Class Notes by Michal Segalov, Arik Nemtsov

Abstract Interpretation – Part II

As described in the previous lecture, abstract interpretation deals with the approximation of computer programs, based on monotonic functions over ordered sets (usually lattices). It can be viewed as a partial execution of a computer program which gains information about its semantics without performing all the calculations.

We choose an abstract domain to represent interesting properties of the program and use abstract semantics to carry out program transitions. Our chaotic-iterations based work-list algorithm is used to find a LFP solution to the program, at which point we can gather valuable insights about the program.

Galois Connections – a Reminder

Let L_1, L_2 be two complete lattices

$(L_1, \subseteq_1) = (L_1, \subseteq_1, \sqcup_1, \sqcap_1, \top_1, \bot_1)$
$(L_2, \subseteq_2) = (L_2, \subseteq_2, \sqcup_2, \sqcap_2, \top_2, \bot_2)$

And let α, γ be the following functions:

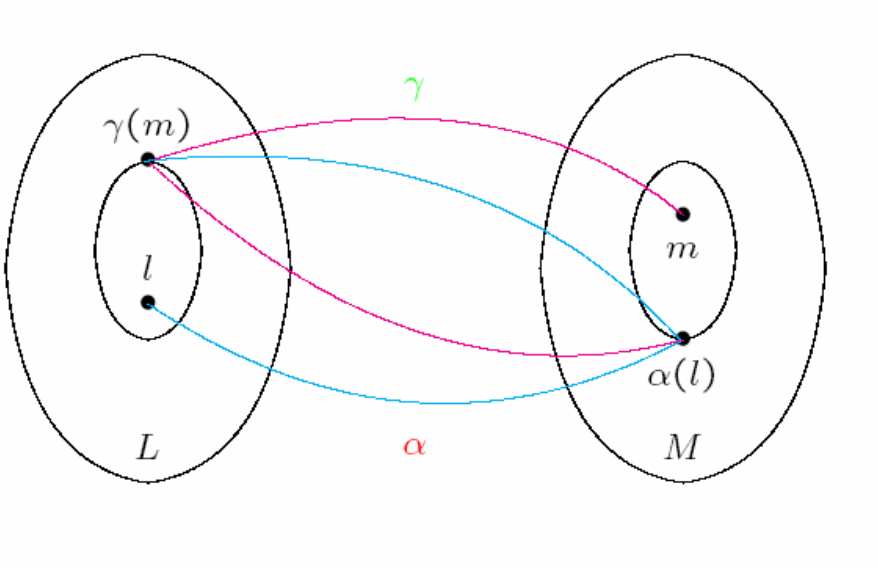
$\alpha : L_1 \to L_2$
$\gamma : L_2 \to L_1$

Then

(L_1, a, γ, L_2) is a Galois connections if

- α, γ are monotone
- for all $c \in L_1$: $\gamma(\alpha(c)) \supseteq c$
- for all $a \in L_2$: $\alpha(\gamma(a)) \subseteq a$

Fact : If (L_1, a, γ, L_2) is a Galois connection then $\gamma \circ \alpha \circ \gamma = \gamma$ *and* $\alpha \circ \gamma \circ \alpha = \alpha$

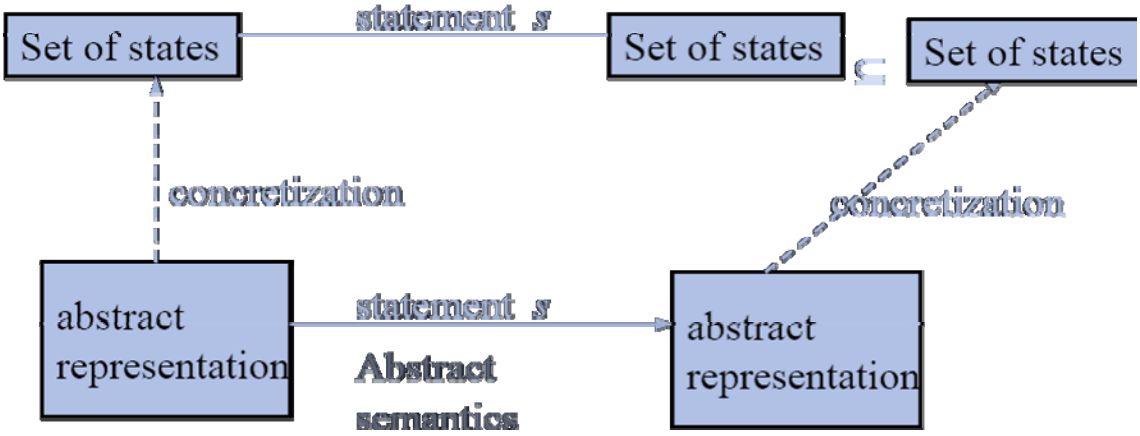


Global Soundness Theorem

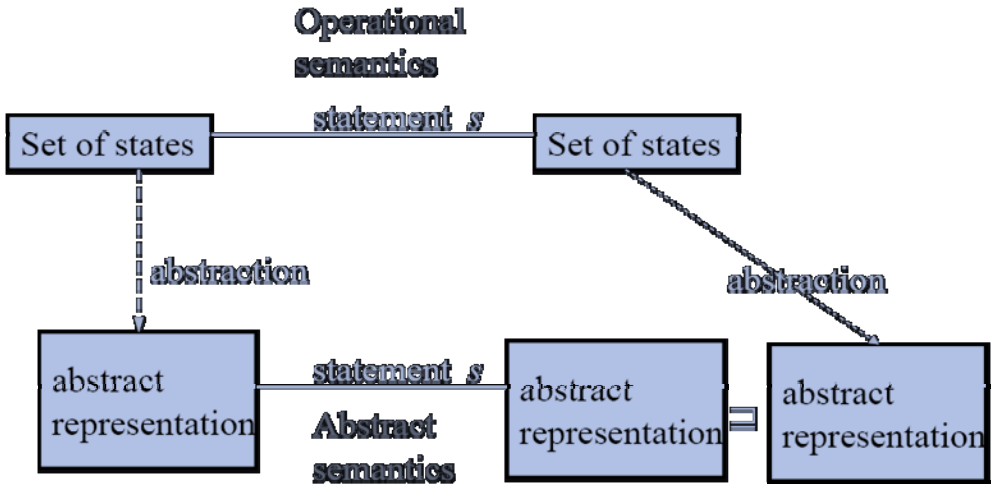
Basically, an abstract interpretation is nothing more than a Galois Connection. Let the Galois Connection (α, γ) from C to A define our abstract interpretation. Let $f: C \to C, f^{\#}: A \to A$ be monotone functions (these represent the transfer functions on our concrete and abstract domains).

