Principles of program analysis (2. corr. print).

Book · January 2005

Source: DBLP

CITATIONS
113

READS
1,402

3 authors, including:

Chris Hankin
Imperial College London
188 PUBLICATIONS **4,737** CITATIONS

SEE PROFILE

Principles of Program Analysis

Chris Hankin

thanks to:

Flemming Nielson and Hanne Riis Nielson

Imperial College London

Introduction and Fixed Points

Program analysis is an automatic technique for finding out properties of programs without having to run them.

- Optimising compilers
- Automated program verification
- Security

Some techniques:

- Data Flow Analysis
- Control Flow Analysis
- Types and Effects Systems
- Abstract Interpretation

Book: *Principles of Program Analysis* by F. Nielson, H.R. Nielson and C. Hankin, Springer Verlag, 2005 (2nd corrected printing).

A first example:

$$[\textbf{input}\ n]^1;$$

$$[m := 2]^2;$$

$$\texttt{while}\ [n > 1]^3\ \texttt{do}$$

$$\quad [m := m \times n]^4;$$

$$\quad [n := n - 1]^5;$$

$$[\textbf{output}\ m]^6;$$

We can statically determine that the value of *m* at statement 6 will be even for any input *n*. A program analysis can determine this by propagating parity information *forwards* from the start of the program.

We can assign one of three properties to each variable:

- even – the value is known to be even
- odd – the value is known to be odd
- unknown – the parity of the value is unknown

(Take care of loop)

1:	m: unknown	n: unknown
2:	m: unknown	n: unknown
3:	m: even	n: unknown
4:	m: even	n: unknown
5:	m: even	n: unknown
6:	m: even	n: unknown

The program computes 2 times the factorial of n for any positive value of n. Replacing statement 2 by:

$$[m := 1]^2;$$

gives a program that computes factorials but then the program analysis is unable to tell us anything about the parity of m at statement 6.

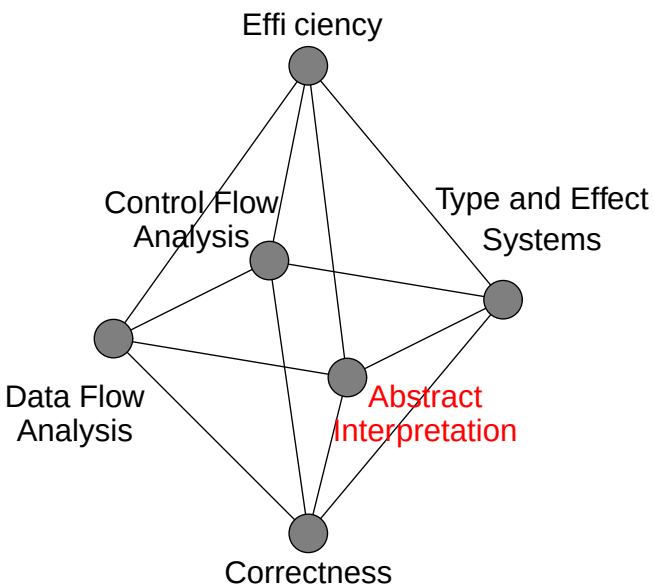
This is correct because m could be even or odd. However, even if we fix the input to be positive and even, by some suitable conditional assignment, the program analysis will still not accurately predict the evenness of m at statement 6.

This loss of accuracy is a common feature of program analyses: many properties that we are interested in are essentially *undecidable* and therefore we cannot hope to detect them accurately. We have to ensure that the answers from program analysis are at least *safe*.

- **yes** means definitely yes, and
- **no** means possibly no.

In the modified factorial program, it is safe to say that the parity of m is unknown at 6 – it would not be safe to say that m is even.

Overview



- Introduction – two approaches to correctness
- Fixed Points – widening and narrowings
- Galois Connections
- Induced operations

Abstract interpretation invented by Patrick Cousot and Radhia Cousot in 1977.

- strictness analysis (Mycroft, 1981; Burn, Hankin and Abramsky, 1985)
- many applications in logic programming
- general semantics-based framework applied to many paradigms

We will give a language and semantics independent treatment.

A first-order approach

To set the scene, imagine some programming language. Its semantics identifies some set V of values (like states, closures, double precision reals) and specifies how a program p transforms one value v_1 to another v_2; we may write

$$p \vdash v_1 \leadsto v_2$$

In a similar way, a program analysis identifies the set L of properties (like shapes of states, abstract closures, lower and upper bounds for reals) and specifies how a program p transforms one property l_1 to another l_2:

$$p \vdash l_1 \rhd l_2$$

It is customary to require $\rhd$ to be deterministic and thereby define a function; this will allow us to write $f_p(l_1) = l_2$ to mean $p \vdash l_1 \rhd l_2$.

Every program analysis should be correct with respect to the semantics. For a class of (so-called first-order) program analyses this is established by directly relating properties to values using a correctness relation:

$$R : V \times L \to \{\mathit{true}, \mathit{false}\}$$

The intention is that $v\ R\ l$ formalises our claim that the value v is described by the property l.

To be useful one has to prove that the correctness relation R is preserved under computation: if the relation holds between the initial value and the initial property then it also holds between the final value and the final property. This may be formulated as the implication

$$v_1\ R\ l_1\ \wedge\ p \vdash v_1 \rightsquigarrow v_2\ \wedge\ p \vdash l_1 \rhd l_2 \quad \Rightarrow \quad v_2\ R\ l_2$$

A relation R satisfying a condition like this is often called a logical relation and the implication is sometimes written $(p \vdash \cdot \rightsquigarrow \cdot)(R \twoheadrightarrow R)(p \vdash \cdot \rhd \cdot)$.

The theory of Abstract Interpretation comes to life when we augment the set of properties L with a preorder structure and relate this to the correctness relation R. The most common scenario is when $L = (L, \sqsubseteq, \sqcup, \sqcap, \bot, \top)$ is a complete lattice with partial ordering $\sqsubseteq$. We then impose the following relationship between R and L:

$$v \; R \; l_1 \;\wedge\; l_1 \sqsubseteq l_2 \quad \Rightarrow \quad v \; R \; l_2 \tag{1}$$

$$(\forall l \in L' \subseteq L : v \; R \; l) \quad \Rightarrow \quad v \; R \; (\bigsqcap L') \tag{2}$$

$$v \; R \; l_1 \;\wedge\; l_1 \sqsubseteq l_2 \Rightarrow v \; R \; l_2$$

- The first condition says that the smaller the property is with respect to the partial ordering, the better (i.e. more precise) it is.
- This is an “arbitrary” decision in the sense that we could instead have decided that the larger the property is, the better it is, as is indeed the case in much of the literature on Data Flow Analysis; luckily the principle of duality from lattice theory tells us that this difference is only a cosmetic one.

$$(\forall l \in L' \subseteq L : v\ R\ l) \Rightarrow v\ R\ (\bigsqcap L')$$

- The second condition says that there is always a best property for describing a value. This is important for having to perform only one analysis (using the best property, i.e. the greatest lower bound of the candidates) instead of several analyses (one for each of the candidates).

- The condition has two immediate consequences:

$$v\ R\ \top$$

$$v\ R\ l_1\ \wedge\ v\ R\ l_2 \quad \Rightarrow \quad v\ R\ (l_1 \sqcap l_2)$$

Representation functions

An alternative approach to the use of a correctness relation $R : V \times L \to \{\mathit{true}, \mathit{false}\}$ between values and properties is to use a representation function:

$$\beta : V \to L$$

The idea is that β maps a value to the best property describing it. The correctness criterion for the analysis will then be formulated as follows:

$$\beta(v_1) \sqsubseteq l_1\ \wedge\ p \vdash v_1 \rightsquigarrow v_2\ \wedge\ p \vdash l_1 \rhd l_2 \quad \Rightarrow \quad \beta(v_2) \sqsubseteq l_2$$

Thus the idea is that if the initial value v_1 is safely described by l_1 then the final value v_2 will be safely described by the result l_2 of the analysis.

We first show how to define a correctness relation R_β from a given representation function β:

$$v \; R_\beta \; l \quad \text{iff} \quad \beta(v) \sqsubseteq l$$

Next we show how to define a representation function β_R from a correctness relation R:

$$\beta_R(v) \quad = \quad \bigsqcap \{l \mid v \; R \; l\}$$

A modest generalisation

We shall conclude this section by performing a modest generalisation of the development performed so far. A program p specifies how one value v_1 is transformed into another value v_2:

$$p \vdash v_1 \rightsquigarrow v_2$$

Here $v_1 \in V_1$ and $v_2 \in V_2$ and we shall subsequently refrain from imposing the condition that $V_1 = V_2$; thus we shall allow the programs to have different "argument" and "result" types – for example, this will be the case for most functional programs.

The analysis of p specifies how a property l_1 is transformed into a property l_2:

$$p \vdash l_1 \rhd l_2$$

Here $l_1 \in L_1$ and $l_2 \in L_2$ and again we shall refrain from imposing the restriction that $L_1 = L_2$.

