starting from some basis $\bot^\sharp$ to limit ordinals using an inductive join, $\coprod^\sharp \in \wp(\mathcal{P}^\sharp \hookrightarrow \mathcal{P}^\sharp)$; we can denote this as $F^{\sharp\lambda}$ for all ordinals $\lambda \in Ord$.

$$\begin{cases} F^{\sharp 0} = \bot^\sharp \\ F^{\sharp\lambda+1} = F^\sharp(F^{\sharp\lambda}) \\ F^{\sharp\lambda} = \coprod_{\beta<\lambda}^\sharp F^{\sharp\beta} \quad \textit{when } \lambda > 0 \textit{is a limit ordinal} \end{cases} \tag{2}$$

The abstract iterates may be in increasing order for a partial order $\sqsubseteq^\sharp \in \wp(\mathcal{P}^\sharp \times \mathcal{P}^\sharp)$. This may induce an ordering, $\langle \mathcal{P}^\sharp; \sqsubseteq^\sharp, \bot^\sharp, \sqcup^\sharp \rangle$, which ensures that the abstract iteration converges.

4 Computing the Abstract Approximation

The main goal of abstract interpretation is to find an abstract property $a \in \mathcal{P}^\sharp$ that soundly approximates the concrete semantics. The following syntax is derived from the 1992 paper by Patrick Cousot and Radhia Cousot[3], and the 1993 paper by Kim Marriott[5], with slight modifications for clarity and consistency.

4.1 Soundness Correspondence

To begin, we introduce the idea of a soundness relation σ which we can use to reason about the correspondence between concrete and abstract semantics. We can define the meaning of the abstract properties by a relation

$$\sigma \in \wp(\mathcal{P}^\natural \times \mathcal{P}^\natural)$$

where

$$\langle c, a \rangle \in \sigma$$

means that the concrete semantic c has the abstract property a.

4.2 Proving Soundness of Abstract Semantics

We prove the soundness of abstract semantics inductively. The goal is to show that $\langle F^{\natural\epsilon}, F^{\sharp\epsilon'} \rangle \in \sigma$:

Proof.

- The base case is $\langle F^{\natural 0}, F^{\sharp 0} \rangle \in \sigma$
- The inductive step is $\forall \lambda, \lambda' \in Ord : \langle F^{\natural\lambda}, F^{\sharp\lambda'} \rangle \in \sigma \implies \exists \mu > \lambda, \exists \mu > \lambda' \in Ord : \langle F^{\natural\mu}, F^{\sharp\mu'} \rangle \in \sigma$

By induction on λ, we have $\langle F^{\natural\lambda} F^{\sharp\lambda} \rangle \in \rho \forall \lambda \in Ord$. Since both sequences, $F^\natural, F^\sharp$ converge, we have $F^{\natural\epsilon} = F^{\natural\mu}, F^{\sharp\epsilon'} = F^{\sharp\mu}$ and $\langle F^{\natural\mu}, F^{\sharp\mu} \rangle \in \rho$ for the maximum μ of ϵ, ϵ' □

4.3 Abstraction and Concretization Functions

The semantics of abstract properties can be given by a concretization function $\gamma \in \mathcal{P}^\sharp \mapsto \mathcal{P}^\natural$, where $\gamma(\mathcal{P}^\sharp)$ is the concrete property corresponding to the abstract property $\rho^\sharp \in \mathcal{P}^\sharp$. In our framework, we assume that there is always an abstract approximation for every concrete property:

$$\forall c \in \mathcal{P}^\natural \exists a \in \mathcal{P}^\sharp : \langle c, a \rangle \in \sigma$$

We can then formalize the idea of approximation by introducing an abstraction function $\alpha \in \mathcal{P}^\natural \mapsto \mathcal{P}^\sharp$ which gives the best approximation $\alpha(\rho^\natural)$ of a concrete property. This can be formalized as

$$\alpha = \{\langle c, a \rangle \in \sigma \| \forall a' \in \mathcal{P}^\sharp : (\langle c, a' \rangle \in \sigma \wedge a' \preceq^\sharp a) \Rightarrow (a \preceq^\sharp a')\}$$