We say our abstract interpretation is *globally sound* when one of the following occurs:

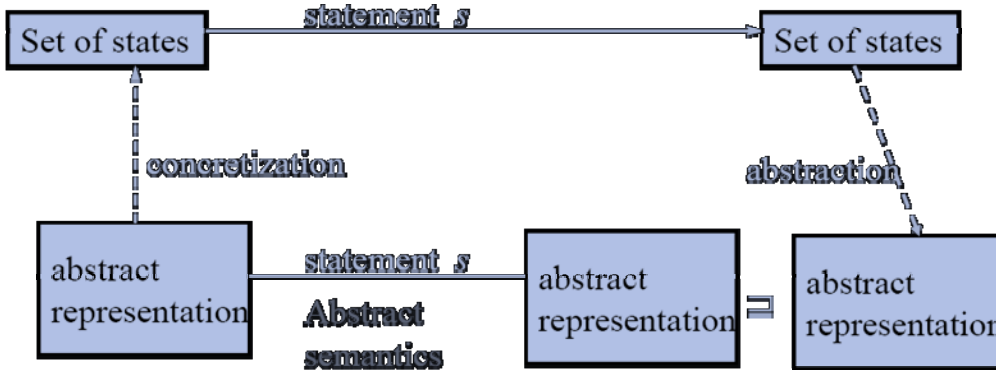
1. $\forall a \in A, f(\gamma(a)) \sqsubseteq \gamma\left(f^{\#}(a)\right)$



2. $\forall c \in C: \alpha(f(c)) \sqsubseteq f^{\#}(\alpha(c))$



3. $\forall a \in A: \alpha\left(f\left(\gamma(a)\right)\right) \sqsubseteq f^{\#}(a)$



Note that the three above properties are equivalent – each can derive the other two (using the properties of Galois Connections).

Intuitively, global soundness guarantees the iterative algorithm can only over-approximate the collecting semantics (e.g., err on the side of caution) when using our abstract interpretation scheme. To formalize this notion, we show two important properties of the LFP when using a globally sound abstract interpretation:

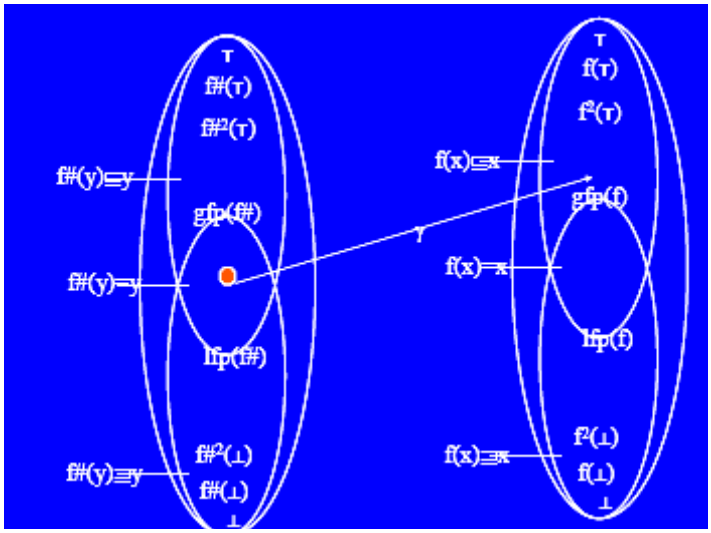
1. $lfp(f) \sqsubseteq \gamma\left(lfp(f^{\#})\right)$

   Proof: let a be the least fixed point of $f^{\#}$. This means $f^{\#}(a) = a$. Now placing this point into expression (1) above we have that $f\left(\gamma(a)\right) \sqsubseteq \gamma\left(f^{\#}(a)\right) = \gamma(a)$, or equivalently, $\gamma(a) \in Red(f)$. But by definition $lfp(f) = \sqcap Red(f)$, so $lfp(f) \sqsubseteq \gamma(a) = \gamma\left(lfp(f^{\#})\right)$.

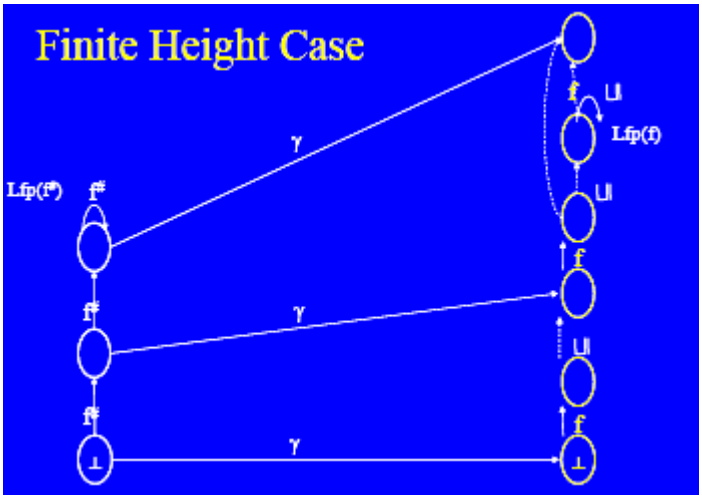
2. $\alpha\left(lfp(f)\right) \sqsubseteq lfp(f^{\#})$

This property is implied by the previous property and the fact a Galois Connection maintains $c \sqsubseteq \gamma(a) \Leftrightarrow \alpha(c) \sqsubseteq a$.