Turning to the correctness conditions we shall now assume that we have two correctness relations, one for V_1 and L_1 and one for V_2 and L_2:

$$R_1 : V_1 \times L_1 \to \{\mathit{true}, \mathit{false}\} \textit{ generated by } \beta_1 : V_1 \to L_1$$

$$R_2 : V_2 \times L_2 \to \{\mathit{true}, \mathit{false}\} \textit{ generated by } \beta_2 : V_2 \to L_2$$

Correctness of f_p now amounts to

$$v_1 \; R_1 \; l_1 \;\; \wedge \;\; p \vdash v_1 \rightsquigarrow v_2 \quad \Rightarrow \quad v_2 \; R_2 \; f_p(l_1)$$

for all $v_1 \in V_1$, $v_2 \in V_2$ and $l_1 \in L_1$. Using the concept of logical relations (briefly mentioned above) this can be written as:

$$(p \vdash \cdot \rightsquigarrow \cdot) \, (R_1 \twoheadrightarrow R_2) \, f_p$$

To be precise, $\rightsquigarrow \; (R_1 \twoheadrightarrow R_2) \, f$ means that:

$$\forall v_1, v_2, l_1 : v_1 \rightsquigarrow v_2 \;\; \wedge \;\; v_1 \; R_1 \; l_1 \;\; \Rightarrow \;\; v_2 \; R_2 \; f(l_1)$$

Higher-order formulation.

The representation function β can be defined from the representation functions β_1 and β_2 and it will be denoted $\beta_1 \twoheadrightarrow \beta_2$:

$$(\beta_1 \twoheadrightarrow \beta_2)(\rightsquigarrow) = \lambda l_1 . \bigsqcup \{\beta_2(v_2) \mid \beta_1(v_1) \sqsubseteq l_1 \;\; \wedge \;\; v_1 \rightsquigarrow v_2\}$$

Consider the program `plus` with the semantics given by

$$\texttt{plus} \vdash (z_1, z_2) \rightsquigarrow z_1 + z_2$$

where $z_1, z_2 \in \mathbf{Z}$. A very precise analysis might use the complete lattices $(\mathcal{P}(\mathbf{Z}), \subseteq)$ and $(\mathcal{P}(\mathbf{Z} \times \mathbf{Z}), \subseteq)$ as follows:

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

where $ZZ \subseteq \mathbf{Z} \times \mathbf{Z}$.

Consider now the correctness relations R_{Z} and $R_{\mathrm{Z}\times\mathrm{Z}}$ *generated by* the representation functions:

$$\begin{aligned} \beta_{\mathrm{Z}}(z) &= \{z\} \\ \beta_{\mathrm{Z}\times\mathrm{Z}}(z_1, z_2) &= \{(z_1, z_2)\} \end{aligned}$$

The correctness of the analysis of `plus` can now be expressed by

$$\begin{aligned} &\forall z_1, z_2, z, ZZ : \texttt{plus} \vdash (z_1, z_2) \rightsquigarrow z \\ &\quad \wedge\ (z_1, z_2)\ R_{\mathrm{Z}\times\mathrm{Z}}\ ZZ\ \Rightarrow\ z\ R_{\mathrm{Z}}\ f_{\mathrm{plus}}(ZZ) \end{aligned}$$

or more succinctly

$$(\texttt{plus} \vdash \cdot \rightsquigarrow \cdot)\ (R_{\mathrm{Z}\times\mathrm{Z}} \twoheadrightarrow R_{\mathrm{Z}})\ f_{\mathrm{plus}}$$

The representation function $\beta_{\mathrm{Z}\times\mathrm{Z}} \twoheadrightarrow \beta_{\mathrm{Z}}$ satisfies

$$(\beta_{\mathrm{Z}\times\mathrm{Z}} \twoheadrightarrow \beta_{\mathrm{Z}})(p \vdash \cdot \rightsquigarrow \cdot) = \lambda ZZ.\{z \mid (z_1, z_2) \in ZZ \wedge p \vdash (z_1, z_2) \rightsquigarrow z\}$$

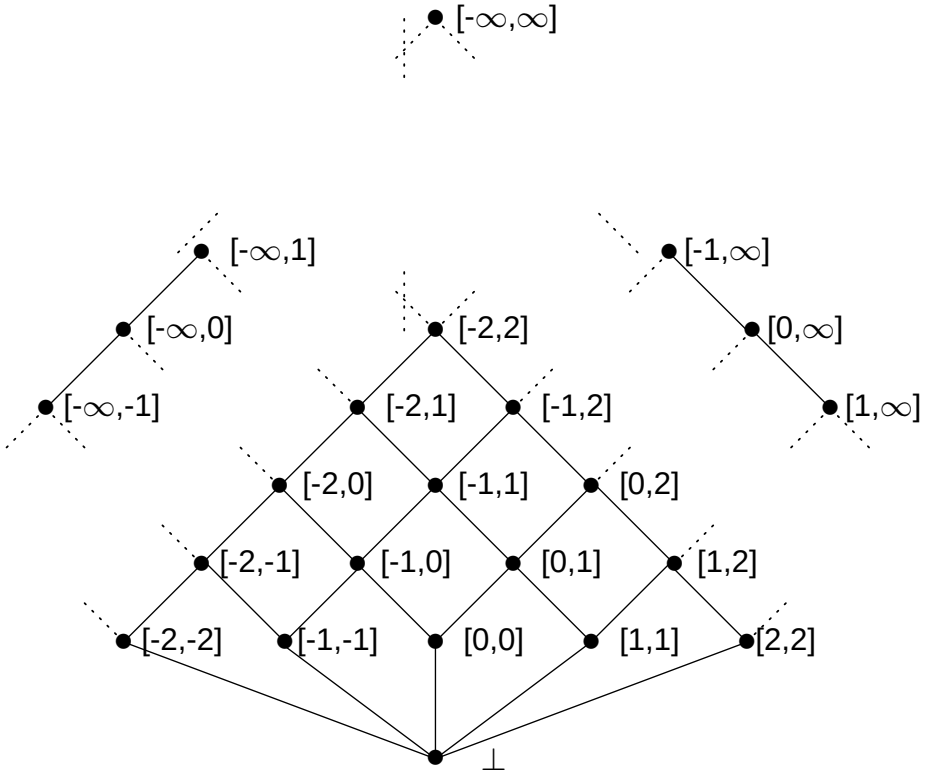
so the correctness can also be expressed as $(\beta_{\mathrm{Z}\times\mathrm{Z}} \twoheadrightarrow \beta_{\mathrm{Z}})(\mathtt{plus} \vdash \cdot \rightsquigarrow \cdot) \sqsubseteq f_{\mathrm{plus}}$.

The above example illustrates how the abstract concepts can give a more succinct formulation of the correctness of the analysis. In the following we shall see several cases where we move freely between what we may call a “concrete” formulation of a property and an “abstract” formulation of the same property. And we shall see that the latter often will allow us to reuse general results so that we do not have to redevelop parts of the theory for each application considered.

We shall now present a complete lattice that may be used for Array Bound Analysis, i.e. for determining if an array index is always within the bounds of the array – if this is the case then a number of run-time checks can be eliminated.
The lattice $(\textbf{Interval}, \sqsubseteq)$ of intervals over **Z** may be described as follows.
The elements are

$$\textbf{Interval} = \{\bot\} \cup \{[z_1, z_2] \mid z_1 \leq z_2, z_1 \in \mathbf{Z} \cup \{-\infty\}, z_2 \in \mathbf{Z} \cup \{\infty\}\}$$

- The ordering $\leq$ on **Z** is extended to an ordering on $\mathbf{Z}' = \mathbf{Z} \cup \{-\infty, \infty\}$ by setting $-\infty \leq z$, $z \leq \infty$, and $-\infty \leq \infty$ (for all $z \in \mathbf{Z}$).
- Intuitively, $\bot$ denotes the empty interval and $[z_1, z_2]$ is the interval from z_1 to z_2 including the end points if they are in **Z**.
- We shall use *int* to range over elements of **Interval**.
- Intuitively $\mathit{int}_1 \sqsubseteq \mathit{int}_2$ means that $\{z \mid z \in \mathit{int}_1\} \subseteq \{z \mid z \in \mathit{int}_2\}$



To give a succinct definition of the partial ordering we define the infimum and supremum operations on intervals as follows:

$$\inf(\mathit{int}) \quad = \quad \begin{cases} \infty & \text{if } \mathit{int} = \bot \\ z_1 & \text{if } \mathit{int} = [z_1, z_2] \end{cases}$$

$$\sup(\mathit{int}) \quad = \quad \begin{cases} -\infty & \text{if } \mathit{int} = \bot \\ z_2 & \text{if } \mathit{int} = [z_1, z_2] \end{cases}$$

This allows us to define:

$$\mathit{int}_1 \sqsubseteq \mathit{int}_2 \quad \text{iff} \quad \inf(\mathit{int}_2) \leq \inf(\mathit{int}_1) \wedge \sup(\mathit{int}_1) \leq \sup(\mathit{int}_2)$$

- Given a complete lattice $L = (L, \sqsubseteq, \sqcup, \sqcap, \bot, \top)$ the effect of a program, p, in transforming one property, l_1, into another, l_2, i.e. $p \vdash l_1 \rhd l_2$, is normally given by an equation

$$f(l_1) = l_2$$

  for a monotone function $f : L \to L$ dependent on the program p.

- Note that the demand that f is monotone is very natural for program analysis; it merely says that if l_1' describes at least the values that l_1 does then also $f(l_1')$ describes at least the values that $f(l_1)$ does.

- For recursive or iterative program constructs we ideally want to obtain the least fixed point, *lfp*(f), as the result of a *finite* iterative process.