If $p_1^\sharp = \alpha(p^\natural)$ and $p_1^\sharp \preceq^\sharp p_2^\sharp$, then the abstract property $p_2^\sharp$ is also a sound, albeit less precise approximation of $p^\natural$. Thus, $\alpha(p^\natural) \preceq^\sharp p^\sharp$. Next, if $p_1^\natural = \gamma(p^\sharp)$ and $p_2^\natural \preceq^\natural p_1^\natural$, then $p^\sharp$ also approximates

$p_2^\natural$ with $p_2^\natural$, providing more precise information on the program execution, so $p^\natural \preceq^\natural \rho(p^\sharp)$. If the two relations are equivalent, then they form exactly a Galois connection:

$$\mathcal{P}^\natural(\preceq^\natural) \underset{\alpha}{\overset{\gamma}{\rightleftarrows}} \mathcal{P}^\sharp(\preceq^\sharp)$$

From this, we can reinterpret the reductivity property of Galois connections as the fact that the concretization process introduces no loss of information. From an abstract point of view, $\alpha(p^\natural)$ is as precise as possible. Also, reflexivity can be interpreted by the fact that the loss of information in the abstraction process is sound. It follows that α and γ are monotone functions, from the fact that abstraction and concretization preserve the soundness of approximation.

Thus with this revelation, given that $(L_1, \leq_1)$ is the concrete lattice, and $(L_2, \leq_2)$ is the abstract lattice, α is said to be the abstraction, and γ is the concretization. The two properties $\alpha(x) \leq_2 y$ and $x \leq_1 \gamma(y)$ both mean that y is a correct approximation of the concrete value x. $\alpha(x)$ is the most precise approximation of $x \in L_1$ in L_2. $\gamma(y)$ is the least precise element of L_1 which can be approximated by $y \in L_2$.

Example 4.1. *We can approximate parity as follows: The concrete interpretation of parity is* $\rho\mathbb{N}$ *where* $\mathbb{N}$ *is a set of natural numbers, and the abstract interpretation of parity is a complete lattice of descriptions. The concretization function* $\gamma : Parity \to \rho\mathbb{N}$ *is defined by:*

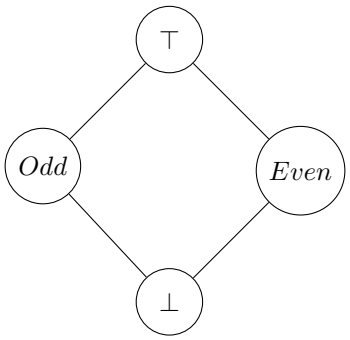

Figure 1: Abstract interpretation of parity: a complete lattice.

$$\gamma P = \begin{cases} \emptyset, & \text{if } P = \bot \\ \{n \in \mathbb{N} | n \text{is odd}\}, & \text{if } P = odd \\ \{n \in \mathbb{N} | n \text{is even}\}, & \text{if } P = even \\ \mathbb{N}, & \text{if } P = \top \end{cases} \tag{3}$$

And the abstraction function $\alpha : \rho\mathbb{N} \to Parity$ *is defined by:*

$$\alpha\mathbb{N} = \begin{cases} \bot, & \text{if } \mathbb{N} = \emptyset \\ odd, & \text{if } \forall n \in \mathbb{N}, n \text{is odd} \\ even, & \text{if } \forall n \in \mathbb{N}, n \text{is even} \\ \top, & \text{otherwise} \end{cases} \tag{4}$$

For instance, the set {2,4} is approximated by the abstraction function as even, because $\{2,4\} \subseteq (\gamma\ even)$. *The set {2,3}, however, is approximated by* $\top$.