Or graphically:



We can schematically where the concrete and abstract LFP's are encountered when employing both transfer functions (in lattices with finite height):



In globally sound abstract interpretation schemes we are assured $\gamma\left(lfp(f^{\#})\right)$ does not occur "before" $lfp(f)$ when traversing the concrete transfer function (but it may occur much after).

Local Soundness

For any atomic statement S in our language we can define local concrete semantics and local abstract semantics (local in a sense of "belonging to a

single statement"). The local concrete semantics are defined using natural or structural operational semantics. The local abstract semantics have states that are derived from our chosen abstract domain, but are otherwise similar in definition to the concrete semantics.

Let's define for example part of the local concrete and abstract semantics for the Constant Propagation (CP) problem.

Local Concrete Semantics (CP):

- $[\![S]\!]_1[Var_n \to Z] \to [Var_n \to Z]$. A statement transforms a state (in this case variable to integer mapping) to another state.
- $[\![x := a]\!]s = s[x \to A[\![a]\!]s]$. Definition of assignment - A is the operator of meaning for arithmetic expressions (see lecture on operational semantics for details).
- $[\![skip]\!]s = s$. Skip statement does not affect state.
- ....

Local Abstract Semantics (CP):

- $[\![S]\!]^\#_1[Var_n \to L] \to [Var_n \to L]$. This time a state is a mapping from variable to $L = Z \cup \{\top, \bot\}$.
- $[\![x := a]\!]^\# s = s[x \to A([\![a]\!]^\# s)]$. Assignment is basically the same.
- $[\![skip]\!]^\# s = s$. Skip also.
- ....

An atomic statement S is deemed *locally sound* with respect to given concrete and abstract semantics when one of the following is shown:

1. $f(\gamma(a)) \sqsubseteq \gamma(f^\#(a))$ where a is in the abstract domain
2. $\alpha(f(c)) \sqsubseteq f^\#(\alpha(c))$ where c is in the concrete domain
3. $\alpha(f(\gamma(a))) \sqsubseteq f^\#(a)$ where a is in the abstract domain

Notice these properties are the local equivalents (without the $\forall$ sign) of the previous properties 1-3 occurring at global soundness. Again the above three expression are equivalent assuming (α, γ) is a valid Galois Connection.

For the Constant Propagation problem local soundness of a statement implies the following:

1. $\alpha_{CP}(\{[\![S]\!]\sigma | \sigma \in CS\}) \sqsubseteq [\![S]\!]^\#(\alpha_{CP}(CS))$
2. $\{[\![S]\!]\sigma | \sigma \in \gamma_{CP}(DF)\} \sqsubseteq \gamma_{CP}([\![S]\!]^\#(DF))$

3. $\alpha_{CP}(\{[\![S]\!]\sigma|\sigma \in \gamma_{CP}(DF)\}) \sqsubseteq [\![S]\!]^{\#}(DF)$

Where CS is current state given by collecting semantics (in the concrete domain) and DF is the current state given by abstract semantics (in the abstract domain). $[\![S]\!], [\![S]\!]^{\#}$ are defined as before.

Let's show the local soundness of assignment in Constant Propagation:

The abstraction function of CP is defined as $\alpha_{CP}(CS) = \sqcup \{\beta_{CP}(\sigma)|\sigma \in CS\} = \sqcup \{\sigma|\sigma \in CS\}$. It is clear the statement $[\![x := a]\!]$ is a monotone function (part of our monotone framework). For all monotone functions we have:

$\sqcup \{f(z)|z \in X\} \sqsubseteq f(\sqcup \{z|z \in X\})$

And so for α_{CP} we have:

$\alpha_{CP}(\{[\![x := a]\!]\sigma|\sigma \in CS\}) = \sqcup \{[\![x := a]\!]\sigma|\sigma \in CS\}$
$\sqsubseteq [\![x := a]\!](\{\sigma|\sigma \in CS\}) = [\![x := a]\!]^{\#}(\alpha_{CP}(CS))$

Global Soundness

In [1],Cousot & Cousot showed that when the abstract interpretation of every atomic statement is locally sound, the result of the iterative analysis is guaranteed to be sound. This result implies a general scheme for constructing globally sound abstract interpretations:

1. Define appropriate operational semantics (for instance as Natural Operational Semantics) for the concrete domain
2. Define collecting semantics for the concrete domain
3. Define an abstract domain
4. Establish a Galois Connection between collecting states and abstract states
5. Show that the abstract interpretation of atomic statement is locally sound with respect to the collecting semantics
6. Conclude the result of iterative analysis is sound with respect to the collecting semantics (i.e., global soundness)

Cartesian Product

We wish to take two lattices and use them to define the lattice which represents their Cartesian product.

Given L_1, L_2, define a poset $L = (L_1 \times L_2, \subseteq)$:

- $(x_1, x_2) \subseteq (y_1, y_2)$ if $x_1 \subseteq y_1 \ \wedge \ x_2 \subseteq y_2$

L is a complete lattice.