- However, the iterative sequence $(f^n(\bot))_n$ need *not* eventually stabilise nor need its least upper bound necessarily equal *lfp*(f).

- This might suggest considering the iterative sequence $(f^n(\top))_n$ and, even when it does not eventually stabilise, we can always terminate the iteration at an arbitrary point in time. While this is safe it turns out to be grossly imprecise in practice.

- A fixed point of f is an element $l \in L$ such that $f(l) = l$ and we write

$$\mathit{Fix}(f) = \{l \mid f(l) = l\}$$

  for the set of fixed points.
- The function f is reductive at l if and only if $f(l) \sqsubseteq l$ and we write

$$\mathit{Red}(f) = \{l \mid f(l) \sqsubseteq l\}$$

  for the set of elements upon which f is reductive; we shall say that f itself is reductive if $\mathit{Red}(f) = L$.
- Similarly, the function f is extensive at l if and only if $f(l) \sqsupseteq l$ and we write

$$\mathit{Ext}(f) = \{l \mid f(l) \sqsupseteq l\}$$

  for the set of elements upon which f is extensive; we shall say that f itself is extensive if $\mathit{Ext}(f) = L$.

Since L is a complete lattice it is always the case that the set $\mathit{Fix}(f)$ will have a greatest lower bound in L and we denote it by $\mathit{lfp}(f)$; this is actually the least fixed point of f because Tarski's Theorem ensures that:

$$\mathit{lfp}(f) = \bigsqcap \mathit{Fix}(f) = \bigsqcap \mathit{Red}(f) \in \mathit{Fix}(f) \subseteq \mathit{Red}(f)$$

Similarly, the set $\mathit{Fix}(f)$ will have a least upper bound in L and we denote it by $\mathit{gfp}(f)$; this is actually the greatest fixed point of f because Tarski's Theorem ensures that:

$$\mathit{gfp}(f) = \bigsqcup \mathit{Fix}(f) = \bigsqcup \mathit{Ext}(f) \in \mathit{Fix}(f) \subseteq \mathit{Ext}(f)$$

In Denotational Semantics it is customary to iterate to the least fixed point by taking the least upper bound of the sequence $(f^n(\bot))_n$. However, we have not imposed any continuity requirements on f (e.g. that $f(\bigsqcup_n l_n) = \bigsqcup_n (f(l_n))$ for all ascending chains $(l_n)_n$) and consequently we cannot be sure to actually reach the fixed point. In a similar way one could consider the greatest lower bound of the sequence $(f^n(\top))_n$. One can show that

$$f^n(\bot) \sqsubseteq \bigsqcup\nolimits_n f^n(\bot) \sqsubseteq \mathit{lfp}(f) \sqsubseteq$$
$$\mathit{gfp}(f) \sqsubseteq \bigsqcap\nolimits_n f^n(\top) \sqsubseteq f^n(\top)$$

indeed all inequalities (i.e. $\sqsubseteq$) can be strict (i.e. $\neq$).

Widenings

Since we cannot guarantee that the iterative sequence $(f^n(\bot))_n$ eventually stabilises nor that its least upper bound necessarily equals $\mathit{lfp}(f)$, we must consider another way of approximating $\mathit{lfp}(f)$.

The idea is now to replace it by a new sequence $(f^n_\nabla)_n$ that is known to eventually stabilise and to do so with a value that is a safe (upper) approximation of the least fixed point. The construction of the new sequence is parameterised on the operator ∇, called a widening operator; the precision of the approximated fixed point as well as the cost of computing it depends on the actual choice of widening operator.

In preparation for the development, an operator $\check{\sqcup} : L \times L \to L$ on a complete lattice $L = (L, \sqsubseteq)$ is called an upper bound operator if

$$l_1 \sqsubseteq (l_1 \mathbin{\check{\sqcup}} l_2) \sqsupseteq l_2$$

for all $l_1, l_2 \in L$, i.e. it always returns an element larger than both its arguments. Note that we do *not* require $\check{\sqcup}$ to be monotone, commutative, associative, nor absorptive (i.e. that $l \mathbin{\check{\sqcup}} l = l$).

Let $(l_n)_n$ be a sequence of elements of L and let $\phi : L \times L \to L$ be a total function on L. We shall now use ϕ to construct a new sequence $(l_n^\phi)_n$ defined by:

$$l_n^\phi = \begin{cases} l_n & \text{if } n = 0 \\ l_{n-1}^\phi \mathbin{\phi} l_n & \text{if } n > 0 \end{cases}$$

The following result expresses that any sequence can be turned into an ascending chain by an upper bound operator:

If $(l_n)_n$ is a sequence and $\check{\sqcup}$ is an upper bound operator then $(l_n^{\check{\sqcup}})_n$ is an ascending chain; furthermore $l_n^{\check{\sqcup}} \sqsupseteq \bigsqcup\{l_0, l_1, \cdots, l_n\}$ for all n.

Consider the complete lattice $(\textbf{Interval}, \sqsubseteq)$ and let *int* be an arbitrary but fixed element of **Interval**. Consider the following operator $\check{\sqcup}^{int}$ defined on **Interval**:

$$\mathit{int}_1 \check{\sqcup}^{int} \mathit{int}_2 = \begin{cases} \mathit{int}_1 \sqcup \mathit{int}_2 & \text{if } \mathit{int}_1 \sqsubseteq \mathit{int} \ \vee \ \mathit{int}_2 \sqsubseteq \mathit{int}_1 \\ [-\infty, \infty] & \text{otherwise} \end{cases}$$

Note that the operation is not symmetric: for $\mathit{int} = [0, 2]$ we e.g. have $[1, 2]\check{\sqcup}^{int}[2, 3] = [1, 3]$ whereas $[2, 3]\check{\sqcup}^{int}[1, 2] = [-\infty, \infty]$.

It is immediate that $\check{\sqcup}^{int}$ is an upper bound operator. Consider now the sequence:

$$[0, 0], [1, 1], [2, 2], [3, 3], [4, 4], [5, 5], \cdots$$

If $\mathit{int} = [0, \infty]$, then the upper bound operator will transform the above sequence into the ascending chain:

$$[0, 0], [0, 1], [0, 2], [0, 3], [0, 4], [0, 5], \cdots$$

However, if $\mathit{int} = [0, 2]$, then we will get the following ascending chain

$$[0, 0], [0, 1], [0, 2], [0, 3], [-\infty, \infty], [-\infty, \infty], \cdots$$

which eventually stabilises.

We can now introduce a special class of upper bound operators that will help us to approximate the least fixed points: An operator $\nabla : L \times L \to L$ is a widening operator if and only if:

- it is an upper bound operator, and
- for all ascending chains $(l_n)_n$ the ascending chain $(l_n^\nabla)_n$ eventually stabilises.

The idea is as follows: Given a monotone function $f : L \to L$ on a complete lattice L and given a widening operator ∇ on L, we shall calculate the sequence $(f_\nabla^n)_n$ defined by

$$f_\nabla^n = \begin{cases} \bot & \text{if } n = 0 \\ f_\nabla^{n-1} & \text{if } n > 0 \ \wedge \ f(f_\nabla^{n-1}) \sqsubseteq f_\nabla^{n-1} \\ f_\nabla^{n-1} \ \nabla \ f(f_\nabla^{n-1}) & \text{otherwise} \end{cases}$$

For any upper bound operator:

(i) the sequence $(f_\nabla^n)_n$ is an ascending chain;

(ii) if $f(f_\nabla^m) \sqsubseteq f_\nabla^m$ for some m then the sequence $(f_\nabla^n)_n$ eventually stabilises and furthermore $\forall n > m : f_\nabla^n = f_\nabla^m$ and $\bigsqcup_n f_\nabla^n = f_\nabla^m$;

(iii) if $(f_\nabla^n)_n$ eventually stabilises then there exists an m such that $f(f_\nabla^m) \sqsubseteq f_\nabla^m$; and

(iv) if $(f_\nabla^n)_n$ eventually stabilises then $\bigsqcup_n f_\nabla^n \sqsupseteq \mathit{lfp}(f)$.

- Moreover, if ∇ is a widening operator then the ascending chain $(f_\nabla^n)_n$ eventually stabilises.

Consider the complete lattice $(\textbf{Interval}, \sqsubseteq)$. Let K be a *finite* set of integers, e.g. the set of integers explicitly mentioned in a given program. We shall now define a widening operator ∇_K based on K. The idea is that $[z_1, z_2] \; \nabla_K \; [z_3, z_4]$ is something like

$$[\; \mathsf{LB}(z_1, z_3) \; , \mathsf{UB}(z_2, z_4) \;]$$

where $\mathsf{LB}(z_1, z_3) \in \{z_1\} \cup K \cup \{-\infty\}$ is the best possible lower bound and $\mathsf{UB}(z_2, z_4) \in \{z_2\} \cup K \cup \{\infty\}$ is the best possible upper bound. In this way a change in any of the bounds of the interval $[z_1, z_2]$ can only take place in a finite number of steps (corresponding to the elements of K).