Proposition 4. *Define the set of concrete properties* $\mathcal{P}^\natural$, *the set of abstract properties* $\mathcal{P}^\sharp$, *the relation* $\rho \in (\mathcal{P}^\natural \times \mathcal{P}^\sharp)$, *the bases* $\bot^\natural \in \mathcal{P}^\sharp$, *and* $\bot^\sharp \in \mathcal{P}^\natural$ *such that* $\langle \bot^\natural, \bot^\sharp \rangle \in \rho$ *the concrete semantic function,* $F^\natural$, *and the abstract semantic function,* $F^\sharp$ *such that* $\forall c \in \mathcal{P}^n atural : \forall a \in \mathcal{P}^\sharp : \langle c, a \rangle \in \rho \implies \langle F^\natural(c), F^\sharp(a) \rangle \in \rho$, *the concrete and abstract inductive joins,* $\coprod^\natural, \coprod^\sharp$ *such that* $(\forall \beta < \lambda : \langle F^{\natural\beta}, F^{\sharp\beta} \rangle \in \rho) \implies \langle \coprod^\natural_{\beta<\lambda} F^{\natural\beta}, \coprod^\sharp_{\beta<\lambda} F^{\sharp\beta} \rangle \in \rho$ *for all limit ordinals* $\lambda > 0$. *Assuming that the concrete and abstract iteration sequences converge, then their respective limits* $\langle F^{\natural\epsilon}, F^{\natural\epsilon'} \rangle \in \rho$.

4.4 Construction

Abstract semantics can be systematically constructed from concrete semantics and the Galois connection. Our presentation of the syntax of abstractions follows Bruno Blanchet's paper on abstract interpretation[1], with slight modifications.

The general approach to abstractions is to start from the concrete semantics, obtain the collecting semantics from it, and then abstract the collecting semantics. Let us define first some syntax for abstractions of values, environments, expressions, and states:

	Concrete semantics	Collecting semantics	Abstraction	Abstract semantics
Values	$\mathbb{Z}$	$\mathcal{P}(\mathbb{Z}), \subseteq$	$\underset{\gamma_S}{\overset{\alpha_S}{\rightleftarrows}}$	$L, \leq$
Environments	$\rho : Var \to \mathbb{Z}$	$R : \mathcal{P}(Var \to \mathbb{Z}), \subseteq$	$\underset{\gamma_R}{\overset{\alpha_R}{\rightleftarrows}}$	$R^\sharp : Var \to L, \leq$
Expressions	$[\![E]\!]\rho \in \mathbb{Z}$	$[\![E]\!]R \in \mathcal{P}(\mathbb{Z})$		$[\![E]\!]^\sharp R^\sharp \in L$

4.4.1 Abstraction of Values

To verify the correctness of an abstraction, $a^\sharp$, of a value a, one must check if $\alpha(a) \leq a^\sharp$. This is equivalent to checking $a \leq \gamma(a^\sharp)$.

4.4.2 Abstraction of Environments

Let us better define what an environment is and expand on the syntax presented above: environments map variables to values (ex. $\in \mathbb{Z}$). ρ represents a single environment, and R is the set of all environments.

The abstraction of environments can be defined in two steps (with one additional but optional step to finding the Galois connection of environments):

1. Abstract sets of environments to mappings from variables to sets of integers

$$\mathcal{P}(Var \to \mathbb{Z}) \underset{\gamma_{R1}}{\overset{\alpha_{R1}}{\rightleftarrows}} Var \to \mathcal{P}(\mathbb{Z})$$

   The abstraction is defined by $\alpha_{R1}(R_1) = \lambda x.\{\rho(x) | \rho \in R_1\}$, and the concretization is defined by $\gamma_{R1}(R_1^\sharp) = \{\lambda x.y | y \in R_1^\sharp(x)\}$. Note that this abstraction ensures that only the set of possible values of each variable is kept.

2. Abstract each result into L via (α, γ)

$$Var \to \mathcal{P}(\mathbb{Z}) \underset{\gamma_{R2}}{\overset{\alpha_{R2}}{\leftrightarrows}} Var \to L$$

The abstraction is defined by $\alpha_{R2}(R_2) = \lambda x.\alpha(R_2(x))$, and the concretization is defined by $\gamma_{R2}(R_1^\sharp) = \lambda x.\gamma(R_2^\sharp(x))$.