To understand what the elements in L represent, we use a concrete lattice C, and define a Galois Connection between L and C. (C is usually a powerset)

Formally, given $L_1, L_2, L = (L_1 \times L_2)$ a concrete lattice C and two Galois connections $(C, a_1, \gamma_1, L_1), (C, a_2, \gamma_2, L_2)$ define the Galois connection $(C, a, \gamma, L_1 \times L_2)$ as following:

$\alpha : C \to L_1 \times L_2$ and $\gamma : L_1 \times L_2 \to C$

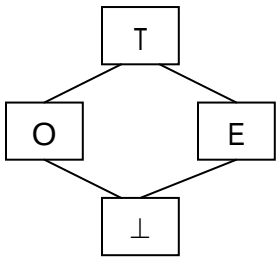
$\alpha(c) = \langle \alpha_1(c), \alpha_2(c) \rangle$

$\gamma(a) = \langle \gamma_1(a), \gamma_2(a) \rangle$

Example:

We saw in class the lattice parity and the lattice sign, we will now describe the lattice (Parity X sign).

Let L_1 be a parity lattice:

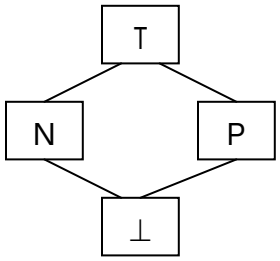


(Where T represents unknown, z is no value, E is even and O is odd)

$\alpha_1(Z) = \{ parity(z) \mid z \in Z \}$

$\gamma_1(A) = \{ z \in Z \mid parity(z) \in A \}$

Let L_2 be a sign lattice

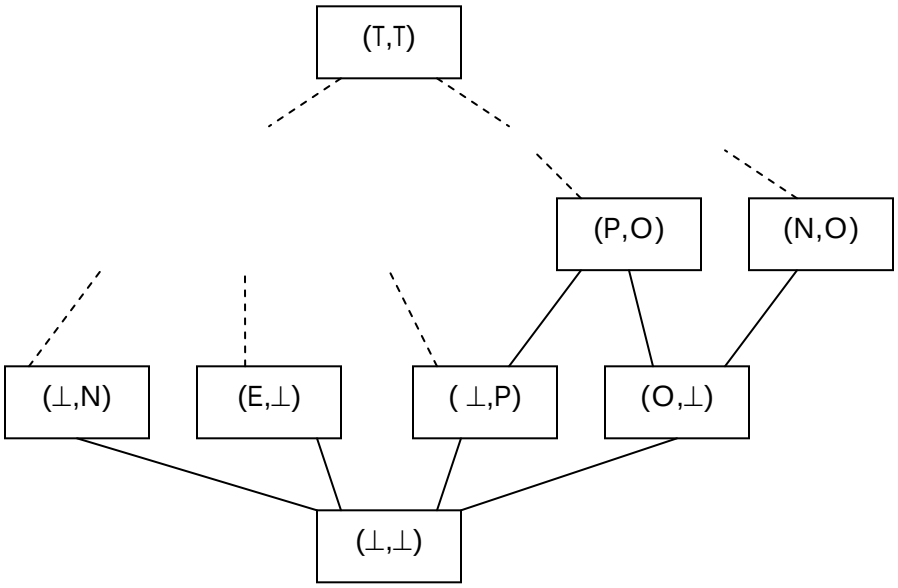


(Where T represents unknown, z is no value, N is negative and P is positive)

$\alpha_2(Z) = \{sign(z) \mid z \in Z\}$

$\gamma_2(A) = \{z \in Z \mid sign(z) \in A\}$

$L_1 \times L_2$ is the lattice:

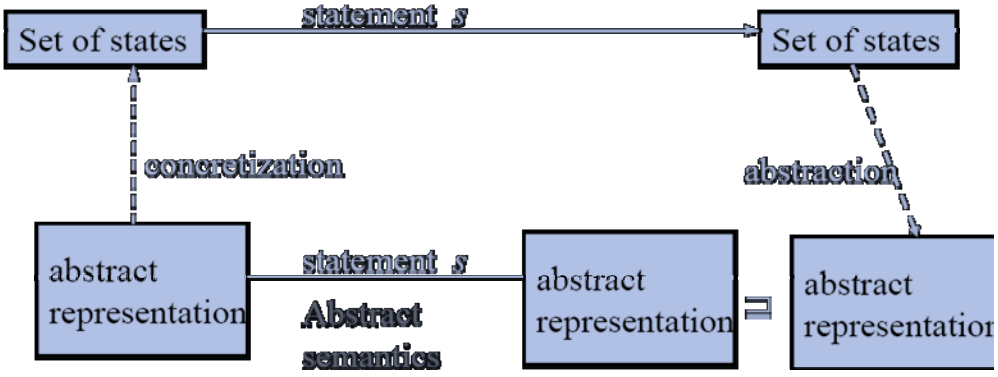


Note that since $\gamma(z_1, z_2) = \gamma_1(z_1) \sqcup \gamma_2(z_2)$, $\alpha(\gamma(z, P)) = (z_1, z_2)$, although $\gamma_1(z) = z_1, \gamma_2(P) \neq z_2$.

Properties of Abstractions

Best Abstract Transformer

If we use a globally sound abstract interpretation the usual situation is this:



Or formally - $\forall a \in A: \alpha\left(f\left(\gamma(a)\right)\right) \sqsubseteq f^{\#}(a)$.

If we concrete our abstract state, perform a statement in the concrete domain and then re-abstract the state, we will usually get a more accurate result than simply performing the statement in the abstract domain. This is reasonable – we use the abstraction to "forget" some properties of the concrete state, thus easing our computational burden, but also reducing accuracy.