For the precise definition we let $z_i \in \mathbf{Z}' = \mathbf{Z} \cup \{-\infty, \infty\}$ and write:

$$\mathsf{LB}_K(z_1, z_3) = \begin{cases} z_1 & \text{if } z_1 \leq z_3 \\ k & \text{if } z_3 < z_1 \;\wedge\; k = \mathsf{max}\{k \in K \mid k \leq z_3\} \\ -\infty & \text{if } z_3 < z_1 \;\wedge\; \forall k \in K : z_3 < k \end{cases}$$

$$\mathsf{UB}_K(z_2, z_4) = \begin{cases} z_2 & \text{if } z_4 \leq z_2 \\ k & \text{if } z_2 < z_4 \;\wedge\; k = \mathsf{min}\{k \in K \mid z_4 \leq k\} \\ \infty & \text{if } z_2 < z_4 \;\wedge\; \forall k \in K : k < z_4 \end{cases}$$

We can now define $\nabla = \nabla_K$ by:

$$\mathit{int}_1 \nabla \mathit{int}_2 = \begin{cases} \bot & \text{if } \mathit{int}_1 = \mathit{int}_2 = \bot \\ [\ \mathsf{LB}_K(\mathsf{inf}(\mathit{int}_1), \mathsf{inf}(\mathit{int}_2)), \mathsf{UB}_K(\mathsf{sup}(\mathit{int}_1), \mathsf{sup}(\mathit{int}_2))\ ] & \\ & \text{otherwise} \end{cases}$$

As an example consider the ascending chain $(\mathit{int}_n)_n$:

$$[0,1], [0,2], [0,3], [0,4], [0,5], [0,6], [0,7], \cdots$$

and assume that $K = \{3, 5\}$. Then $(\mathit{int}_n^{\nabla})_n$ will be the chain

$$[0,1], [0,3], [0,3], [0,5], [0,5], [0,\infty], [0,\infty], \cdots$$

Narrowings

- Using the technique of widening we managed to arrive at an upper approximation f_{∇}^m of the least fixed point of f.
- However, we have $f(f_{\nabla}^m) \sqsubseteq f_{\nabla}^m$ so f is reductive at f_{∇}^m and this immediately suggests a way of improving the approximation by considering the iterative sequence $(f^n(f_{\nabla}^m))_n$.
- Since $f_{\nabla}^m \in \mathit{Red}(f)$ this will be a descending chain with $f^n(f_{\nabla}^m) \in \mathit{Red}(f)$ and hence $f^n(f_{\nabla}^m) \sqsupseteq \mathit{lfp}(f)$ for all n.
- Once again we have no reason to believe that this descending chain eventually stabilises although it is of course safe to stop at an arbitrary point.

An operator $\Delta : L \times L \to L$ is a narrowing operator if:

- $l_2 \sqsubseteq l_1 \;\Rightarrow\; l_2 \sqsubseteq (l_1 \;\Delta\; l_2) \sqsubseteq l_1$ for all $l_1, l_2 \in L$, and
- for all descending chains $(l_n)_n$ the sequence $(l_n^{\Delta})_n$ eventually stabilises.

Note that we do *not* require Δ to be monotone, commutative, associative or absorptive. One can show that $(l_n^{\Delta})_n$ is a descending chain when $(l_n)_n$ is a descending chain.

For f_{∇}^m satisfying $f(f_{\nabla}^m) \sqsubseteq f_{\nabla}^m$, i.e. $\mathit{lfp}_{\nabla}(f) = f_{\nabla}^m$, we now construct the sequence $([f]_{\Delta}^n)_n$ by

$$[f]_{\Delta}^n = \begin{cases} f_{\nabla}^m & \text{if } n = 0 \\ [f]_{\Delta}^{n-1} \;\Delta\; f([f]_{\Delta}^{n-1}) & \text{if } n > 0 \end{cases}$$

- If Δ is a narrowing operator and $f(f^m_\nabla) \sqsubseteq f^m_\nabla$ then $([f]^n_\Delta)_n$ is a descending chain in $\mathit{Red}(f)$ and

$$[f]^n_\Delta \sqsupseteq f^n(f^m_\nabla) \sqsupseteq \mathit{lfp}(f)$$

  for all n.
- If Δ is a narrowing operator and $f(f^m_\nabla) \sqsubseteq f^m_\nabla$ then the descending chain $([f]^n_\Delta)_n$ eventually stabilises.
- It is important to stress that narrowing operators are *not* the dual concept of widening operators. In particular, the sequence $(f^n_\nabla)_n$ may step outside $\mathit{Ext}(f)$ in order to end in $\mathit{Red}(f)$, whereas the sequence $([f]^n_\Delta)_n$ stays in $\mathit{Red}(f)$ all the time.

Consider the complete lattice $(\textbf{Interval}, \sqsubseteq)$. Basically there are two kinds of infinite descending chains in **Interval**: those with elements of the form $[-\infty, z]$ and those with elements of the form $[z, \infty]$ where $z \in \mathbf{Z}$. Consider an infinite sequence of the latter form; it will have elements

$$[z_1, \infty], [z_2, \infty], [z_3, \infty], \cdots$$

where $z_1 < z_2 < z_3 < \cdots$. The idea is now to define a narrowing operator Δ_N that will force the sequence to stabilise when $z_i \geq N$ for some fixed non-negative integer N. Similarly, for a descending chain with elements of the form $[-\infty, z_i]$ the narrowing operator will force it to stabilise when $z_i \leq -N$.

Formally, we shall define $\Delta = \Delta_N$ by

$$\mathit{int}_1 \;\Delta\; \mathit{int}_2 \quad = \quad \begin{cases} \bot & \text{if } \mathit{int}_1 = \bot \;\vee\; \mathit{int}_2 = \bot \\ [z_1, z_2] & \text{otherwise} \end{cases}$$

where

$$z_1 \quad = \quad \begin{cases} \inf(\mathit{int}_1) & \text{if } N < \inf(\mathit{int}_2) \wedge \sup(\mathit{int}_2) = \infty \\ \inf(\mathit{int}_2) & \text{otherwise} \end{cases}$$

$$z_2 \quad = \quad \begin{cases} \sup(\mathit{int}_1) & \text{if } \inf(\mathit{int}_2) = -\infty \wedge \sup(\mathit{int}_2) < -N \\ \sup(\mathit{int}_2) & \text{otherwise} \end{cases}$$

So consider e.g. the infinite descending chain $([n, \infty])_n$

$$[0, \infty], [1, \infty], [2, \infty], [3, \infty], [4, \infty], [5, \infty], \cdots$$

and assume that $N = 3$. Then the operator will give the sequence $([n, \infty]^{\Delta})_n$:

$$[0, \infty], [1, \infty], [2, \infty], [3, \infty], [3, \infty], [3, \infty], \cdots$$

Galois Connections

Galois connections

Sometimes calculations on a complete lattice L may be too costly or even uncomputable and this may motivate replacing L by a simpler lattice M.
An example is when L is the powerset of integers and M is a lattice of intervals.
To express the relationship between L and M it is customary to use an abstraction function

$$\alpha : L \to M$$

and a concretisation function

$$\gamma : M \to L$$

We shall write

$$(L, \alpha, \gamma, M)$$

or

$$L \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} M$$

We define (L, α, γ, M) to be a Galois connection between the complete lattices $(L, \sqsubseteq)$ and $(M, \sqsubseteq)$ if and only if

$\alpha : L \to M$ and $\gamma : M \to L$ are monotone functions

that satisfy:

$$\gamma \circ \alpha \sqsupseteq \lambda l.l$$

$$\alpha \circ \gamma \sqsubseteq \lambda m.m$$

Let $\mathcal{P}(\mathbf{Z}) = (\mathcal{P}(\mathbf{Z}), \subseteq)$ be the complete lattice of sets of integers and let **Interval** $=$ (**Interval**, $\sqsubseteq$) be the complete lattice of intervals. We shall now define a Galois connection

$$(\mathcal{P}(\mathbf{Z}), \alpha_{\mathrm{ZI}}, \gamma_{\mathrm{ZI}}, \textbf{Interval})$$

between $\mathcal{P}(\mathbf{Z})$ and **Interval**.
The concretisation function γ_{ZI} : **Interval** $\to \mathcal{P}(\mathbf{Z})$ is defined by

$$\gamma_{\mathrm{ZI}}(\mathit{int}) = \{z \in \mathbf{Z} \mid \mathsf{inf}(\mathit{int}) \leq z \leq \mathsf{sup}(\mathit{int})\}$$

Thus γ_{ZI} will extract the set of elements described by the interval, e.g. $\gamma_{\mathrm{ZI}}([0, 3]) = \{0, 1, 2, 3\}$ and $\gamma_{\mathrm{ZI}}([0, \infty]) = \{z \in \mathbf{Z} \mid z \geq 0\}$.

The abstraction function $\alpha_{\mathrm{ZI}} : \mathcal{P}(\mathbf{Z}) \to$ **Interval** is defined by

$$\alpha_{\mathrm{ZI}}(Z) = \begin{cases} \bot & \text{if } Z = \emptyset \\ [\inf'(Z), \sup'(Z)] & \text{otherwise} \end{cases}$$

Where, if $Z' = Z \cup \{-\infty, \infty\}$ and $\inf'$ and $\sup'$ are the corresponding infimum and supremum operators. For example, $\inf'(\emptyset) = \infty$, $\inf'(Z) = z'$ if $z' \in Z$ is the least element of Z, and $\inf'(Z) = -\infty$ otherwise.