3. The Galois connection of environments:

$$\mathcal{P}(Var \to \mathbb{Z}) \underset{\gamma_R}{\overset{\alpha_R}{\leftrightarrows}} Var \to L$$

may be obtained by composing the two Galois connections:

$$\alpha_R = \alpha_{R2} \circ \alpha_{R1}$$

$$\gamma_R = \gamma_{R1} \circ \gamma_{R2}$$

4.4.3 Abstraction of Expressions

$[\![E]\!]\rho$ is the evaluation of the expression E in a specified environment ρ in the set of all environments in R. In other words, $[\![E]\!]R = \{[\![E]\!]\rho | \rho \in R\}$. To check the correctness of the abstract semantics of an expression, one must verify whether the environment is correctly abstracted; if $\alpha_R(R) \leq R^\sharp$ then $\alpha([\![E]\!]R \leq [\![E]\!]^\sharp R^\sharp$. In other words, if an environment is abstracted correctly, then the result of the expression is correctly abstracted.

5 Application of Abstract Interpretation in Astrée

Astrée is a fully automatic static code analyzer that uses abstract interpretation to prove the absence of runtime errors and invalid concurrent behavior in safety-critical software written or generated in C or C++. Using abstract interpretation, Astrée builds an overapproximation of the trace semantic properties of a program and proves that the abstract properties imply the absence of runtime errors. There are several different abstractions that are automatically combined to better represent the software and thus increase precision: non-relational values (such as intervals), and weakly relational figures (such as octagons). The soundness of abstractions is based on Galois connections.

Abstract interpretation is particularly useful in the context of Astrée because it allows for a modular and scalable approach to verification. Astrée uses a combination of different abstract domains that are assembled and parameterized (via user specifications) to build application-specific analyzers, improving precision vastly compared to other analyzers. The modularization and speed of Astrée are achieved easily as it is written in OCaml; its modules and functors allow for a clean division of code while allowing separated compilation. Furthermore, in case of false alarms, Astrée can be easily extended by introducing new abstract domains enhancing the precision of the analysis. These characteristics allow Astrée to balance the need for accuracy with the need for efficiency, ensuring that the analysis is both precise and scalable. Astrée can be run on multicore parallel or distributed machines and has been able to since 2005.

By using abstract interpretation, Astrée is able to provide guarantees about the behaviour of a system without having to consider all possible inputs or all possible implementations of the system. This allows Astrée to perform analysis quickly and accurately, even for complex and large-scale systems.

The development of Astrée started in November 2001 at Laboratoire d'Informatique of the école Normale Supérieure (LIENS) under the support of the ASTRéE project, the Centre National de la Recherche Scientifique, the école Normale Supérieure and, since September 2007, by INRIA (Paris-Rocquencourt). The first presentation of Astrée was in August, 2004 at the topical day on abstract interpretation of the IFIP World Computer Congress in Toulouse (France), and made its paper debut in 2005. For further reading, please see the paper by the project participants and leaders, Patrick Cousot, Radhia Cousot, Jerôme Feret, Laurent Mauborgne, Antoine Miné, David Monniaux, and Xavier Rival[4].

References

[1] Bruno Blanchet. Introduction to abstract interpretation. November 2002.

[2] Patrick Cousot and Radhia Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Conference Record of the Fourth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 238–252, Los Angeles, California, January 1977. ACM Press, New York, NY.

[3] Patrick Cousot and Radhia Cousot. Abstract interpretation frameworks. In *Journal of Logic and Computation*, volume 2, pages 511–547. Oxford University Press, August 1992.

[4] Patrick Cousot, Radhia Cousot, Jerôme Feret, Laurent Mauborgne, Antoine Miné, David Monniaux, and Xavier Rival. The astrée analyzer. In *Programming Languages and Systems*, pages 21–30, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[5] Kim Marriott. Frameworks for abstract interpretation. In *Acta Informatica*, volume 30, pages 103–129, Berlin, Heidelberg, March 1993. Springer-Verlag.