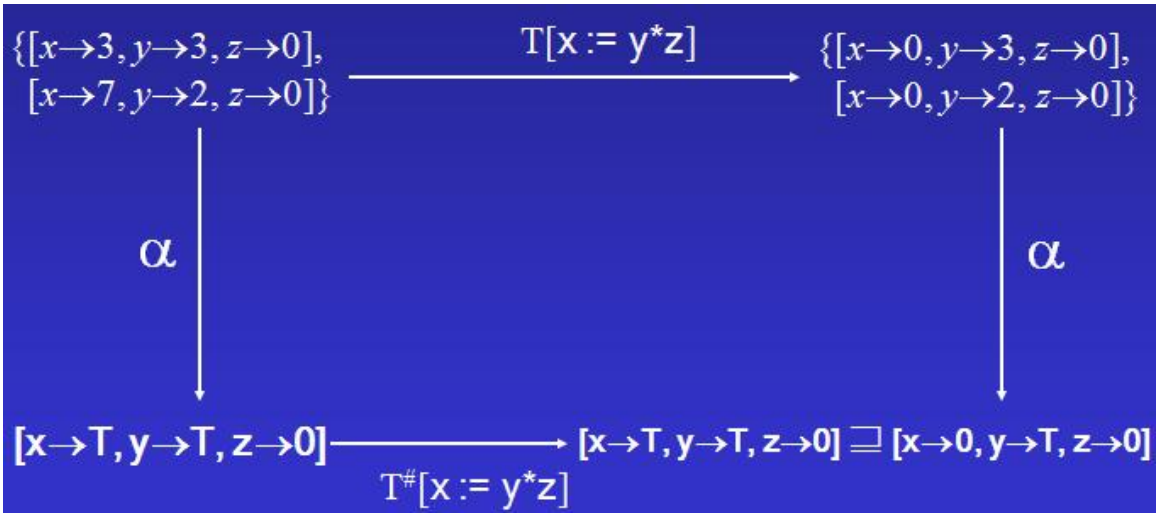
Clearly one cannot get a more accurate result than $\alpha\left(f\left(\gamma(a)\right)\right)$ in the abstract domain. This is essentially cheating – we lose the necessary precision needed to represent the state in the abstract domain, but the collecting semantics assure us maximum precision when performing the statement.

The best transformer is defined as in the above:

$$\alpha \circ f \circ \gamma : A \to A$$

Computing the best transformer can be hard since the collecting semantics for a statement sometimes contain an infinite number of states. Many times we will have to settle for less accurate abstract semantics, but the best transformer provides a measure for the "best" precision obtainable by our abstract interpretation scheme.

Here's an example that shows the abstract transformer in the Constant Propagation problem is not the best transformer:



This is also referred to as the *Induced Analysis* for a problem, as this analysis is induced by the Galois Connection and the original analysis function $f : C \to C$. This type of construction can also be used to approximate an analysis function operating in different concrete domains ($f : A_1 \to A_2$).

We'll show an example of based on approximating the function f_{plus}:

$$f_{\text{plus}}(ZZ) = \{z_1 + z_2 \mid (z_1, z_2) \in ZZ\}$$

Our first concrete domain comes from Detection of Signs Analysis in pairs:

$$(\mathcal{P}(\mathbf{Z} \times \mathbf{Z}), \alpha_{SS'}, \gamma_{SS'}, \mathcal{P}(\mathbf{Sign} \times \mathbf{Sign}))$$

where

$$\begin{aligned} \alpha_{SS'}(ZZ) &= \{(\text{sign}(z_1), \text{sign}(z_2)) \mid (z_1, z_2) \in ZZ\} \\ \gamma_{SS'}(SS) &= \{(z_1, z_2) \mid (\text{sign}(z_1), \text{sign}(z_2)) \in SS\} \end{aligned}$$

Our second domain is the unit version of the same analysis:

$$(\mathcal{P}(\mathbf{Z}), \alpha_{\text{sign}}, \gamma_{\text{sign}}, \mathcal{P}(\mathbf{Sign}))$$

where $\text{sign} : \mathbf{Z} \to \mathbf{Sign}$ is specified by:

$$\text{sign}(z) = \begin{cases} - & \text{if } z < 0 \\ 0 & \text{if } z = 0 \\ + & \text{if } z > 0 \end{cases}$$

Now we can define the approximation for f_{plus} based on inducing along the abstraction function:

$$g_{\text{plus}} = \alpha_{\text{sign}} \circ f_{\text{plus}} \circ \gamma_{SS'}$$

We calculate

$$\begin{aligned} g_{\text{plus}}(SS) &= \alpha_{\text{sign}}(f_{\text{plus}}(\gamma_{SS'}(SS))) \\ &= \alpha_{\text{sign}}(f_{\text{plus}}(\{(z_1, z_2) \in \mathbf{Z} \times \mathbf{Z} \mid (\text{sign}(z_1), \text{sign}(z_2)) \in SS\})) \\ &= \alpha_{\text{sign}}(\{z_1 + z_2 \mid z_1, z_2 \in \mathbf{Z}, (\text{sign}(z_1), \text{sign}(z_2)) \in SS\}) \\ &= \{\text{sign}(z_1 + z_2) \mid z_1, z_2 \in \mathbf{Z}, (\text{sign}(z_1), \text{sign}(z_2)) \in SS\} \\ &= \bigcup\{s_1 \oplus s_2 \mid (s_1, s_2) \in SS\} \end{aligned}$$

where $\oplus : \mathbf{Sign} \times \mathbf{Sign} \to \mathcal{P}(\mathbf{Sign})$ is the "addition" operator on signs (so e.g. $+ \oplus + = \{+\}$ and $+ \oplus - = \{-, 0, +\}$).

Abstract Traces

When a program is executed one can follow its *trace*. The trace is the set of states a program passes through when run and it can be formulated in both the concrete and abstract domains.

In the concrete domain a stage of execution can be represented by all program states in the collecting semantics. A concrete trace will show the progress of the program during its different stages of execution. To calculate the set of states reachable in stage $T+1$ we simply take those reachable at stage T and calculate the valid $T+1$ states using our concrete analysis function (loop statements withstanding). This means every state in stage $T+1$ is reachable via a path from T, and so we can draw a path in time that represents our specific program execution.

This is no longer true when dealing with abstract traces. The abstract analysis function is inaccurate and can "fabricate" during its trace. This means our path in time does not necessarily exist anymore.