Thus α_{ZI} will determine the smallest interval that includes all the elements of the set, e.g. $\alpha_{\mathrm{ZI}}(\{0, 1, 3\}) = [0, 3]$ and $\alpha_{\mathrm{ZI}}(\{2 * z \mid z > 0\}) = [2, \infty]$.

Adjunctions

We define (L, α, γ, M) to be an adjunction between complete lattices $L = (L, \sqsubseteq)$ and $M = (M, \sqsubseteq)$ if and only if

$$\alpha : L \to M \text{ and } \gamma : M \to L \text{ are total functions}$$

that satisfy

$$\alpha(l) \sqsubseteq m \quad \Leftrightarrow \quad l \sqsubseteq \gamma(m)$$

for all $l \in L$ and $m \in M$.

We shall now see that representation functions can be used to define Galois connections. So consider once again the representation function $\beta : V \to L$ mapping the values of V to the properties of the complete lattice L. It gives rise to a Galois connection

$$(\mathcal{P}(V), \alpha, \gamma, L)$$

between $\mathcal{P}(V)$ and L where the abstraction and concretisation functions are defined by

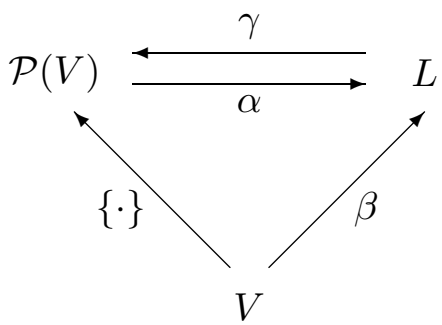
$$\begin{aligned} \alpha(V') &= \bigsqcup\{\beta(v) \mid v \in V'\} \\ \gamma(l) &= \{v \in V \mid \beta(v) \sqsubseteq l\} \end{aligned}$$

for $V' \subseteq V$ and $l \in L$.

Let us pause for a minute to see that this indeed defines an adjunction:

$$\begin{aligned} \alpha(V') \sqsubseteq l &\Leftrightarrow \bigsqcup\{\beta(v) \mid v \in V'\} \sqsubseteq l \\ &\Leftrightarrow \forall v \in V' : \beta(v) \sqsubseteq l \\ &\Leftrightarrow V' \subseteq \gamma(l) \end{aligned}$$

It is also immediate that $\alpha(\{v\}) = \beta(v)$ as illustrated by the diagram:



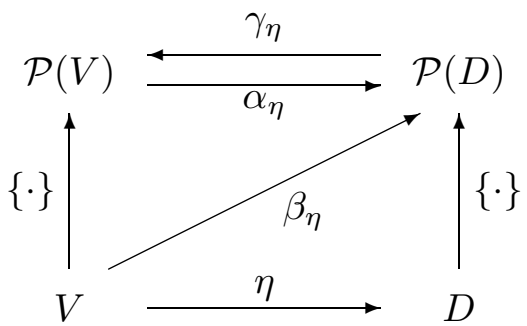
A special case of the above construction that is frequently useful is when $L = (\mathcal{P}(D), \subseteq)$ for some set D and we have an extraction function: $\eta : V \to D$.
We will then define the representation function $\beta_\eta : V \to \mathcal{P}(D)$ by $\beta_\eta(v) = \{\eta(v)\}$ and the Galois connection between $\mathcal{P}(V)$ and $\mathcal{P}(D)$ will now be written

$$(\mathcal{P}(V), \alpha_\eta, \gamma_\eta, \mathcal{P}(D))$$

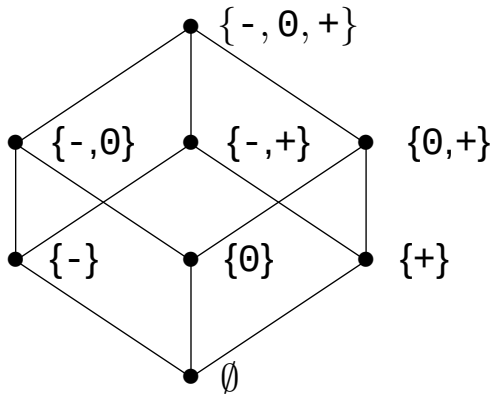
where

$$\begin{array}{lcll} \alpha_\eta(V') & = & \bigcup\{\beta_\eta(v) \mid v \in V'\} & = & \{\eta(v) \mid v \in V'\} \\ \gamma_\eta(D') & = & \{v \in V \mid \beta_\eta(v) \subseteq D'\} & = & \{v \mid \eta(v) \in D'\} \end{array}$$

The relationship between η, β_η, α_η and γ_η is illustrated by the diagram:



Let us consider the two complete lattices $(\mathcal{P}(\mathbf{Z}), \subseteq)$ and $(\mathcal{P}(\mathbf{Sign}), \subseteq)$ where $\mathbf{Sign} = \{\texttt{-}, \texttt{0}, \texttt{+}\}$.



The extraction function

$$\text{sign} : \mathbf{Z} \rightarrow \mathbf{Sign}$$

simply defines the signs of the integers and is specified by:

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

The above construction then gives us a Galois connection

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

with

$$\begin{aligned} \alpha_{\mathsf{sign}}(Z) &= \{\mathsf{sign}(z) \mid z \in Z\} \\ \gamma_{\mathsf{sign}}(S) &= \{z \in \mathbf{Z} \mid \mathsf{sign}(z) \in S\} \end{aligned}$$

where $Z \subseteq \mathbf{Z}$ and $S \subseteq \mathbf{Sign}$.

Properties of Galois Connections

If (L, α, γ, M) is a Galois connection then:

(i) α uniquely determines γ by $\gamma(m) = \bigsqcup\{l \mid \alpha(l) \sqsubseteq m\}$ and γ uniquely determines α by $\alpha(l) = \bigsqcap\{m \mid l \sqsubseteq \gamma(m)\}$.

(ii) α is completely additive and γ is completely multiplicative.

In particular $\alpha(\bot) = \bot$ and $\gamma(\top) = \top$.

If (L, α, γ, M) is a Galois connection then $\alpha \circ \gamma \circ \alpha = \alpha$ and $\gamma \circ \alpha \circ \gamma = \gamma$.

If there is a Galois connection (L, α, γ, M) between L and M then we can construct a correctness relation between V and M and a representation function from V to M.
Let us first focus on the correctness relation. So let
$R : V \times L \to \{\mathit{true}, \mathit{false}\}$ be a correctness relation. Further let (L, α, γ, M) be a Galois connection between the complete lattices L and M. It is then natural to define $S : V \times M \to \{\mathit{true}, \mathit{false}\}$ by

$$v\ S\ m \ \text{ iff } \ v\ R\ (\gamma(m))$$

Continuing the above line of reasoning assume now that R is *generated by* the representation function $\beta : V \to L$, i.e. $v\ R\ l \ \Leftrightarrow\ \beta(v) \sqsubseteq l$. Since (L, α, γ, M) is a Galois connection and hence an adjunction we may calculate

$$\begin{array}{rcl} v\ S\ m & \Leftrightarrow & v\ R\ (\gamma(m)) \\ & \Leftrightarrow & \beta(v) \sqsubseteq \gamma(m) \\ & \Leftrightarrow & (\alpha \circ \beta)(v) \sqsubseteq m \end{array}$$

showing that S is *generated by* $\alpha \circ \beta : V \to M$.

For a Galois connection (L, α, γ, M) there may be several elements of M that describe the same element of L, i.e. γ need not be injective, and this means that M may contain elements that are not relevant for the approximation of L.
The concept of Galois insertion is intended to rectify this: (L, α, γ, M) is a Galois insertion between the complete lattices $L = (L, \sqsubseteq)$ and $M = (M, \sqsubseteq)$ if and only if

$$\alpha : L \to M \text{ and } \gamma : M \to L \text{ are monotone functions}$$

that satisfy:

$$\begin{aligned} \gamma \circ \alpha &\sqsupseteq \lambda l.l \\ \alpha \circ \gamma &= \lambda m.m \end{aligned}$$

Returning to our earlier example,

$$(\mathcal{P}(\mathbf{Z}), \alpha_{\mathrm{ZI}}, \gamma_{\mathrm{ZI}}, \textbf{Interval})$$

is indeed a Galois insertion: we start with an interval, use γ_{ZI} to determine the set of integers it describes and next use α_{ZI} to determine the smallest interval containing this set and we get exactly the same interval as we started with.

For a Galois connection (L, α, γ, M) the following claims are equivalent:

(i) (L, α, γ, M) is a Galois insertion;

(ii) α is surjective: $\forall m \in M : \exists l \in L : \alpha(l) = m$;

(iii) γ is injective:
$\forall m_1, m_2 \in M : \gamma(m_1) = \gamma(m_2) \Rightarrow m_1 = m_2$; and

(iv) γ is an order-similarity: $\forall m_1, m_2 \in M :$
$$\gamma(m_1) \sqsubseteq \gamma(m_2) \Leftrightarrow m_1 \sqsubseteq m_2.$$

Consider the complete lattices $(\mathcal{P}(\mathbf{Z}), \subseteq)$ and $(\mathcal{P}(\mathbf{Sign} \times \textbf{Parity}), \subseteq)$ where $\mathbf{Sign} = \{\texttt{-}, \texttt{0}, \texttt{+}\}$ as before and $\textbf{Parity} = \{\text{odd}, \text{even}\}$. Define the extraction function signparity : $\mathbf{Z} \to \mathbf{Sign} \times \textbf{Parity}$ by:

$$\text{signparity}(z) = \begin{cases} (\text{sign}(z), \text{odd}) & \text{if } z \text{ is odd} \\ (\text{sign}(z), \text{even}) & \text{if } z \text{ is even} \end{cases}$$

This gives rise to a Galois connection $(\mathcal{P}(\mathbf{Z}), \alpha_{\text{signparity}}, \gamma_{\text{signparity}}, \mathcal{P}(\mathbf{Sign} \times \textbf{Parity}))$. The property $(\texttt{0}, \text{odd})$ describes no integers so clearly signparity is not surjective and we have an example of a Galois connection that is not a Galois insertion.