This can happen even when one uses the best transformer as the abstract semantics, as the abstraction function itself leads to the inaccuracy.

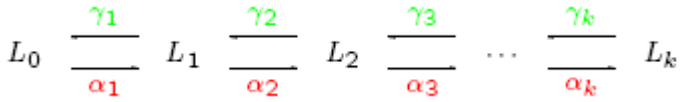
Summary

Abstractions relate runtime semantics and static information to produce a useful and computable analysis for a problem. The concrete semantics helps in designing the abstraction and the best transformer defines precision limits.

Combinations of Abstractions

Functional Composition of Abstractions

The above scheme can also be used to develop abstract interpretations between two abstract domains. Thus we can effectively "overload" abstractions:



We can take for example the problems of Difference in Magnitude and Finite Approximation and overload them:

$$(\mathcal{P}(\mathbf{Z} \times \mathbf{Z}), \alpha_{\text{diff}}, \gamma_{\text{diff}}, \mathcal{P}(\mathbf{Z}))$$

where the extraction function $\text{diff} : \mathbf{Z} \times \mathbf{Z} \to \mathbf{Z}$ calculates the difference in magnitude:

$$\text{diff}(z_1, z_2) = |z_1| - |z_2|$$

The abstraction and concretisation functions are

$$\begin{aligned} \alpha_{\text{diff}}(ZZ) &= \{|z_1| - |z_2| \mid (z_1, z_2) \in ZZ\} \\ \gamma_{\text{diff}}(Z) &= \{(z_1, z_2) \mid |z_1| - |z_2| \in Z\} \end{aligned}$$

for $ZZ \subseteq \mathbf{Z} \times \mathbf{Z}$ and $Z \subseteq \mathbf{Z}$.

$$(\mathcal{P}(\mathbf{Z}), \alpha_{\text{range}}, \gamma_{\text{range}}, \mathcal{P}(\mathbf{Range}))$$

where $\mathbf{Range} = \{\text{<-1}, \text{-1}, 0, \text{+1}, \text{>+1}\}$
and the extraction function $\text{range} : \mathbf{Z} \to \mathbf{Range}$ is

$$\text{range}(z) = \begin{cases} \text{<-1} & \text{if } z < -1 \\ \text{-1} & \text{if } z = -1 \\ 0 & \text{if } z = 0 \\ \text{+1} & \text{if } z = 1 \\ \text{>+1} & \text{if } z > 1 \end{cases}$$

The abstraction and concretisation functions are

$$\begin{aligned} \alpha_{\text{range}}(Z) &= \{\text{range}(z) \mid z \in Z\} \\ \gamma_{\text{range}}(R) &= \{z \mid \text{range}(z) \in R\} \end{aligned}$$

for $Z \subseteq \mathbf{Z}$ and $R \subseteq \mathbf{Range}$.

$$(\mathcal{P}(\mathbf{Z} \times \mathbf{Z}), \alpha_R, \gamma_R, \mathcal{P}(\mathbf{Range}))$$

where

$$\alpha_R = \alpha_{range} \circ \alpha_{diff}$$

$$\gamma_R = \gamma_{diff} \circ \gamma_{range}$$

The explicit formulae for the abstraction and concretisation functions

$$\alpha_R(ZZ) = \{range(|z_1| - |z_2|) \mid (z_1, z_2) \in ZZ\}$$

$$\gamma_R(R) = \{(z_1, z_2) \mid range(|z_1| - |z_2|) \in R\}$$

correspond to the extraction function $range \circ diff$.

Combining Data Flow Analyzers

The idea is to use the algorithms we have to develop new algorithms.

For instance, if we know how to handle pointers and integers separately, how can we combine it and handle them together?

We would like to use the dependencies between integers and pointers to get a better analysis (precision wise and performance wise) without "paying" too much in complexity.

Given two lattices we want to construct a new lattice, which will be a combination of the lattices given.

Component Wise Combinations

We have several analyses of individual components and we wish to combine them into a single analysis.

a. Independent attribute method

b. Relational attribute method

c. Total function space

d. Monotone function space

e. Direct tensor product

Independent attributes method

Given the Galois connections $(C_1, a_1, \gamma_1, L_1), (C_2, a_2, \gamma_2, L_2)$ the *independent attribute method* will define $(C_1 \times C_2, a, \gamma, L_1 \times L_2)$

where

$\alpha : C_1 \times C_2 \to L_1 \times L_2$ and $\gamma : L_1 \times L_2 \to C_1 \times C_2$

$\alpha(\langle c_1, c_2 \rangle) = \langle \alpha_1(c_1), \alpha_2(c_2) \rangle$

$\gamma(\langle a_1, \alpha_2 \rangle) = \langle \gamma_1(a_1), \gamma_2(a_2) \rangle$

This is a Galois connection since

$$\alpha(c_1, c_2) \subseteq (l_1, l_2) \underset{\text{definition of } \alpha}{\Leftrightarrow} \langle \alpha_1(c_1), \alpha_2(c_2) \rangle \subseteq (l_1, l_2)$$

$$\underset{\text{definition of } \alpha_i}{\Leftrightarrow} \alpha_1(c_1) \subseteq l_1 \wedge \alpha_2(c_2) \subseteq l_2$$

$$\underset{\text{use the Galois connections given}}{\Leftrightarrow} c_1 \subseteq \gamma_1(l_1) \wedge c_2 \subseteq \gamma_2(l_2)$$

$$\Leftrightarrow \langle c_1, c_2 \rangle \subseteq \langle \gamma_1(l_1), \gamma_2(l_2) \rangle$$

$$\underset{\text{defintion of } \gamma}{\Leftrightarrow} \langle c_1, c_2 \rangle \subseteq \gamma(\langle l_1, l_2 \rangle)$$

define the abstract effect of elementary statements on L_1 and L_2 $L_1 \times L_2 \square st\square^{\#} : L_1 \times L_2 \to L_1 \times L_2$ from $L_1 \square st\square^{\#} : L_1 \to L_1$, $L_2 \square st\square^{\#} : L_2 \to L_2$. We want to preserve optimality (as much as possible) and soundness.

$L_1 \times L_2 \square st\square^{\#}(\langle a_1, a_2 \rangle) = \langle L_1 \square st\square^{\#}(a_1), L_2 \square st\square^{\#}(a_2) \rangle$.

This is an optimal solution and it is sound as well, but we don't use the dependencies between the lattices and information can be lost.

For instance:

We would like to analyze the sign of two integers.

We take the Galois connection $(P(Z), \alpha_{sign}, \gamma_{sign}, P(sign))$, using the independent attribute method we get $(P(Z) \times P(Z), \alpha_{SS}, \gamma_{SS}, P(sign) \times P(sign))$

$\alpha_{ss}(Z_1, Z_2) = (\{sign(z) \mid z \in Z_1\}, \{sign(z) \mid z \in Z_2\})$

$\gamma_{ss}(A_1, A_2) = (\{a \mid sign(a) \in A_1\}, \{a \mid sign(a) \in A_2\})$

The expression (x,-x) may have a value in $\{(z,-z) \mid z \in Z\}$, which is represented as (Z,Z), hence $\alpha_{SS}(Z,Z) = (\{+,0,-\},\{+,0,-\})$, and the information about the relationship between the 2 components is lost (A better representation would be $\{(+,-),(-,+),(0,0)\}$).

Relational Attribute Method

As previously seen, the independent attribute method allows no interplay between 2 pairs of abstractions and concretization functions and sometimes leads to imprecision. The *relational method attribute* allows interaction between the two components.

Given the Galois connections $(P(C_1), a_1, \gamma_1, L_1), (P(C_2), a_2, \gamma_2, L_2)$
where $\beta_1 : C_1 \to L_1, \beta_2 : C_2 \to L_2$ and $\alpha_i(X) = \cup\{\beta_i(c) \mid c \in X\}$ the *relational attribute method* will define a Galois connection $(P(C_1 \times C_2), a, \gamma, P(L_1 \times L_2))$

$$\alpha(\langle X_1, X_2 \rangle) = \{\langle \beta_1(c_1), \beta_2(c_2) \rangle \mid c_1 \in X_1, c_2 \in X_2\}$$
$$\gamma(\langle Y_1, Y_2 \rangle) = \{\langle c_1, c_2 \rangle \mid \beta_1(c_1) \in Y_1, \beta_2(c_2) \in Y_2\}$$

We will show how this method works on the previous example:

The Galois connection defined is $(P(Z \times Z), a_{SS'}, \gamma_{SS'}, P(sign \times sign))$

$$\alpha_{SS'}(\langle X_1, X_2 \rangle) = \{\langle sign(x_1), sign(x_2) \rangle \mid x_1 \in X_1, x_2 \in X_2\}$$
$$\gamma_{SS'}(\langle Y_1, Y_2 \rangle) = \{\langle z_1, z_2 \rangle \mid sign(z_1) \in Y_1, sign(z_2) \in Y_2\}$$

The expression (x,-x) may have a value in $\{(z,-z) \mid z \in Z\}$, which is an element of $P(Z \times Z)$ and it is described by the set $\alpha_{SS'}(\{(z,-z) \mid z \in Z\}) = \{(-,+),(0,0),(+,-)\}$.

The information about the relationship between (x,-x) is preserved.

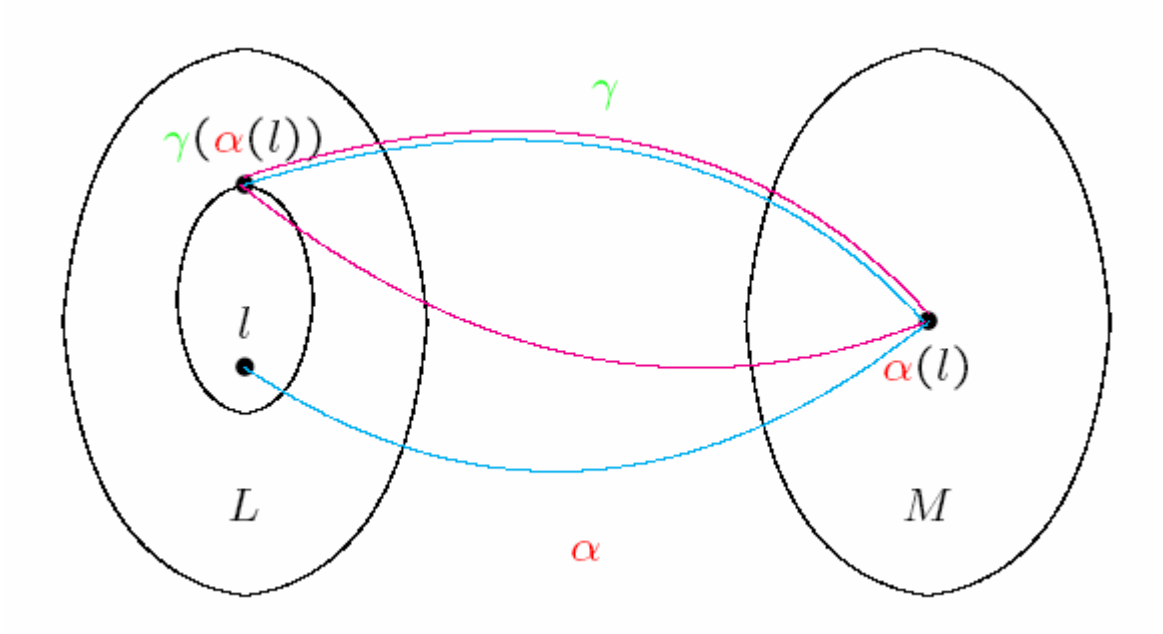
The problem is that the lattices grow to an exponential size, and can also be infinite.