Given a Galois connection (L, α, γ, M) it is always possible to obtain a Galois insertion by enforcing that the concretisation function γ is injective. Basically, this amounts to removing elements from the complete lattice M using a reduction operator, $\varsigma : M \to M$, defined from the Galois connection.

Let (L, α, γ, M) be a Galois connection and define the reduction operator $\varsigma : M \to M$ by

$$\varsigma(m) = \bigsqcap \{m' \mid \gamma(m) = \gamma(m')\}$$

Then $\varsigma[M] = (\{\varsigma(m) \mid m \in M\}, \sqsubseteq_M)$ is a complete lattice and $(L, \alpha, \gamma, \varsigma[M])$ is a Galois insertion.

Systematic design of Galois connections

When developing a program analysis it is often useful to do so in stages: The starting point will typically be a complete lattice $(L_0, \sqsubseteq)$ fairly closely related to the semantics; an example is $(\mathcal{P}(V), \subseteq)$. We may then decide to use a more approximate set of properties and introduce the complete lattice $(L_1, \sqsubseteq)$ related to L_0 by a Galois connection $(L_0, \alpha_1, \gamma_1, L_1)$. This step can then be repeated any number of times: We replace one complete lattice L_i of properties with a more approximate complete lattice $(L_{i+1}, \sqsubseteq)$ related to L_i by a Galois connection $(L_i, \alpha_{i+1}, \gamma_{i+1}, L_{i+1})$. So the situation can be depicted as follows:

$$L_0 \underset{\alpha_1}{\overset{\gamma_1}{\leftrightarrows}} L_1 \underset{\alpha_2}{\overset{\gamma_2}{\leftrightarrows}} L_2 \underset{\alpha_3}{\overset{\gamma_3}{\leftrightarrows}} \cdots \underset{\alpha_k}{\overset{\gamma_k}{\leftrightarrows}} L_k$$

One of the components in the Array Bound Analysis is concerned with approximating the difference in magnitude between two numbers (typically the bound and the index). We shall proceed in two stages: First we shall approximate pairs (z_1, z_2) of integers by their difference in magnitude $|z_1| - |z_2|$ and next we shall further approximate this difference using a finite lattice. The two Galois connections will be defined by extraction functions and they will then be combined by taking their functional composition.

The first stage is specified by the Galois connection

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

where $\mathsf{diff} : \mathbf{Z} \times \mathbf{Z} \to \mathbf{Z}$ is the extraction function calculating the difference in magnitude:

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

The abstraction and concretisation functions α_{diff} and γ_{diff} will then be

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

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

The second stage is specified by the Galois connection

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

where **Range** $= \{$`<-1`, `-1`, `0`, `+1`, `>+1`$\}$. The extraction function range : $\mathbf{Z} \to$ **Range** clarifies the meaning of the elements of **Range**:

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

The abstraction and concretisation functions α_{range} and γ_{range} will then be

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

for $Z \subseteq \mathbf{Z}$ and $R \subseteq$ **Range**.
We then have that the functional composition

$$(\mathcal{P}(\mathbf{Z} \times \mathbf{Z}), \alpha_{\mathrm{R}}, \gamma_{\mathrm{R}}, \mathcal{P}(\textbf{Range}))$$

where $\alpha_{\mathrm{R}} = \alpha_{\mathsf{range}} \circ \alpha_{\mathsf{diff}}$ and $\gamma_{\mathrm{R}} = \gamma_{\mathsf{diff}} \circ \gamma_{\mathsf{range}}$, is a Galois connection.

We obtain the following formulae for the abstraction and concretisation functions:

$$\begin{aligned} \alpha_{\mathrm{R}}(ZZ) &= \{\mathsf{range}(|z_1| - |z_2|) \mid (z_1, z_2) \in ZZ\} \\ \gamma_{\mathrm{R}}(R) &= \{(z_1, z_2) \mid \mathsf{range}(|z_1| - |z_2|) \in R\} \end{aligned}$$

This is the Galois connection specified by the extraction function range ∘ diff :

$\mathbf{Z} \times \mathbf{Z} \to$ **Range**.

The final act

- The first techniques we shall consider are applicable when we have several analyses of *individual* components of a structure and we want to combine them into a single analysis.
- We shall then look at constructions for function spaces.
- Finally, we shall present techniques for combining several analyses of the same structure.

Independent Attribute Method

Let $(L_1, \alpha_1, \gamma_1, M_1)$ and $(L_2, \alpha_2, \gamma_2, M_2)$ be Galois connections. The independent attribute method will then give rise to a Galois connection

$$(L_1 \times L_2, \alpha, \gamma, M_1 \times M_2)$$

where:

$$\begin{aligned} \alpha(l_1, l_2) &= (\alpha_1(l_1), \alpha_2(l_2)) \\ \gamma(m_1, m_2) &= (\gamma_1(m_1), \gamma_2(m_2)) \end{aligned}$$

The Array Bound Analysis will contain a component that performs a Detection of Signs Analysis on pairs of integers. As a starting point, we take the Galois connection

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

specified by the extraction function sign.

It can be used to analyse both components of a pair of integers so using the independent attribute method we will get a Galois connection

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

where α_{SS} and γ_{SS} are given by

$$\begin{aligned} \alpha_{\mathrm{SS}}(Z_1, Z_2) &= (\{\mathsf{sign}(z) \mid z \in Z_1\}, \{\mathsf{sign}(z) \mid z \in Z_2\}) \\ \gamma_{\mathrm{SS}}(S_1, S_2) &= (\{z \mid \mathsf{sign}(z) \in S_1\}, \{z \mid \mathsf{sign}(z) \in S_2\}) \end{aligned}$$

where $Z_i \subseteq \mathbf{Z}$ and $S_i \subseteq \mathbf{Sign}$.

In general the independent attribute method often leads to imprecision. An expression like `(x,-x)` in the source language may have a value in $\{(z,-z) \mid z \in \mathbf{Z}\}$ but in the present setting where we use $\mathcal{P}(\mathbf{Z}) \times \mathcal{P}(\mathbf{Z})$ to represent sets of pairs of integers we cannot do better than representing $\{(z,-z) \mid z \in \mathbf{Z}\}$ by $(\mathbf{Z},\mathbf{Z})$ and hence the best property describing it will be $\alpha_{\mathrm{SS}}(\mathbf{Z},\mathbf{Z}) = (\{\texttt{-},\texttt{0},\texttt{+}\},\{\texttt{-},\texttt{0},\texttt{+}\})$. Thus we lose all information about the relative signs of the two components.

The Relational method

Let $(\mathcal{P}(V_1),\alpha_1,\gamma_1,\mathcal{P}(D_1))$ and $(\mathcal{P}(V_2),\alpha_2,\gamma_2,\mathcal{P}(D_2))$ be Galois connections. The relational method will give rise to the Galois connection

$$(\mathcal{P}(V_1 \times V_2),\alpha,\gamma,\mathcal{P}(D_1 \times D_2))$$

where

$$\begin{aligned} \alpha(VV) &= \bigcup\{\alpha_1(\{v_1\}) \times \alpha_2(\{v_2\}) \mid (v_1,v_2) \in VV\} \\ \gamma(DD) &= \{(v_1,v_2) \mid \alpha_1(\{v_1\}) \times \alpha_2(\{v_2\}) \subseteq DD\} \end{aligned}$$

where $VV \subseteq V_1 \times V_2$ and $DD \subseteq D_1 \times D_2$.

It is instructive to see how the relational method is simplified if the Galois connections $(\mathcal{P}(V_i), \alpha_i, \gamma_i, \mathcal{P}(D_i))$ are given by extraction functions $\eta_i : V_i \to D_i$, i.e. if $\alpha_i(V_i') = \{\eta_i(v_i) \mid v_i \in V_i'\}$ and $\gamma_i(D_i') = \{v_i \mid \eta_i(v_i) \in D_i'\}$. We then have

$$\begin{aligned} \alpha(VV) &= \{(\eta_1(v_1), \eta_2(v_2)) \mid (v_1, v_2) \in VV\} \\ \gamma(DD) &= \{(v_1, v_2) \mid (\eta_1(v_1), \eta_2(v_2)) \in DD\} \end{aligned}$$

which also can be obtained directly from the extraction function $\eta : V_1 \times V_2 \to D_1 \times D_2$ defined by $\eta(v_1, v_2) = (\eta_1(v_1), \eta_2(v_2))$.