Semantic Reduction

Galois Insertions

When looking at a Galois connection (C,a,γ,L), there may be several elements of L (the abstract domain) that describe the same element in the concrete domain. This means that the concrete domain describes some "irrelevant" elements.

A Galois Insertion is a Galois Connection satisfying $\alpha(\gamma(a))=a$ for all $a\in L$.



The following claims are equivalent:

if (C,a,γ,L) is a Galois insertion

1. α is a surjective function: $\forall a\in A\ \exists c\in C\ s.t.\ \alpha(c)=a$
2. γ is injective: $\gamma(a_1)=\gamma(a_2)\Rightarrow a_1=a_2$
3. γ is an order similarity: $\forall a_1,a_2\in L\ \gamma(a_1)\subseteq\gamma(a_2)\Leftrightarrow a_1\subseteq a_2$

Proof:

2:

$$\gamma(a_1)=\gamma(a_2)\Rightarrow\alpha(\gamma(a_1))=\alpha(\gamma(a_2)) \underbrace{\Rightarrow}_{\text{Galois Ins. Property}} a_1=a_2$$

2→ 1:

$\underbrace{\gamma\left(\alpha\left(\gamma\left(a\right)\right)\right)=\gamma\left(a\right)}_{\text{Property of Galois Connections}}\Rightarrow\alpha\left(\gamma\left(a\right)\right)=a$, so for all a in L $\gamma\left(a\right)$ is the element in C for which $\alpha\left(c\right)=a$.

1→3:

γ is monotone, hence $\forall a_1,a_2\in L\ \gamma\left(a_1\right)\subseteq\gamma\left(a_2\right)\Leftarrow a_1\subseteq a_2$.

Suppose $\gamma\left(a_1\right)\subseteq\gamma\left(a_2\right)$. By monotonicity of α : $\alpha\left(\gamma\left(a_1\right)\right)\subseteq\alpha\left(\gamma\left(a_2\right)\right)$

From 1: $\alpha\left(\gamma\left(a_1\right)\right)\subseteq\alpha\left(\gamma\left(a_2\right)\right)\Rightarrow a_1\subseteq a_2$

4→1

$\alpha\left(\gamma\left(a_1\right)\right)\subseteq a_2\Leftrightarrow\gamma\left(a_1\right)\subseteq\gamma\left(a_2\right)\Leftrightarrow a_1\subseteq a_2$

Not all Galois connections are Galois insertions. For instance (sign X parity) isn't, since the property (0, odd) doesn't describe any element in the concrete world.

Reduction Operator

The second mechanism used to increase precision.

Improve the precision of the analysis by recovering properties of the program semantics.

Consider a Galois connection $\left(C,\alpha,\gamma,L\right)$.

An operation $op:A\rightarrow A$ is a semantic reduction if:

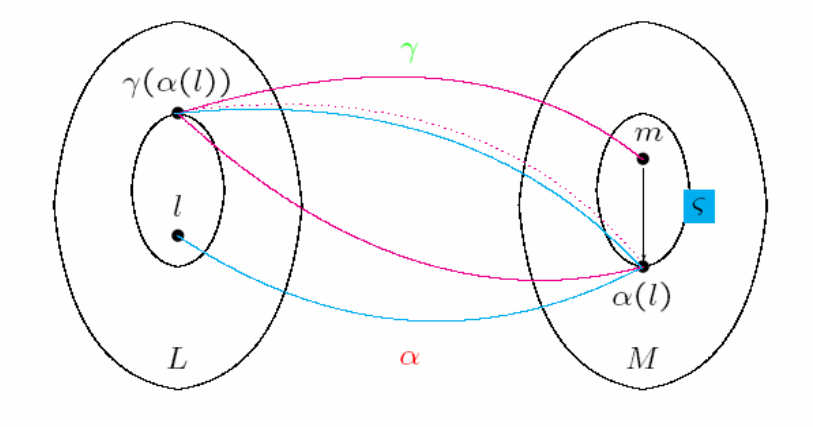
$$\forall a\in A:op\left(a\right)\subseteq a\quad and\quad\gamma\left(op\left(a\right)\right)=\gamma\left(a\right)$$

Any Galois connection can be "forced" to be a Galois insertion using a reduction operator. This operator will for the concretization function to be injective.

The reduction operator can be defined to be
$op:L\rightarrow L\qquad op\left(a\right)=\sqcup\left\{a'\mid\gamma\left(a\right)=\gamma\left(a'\right)\right\}$

Define: $op[L]=\left(\left\{op\left(a\right)\mid a\in L\right\}\right)$. $\left(C,\alpha,\gamma,op\left[L\right]\right)$ is a Galois insertion.

The operator can be applied before and after basic operations and it preserves soundness.



The previous example (parity X sign) can be forced into a Galois insertion by defining $L' = sign\_parity[Z] = \{(-, odd), (-, even), (0, even), (+, odd), (+, even)\}$.

Conclusions

What makes a static analysis good?

- Precise enough
- Efficient enough

To obtain the goal we need to define a good domain. Leaving out non important details, know which concrete data need to be represented (and what doesn't). The abstract interpreters should be precise and efficient, to make an easy implementation the join operation should be efficient and use widening (or small height lattices).

The theory is well founded

- Abstraction
- Soundness
- Chaotic iterations
- Elimination methods
- Modular parts

It does have some weak parts

- Transformations
- Predictable approximations
- User defined abstractions
- System

References

[1] Patrick Cousot & 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, 1977. ACM Press, New York, NY, USA.

[2] http://en.wikipedia.org/wiki/Abstract_interpretation

[3] Chris. Hankin, Hanne Riis Nielson, Flemming Nielson. Principles of Program Analysis, Published 1999, Springer.

[4] www2.imm.dtu.dk/~riis/PPA/slides4.pdf