The relational method can be used to construct a more precise analysis. We will now get a Galois connection

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

where $\alpha_{\mathrm{SS}'}$ and $\gamma_{\mathrm{SS}'}$ are given by

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

where $ZZ \subseteq \mathbf{Z} \times \mathbf{Z}$ and $SS \subseteq \mathbf{Sign} \times \mathbf{Sign}$. This corresponds to using an extraction function $\mathsf{twosigns}' : \mathbf{Z} \times \mathbf{Z} \to \mathbf{Sign} \times \mathbf{Sign}$ given by $\mathsf{twosigns}'(z_1, z_2) = (\mathsf{sign}(z_1), \mathsf{sign}(z_2))$.

Once again consider the expression `(x,-x)` in the source language that has a value in $\{(z,-z) \mid z \in \mathbf{Z}\}$. In the present setting $\{(z,-z) \mid z \in \mathbf{Z}\}$ is an element of $\mathcal{P}(\mathbf{Z} \times \mathbf{Z})$ and it is described by the set $\alpha_{\mathrm{SS}'}(\{(z,-z) \mid z \in \mathbf{Z}\}) = \{(\texttt{-},\texttt{+}),(\texttt{0},\texttt{0}),(\texttt{+},\texttt{-})\}$ of $\mathcal{P}(\mathbf{Sign} \times \mathbf{Sign})$. Hence the information about the relative signs of the two components is preserved.

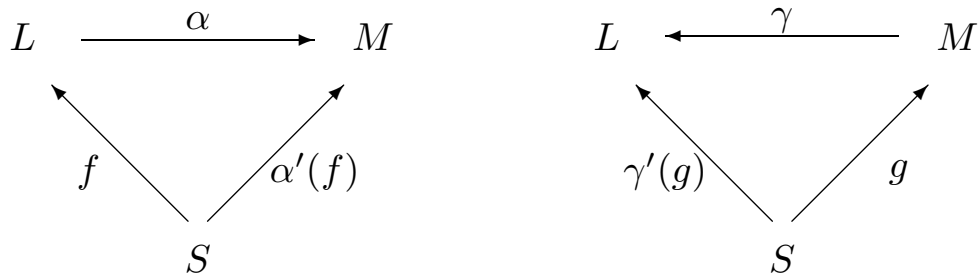
Total function space

Let (L,α,γ,M) be a Galois connection and let S be a set. Then we obtain a Galois connection

$$(S \to L, \alpha', \gamma', S \to M)$$

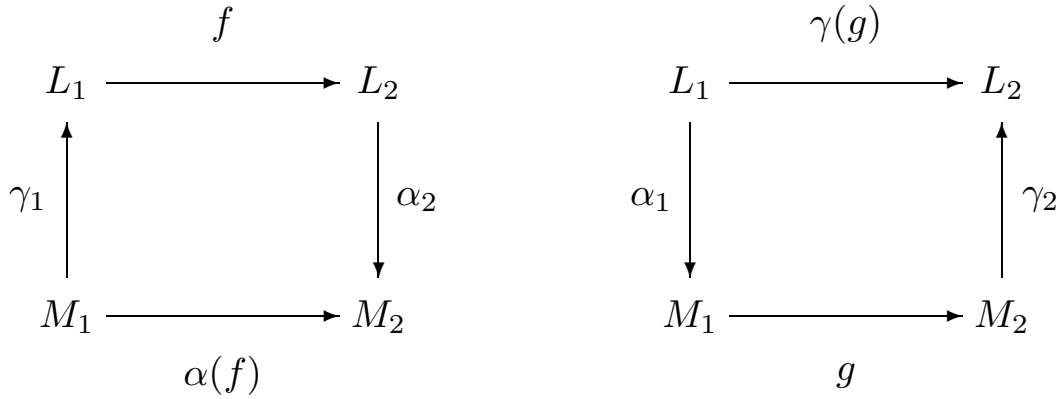
by taking

$$\begin{aligned} \alpha'(f) &= \alpha \circ f \\ \gamma'(g) &= \gamma \circ g \end{aligned}$$



Let $(L_1, \alpha_1, \gamma_1, M_1)$ and $(L_2, \alpha_2, \gamma_2, M_2)$ be Galois connections. Then we obtain the Galois connection $(L_1 \to L_2, \alpha, \gamma, M_1 \to M_2)$ by taking

$$\begin{aligned} \alpha(f) &= \alpha_2 \circ f \circ \gamma_1 \\ \gamma(g) &= \gamma_2 \circ g \circ \alpha_1 \end{aligned}$$



So far our constructions have shown how to combine Galois connections dealing with individual components of the data into Galois connections dealing with composite data. We shall now show how two analyses dealing with the *same* data can be combined into one analysis; this amounts to performing two analyses in parallel. We shall consider two variants of this analysis, one "corresponding" to the independent attribute method and one "corresponding" to the relational method.

Let $(L, \alpha_1, \gamma_1, M_1)$ and $(L, \alpha_2, \gamma_2, M_2)$ be Galois connections. The direct product of the two Galois connections will be the Galois connection

$$(L, \alpha, \gamma, M_1 \times M_2)$$

where α and γ are given by:

$$\begin{aligned} \alpha(l) &= (\alpha_1(l), \alpha_2(l)) \\ \gamma(m_1, m_2) &= \gamma_1(m_1) \sqcap \gamma_2(m_2) \end{aligned}$$

Let us consider how this construction can be used to combine the detection of signs analysis for pairs of integers with the analysis of difference in magnitude.
We get the Galois connection

$$(\mathcal{P}(\mathbf{Z} \times \mathbf{Z}), \alpha_{\mathrm{SSR}}, \gamma_{\mathrm{SSR}}, \mathcal{P}(\mathbf{Sign} \times \mathbf{Sign}) \times \mathcal{P}(\textbf{Range}))$$

where α_{SSR} and γ_{SSR} are given by:

$$\begin{aligned} \alpha_{\mathrm{SSR}}(ZZ) &= (\{(\mathsf{sign}(z_1), \mathsf{sign}(z_2)) \mid (z_1, z_2) \in ZZ\}, \\ &\quad \{\mathsf{range}(|z_1| - |z_2|) \mid (z_1, z_2) \in ZZ\}) \\ \gamma_{\mathrm{SSR}}(SS, R) &= \{(z_1, z_2) \mid (\mathsf{sign}(z_1), \mathsf{sign}(z_2)) \in SS\} \\ &\quad \cap \{(z_1, z_2) \mid \mathsf{range}(|z_1| - |z_2|) \in R\} \end{aligned}$$

Note that the expression `(x, 3*x)` in the source language has a value in $\{(z, 3 * z) \mid z \in \mathbf{Z}\}$ which is described by $\alpha_{\mathrm{SSR}}(\{(z, 3 * z) \mid z \in \mathbf{Z}\}) = (\{(\texttt{-}, \texttt{-}), (\texttt{0}, \texttt{0}), (\texttt{+}, \texttt{+})\}, \{\texttt{0}, \texttt{<-1}\})$. Thus we do not exploit the fact that if the pair is described by $(\texttt{0}, \texttt{0})$ then the difference in magnitude will indeed be described by `0` whereas if the pair is described by $(\texttt{-}, \texttt{-})$ or $(\texttt{+}, \texttt{+})$ then the difference in magnitude will indeed be described by `<-1`.

Direct tensor product

It is possible to do better by letting the two components interact with one another. Again we shall only consider the simple case of powersets so let $(\mathcal{P}(V), \alpha_i, \gamma_i, \mathcal{P}(D_i))$ be Galois connections. Then the direct tensor product is the Galois connection

$$(\mathcal{P}(V), \alpha, \gamma, \mathcal{P}(D_1 \times D_2))$$

where α and γ are defined by:

$$\begin{aligned} \alpha(V') &= \bigcup \{\alpha_1(\{v\}) \times \alpha_2(\{v\}) \mid v \in V'\} \\ \gamma(DD) &= \{v \mid \alpha_1(\{v\}) \times \alpha_2(\{v\}) \subseteq DD\} \end{aligned}$$

where $V' \subseteq V$ and $DD \subseteq D_1 \times D_2$.

We will now get a Galois connection

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

where

$$\begin{array}{rcl} \alpha_{\mathrm{SSR}'}(ZZ) & = & \{(\mathsf{sign}(z_1), \mathsf{sign}(z_2), \mathsf{range}(|z_1| - |z_2|)) \\ & & \mid (z_1, z_2) \in ZZ\} \\ \gamma_{\mathrm{SSR}'}(SSR) & = & \{(z_1, z_2) \\ & & \mid (\mathsf{sign}(z_1), \mathsf{sign}(z_2), \mathsf{range}(|z_1| - |z_2|)) \in SSR\} \end{array}$$

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

Reduced product and tensor product

Let $(L, \alpha_1, \gamma_1, M_1)$ and $(L, \alpha_2, \gamma_2, M_2)$ be Galois connections. Then the reduced product is the Galois insertion

$$(L, \alpha, \gamma, \varsigma[M_1 \times M_2])$$

where

$$\begin{array}{rcl} \alpha(l) & = & (\alpha_1(l), \alpha_2(l)) \\ \gamma(m_1, m_2) & = & \gamma_1(m_1) \sqcap \gamma_2(m_2) \\ \varsigma(m_1, m_2) & = & \bigsqcap \{(m_1', m_2') \mid \gamma_1(m_1) \sqcap \gamma_2(m_2) \\ & & = \gamma_1(m_1') \sqcap \gamma_2(m_2')\} \end{array}$$

Next let $(\mathcal{P}(V), \alpha_i, \gamma_i, \mathcal{P}(D_i))$ be Galois connections for $i = 1, 2$. Then the reduced tensor product is the Galois insertion

$$(\mathcal{P}(V), \alpha, \gamma, \varsigma[\mathcal{P}(D_1 \times D_2)])$$

where

$$\begin{aligned}
\alpha(V') &= \bigcup\{\alpha_1(\{v\}) \times \alpha_2(\{v\}) \mid v \in V'\} \\
\gamma(DD) &= \{v \mid \alpha_1(\{v\}) \times \alpha_2(\{v\}) \subseteq DD\} \\
\varsigma(DD) &= \bigcap\{DD' \mid \gamma(DD) = \gamma(DD')\}
\end{aligned}$$

We noted that the complete lattice $\mathcal{P}(\mathbf{Sign} \times \mathbf{Sign} \times \mathbf{Range})$ contains more than one element that describes the empty set of $\mathcal{P}(\mathbf{Z} \times \mathbf{Z})$. The function $\varsigma_{\mathrm{SSR}'}$ will amount to

$$\begin{aligned}
&\varsigma_{\mathrm{SSR}'}(SSR) = \bigcap\{SSR' \mid \gamma_{\mathrm{SSR}'}(SSR) \\
&= \gamma_{\mathrm{SSR}'}(SSR')\}
\end{aligned}$$

where $SSR, SSR' \subseteq \mathbf{Sign} \times \mathbf{Sign} \times \mathbf{Range}$.

In particular, $\varsigma_{\mathrm{SSR}'}$ will map the singleton sets constructed from the 16 elements

$$
\begin{array}{llll}
(\texttt{-},\texttt{0},\texttt{<-1}), & (\texttt{-},\texttt{0},\texttt{-1}), & (\texttt{-},\texttt{0},\texttt{0}), & \\
(\texttt{0},\texttt{-},\texttt{0}), & (\texttt{0},\texttt{-},\texttt{+1}), & (\texttt{0},\texttt{-},\texttt{>+1}), & \\
(\texttt{0},\texttt{0},\texttt{<-1}), & (\texttt{0},\texttt{0},\texttt{-1}), & (\texttt{0},\texttt{0},\texttt{+1}), & (\texttt{0},\texttt{0},\texttt{>+1}), \\
(\texttt{0},\texttt{+},\texttt{0}), & (\texttt{0},\texttt{+},\texttt{+1}), & (\texttt{0},\texttt{+},\texttt{>+1}), & \\
(\texttt{+},\texttt{0},\texttt{<-1}), & (\texttt{+},\texttt{0},\texttt{-1}), & (\texttt{+},\texttt{0},\texttt{0}) &
\end{array}
$$

to the empty set.

The remaining 29 elements of $\mathbf{Sign} \times \mathbf{Sign} \times$ **Range** are

$$
\begin{array}{lllll}
(\texttt{-},\texttt{-},\texttt{<-1}), & (\texttt{-},\texttt{-},\texttt{-1}), & (\texttt{-},\texttt{-},\texttt{0}), & (\texttt{-},\texttt{-},\texttt{+1}), & (\texttt{-},\texttt{-},\texttt{>+1}), \\
(\texttt{-},\texttt{0},\texttt{+1}), & (\texttt{-},\texttt{0},\texttt{>+1}), & & & \\
(\texttt{-},\texttt{+},\texttt{<-1}), & (\texttt{-},\texttt{+},\texttt{-1}), & (\texttt{-},\texttt{+},\texttt{0}), & (\texttt{-},\texttt{+},\texttt{+1}), & (\texttt{-},\texttt{+},\texttt{>+1}), \\
(\texttt{0},\texttt{-},\texttt{<-1}), & (\texttt{0},\texttt{-},\texttt{-1}), & (\texttt{0},\texttt{0},\texttt{0}), & (\texttt{0},\texttt{+},\texttt{<-1}), & (\texttt{0},\texttt{+},\texttt{-1}), \\
(\texttt{+},\texttt{-},\texttt{<-1}), & (\texttt{+},\texttt{-},\texttt{-1}), & (\texttt{+},\texttt{-},\texttt{0}), & (\texttt{+},\texttt{-},\texttt{+1}), & (\texttt{+},\texttt{-},\texttt{>+1}), \\
(\texttt{+},\texttt{0},\texttt{+1}), & (\texttt{+},\texttt{0},\texttt{>+1}), & & & \\
(\texttt{+},\texttt{+},\texttt{<-1}), & (\texttt{+},\texttt{+},\texttt{-1}), & (\texttt{+},\texttt{+},\texttt{0}), & (\texttt{+},\texttt{+},\texttt{+1}), & (\texttt{+},\texttt{+},\texttt{>+1})
\end{array}
$$

and they describe disjoint subsets of $\mathbf{Z} \times \mathbf{Z}$. Let us call the above set of 29 elements for **AB** (for Array Bound); then $\varsigma_{\mathrm{SSR}'}[\mathcal{P}(\mathbf{Sign} \times \mathbf{Sign} \times \textbf{Range})]$ is isomorphic to $\mathcal{P}(\textbf{AB})$.

To conclude the development of the complete lattice and the associated Galois connection for the Array Bound Analysis we shall simply construct the reduced tensor product of the Galois connections. This will yield a *Galois insertion* isomorphic to

$$(\mathcal{P}(\mathbf{Z} \times \mathbf{Z}), \alpha_{\mathrm{SSR}'}, \gamma_{\mathrm{SSR}'}, \mathcal{P}(\mathbf{AB}))$$

Note that from an implementation point of view the last step of the construction has paid off: if we had stopped with the direct tensor product then the properties would need 45 bits for their representation whereas now 29 bits suffice.

Summary. The Array Bound Analysis has been designed from three simple Galois connections specified by extraction functions:

(i) an analysis approximating integers by their sign,

(ii) an analysis approximating pairs of integers by their difference in magnitude, and

(iii) an analysis approximating integers by their closeness to 0, 1 and -1.

We have illustrated different ways of combining these analyses:

(iv) the relational product of analysis (i) with itself,

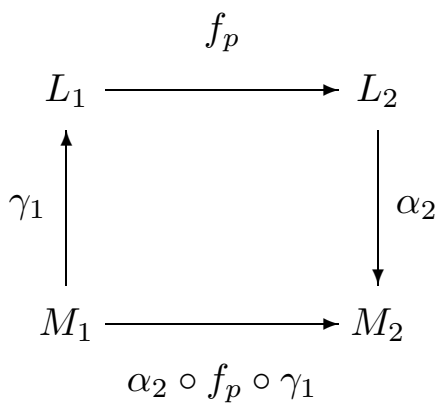
(v) the functional composition of analysis (ii) and (iii), and

(vi) the reduced tensor product of analysis (iv) and (v).

Suppose that we have Galois connections $(L_i, \alpha_i, \gamma_i, M_i)$ such that each M_i is a more approximate version of L_i (for $i = 1, 2$). One way to make use of this is to replace an existing analysis $f_p : L_1 \to L_2$ with a new and more approximate analysis $g_p : M_1 \to M_2$. We already saw that

$$\alpha_2 \circ f_p \circ \gamma_1 \text{ is a candidate for } g_p$$

(just as $\gamma_2 \circ g_p \circ \alpha_1$ would be a candidate for f_p). The analysis $\alpha_2 \circ f_p \circ \gamma_1$ is said to be induced by f_p and the two Galois connections



We studied the simple program `plus` and specified the very precise analysis

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

using the complete lattices $(\mathcal{P}(\mathbf{Z}), \subseteq)$ and $(\mathcal{P}(\mathbf{Z} \times \mathbf{Z}), \subseteq)$.
We introduced the Galois connection

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

for approximating sets of integers by sets of signs.
We used the relational method to get the Galois connection

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

operating on pairs of integers.

We now want to induce a more approximate analysis for the `plus` program

$$g_{\mathrm{plus}} : \mathcal{P}(\mathbf{Sign} \times \mathbf{Sign}) \to \mathcal{P}(\mathbf{Sign})$$

from the existing analysis f_{plus}. To do so we take

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

and simply calculate (for $SS \subseteq \mathbf{Sign} \times \mathbf{Sign}$)

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

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

Let us next consider the situation where the analysis $f_p : L_1 \to L_2$ requires the computation of the least fixed point of a monotone function $F : (L_1 \to L_2) \to (L_1 \to L_2)$ so that $f_p = \mathit{lfp}(F)$. The Galois connections $(L_i, \alpha_i, \gamma_i, M_i)$ give rise to a Galois connection $(L_1 \to L_2, \alpha, \gamma, M_1 \to M_2)$ between the monotone function spaces. We can now apply our technique of inducing and let $G : (M_1 \to M_2) \to (M_1 \to M_2)$ be an upper approximation to $\alpha \circ F \circ \gamma$. It will be natural to take $g_p : M_1 \to M_2$ to be $g_p = \mathit{lfp}(G)$.

Suppose that we have a Galois connection (L, α, γ, M) between the complete lattices L and M, and also a monotone function $f : L \to L$. Often the motivation for approximating f arises because a fixed point of f is desired, and the ascending chain $(f^n(\bot))_n$ does not eventually stabilise (or may do so in too many iterations). Instead of using $\alpha \circ f \circ \gamma : M \to M$ to remedy this situation it is often possible to consider a widening operator $\nabla_M : M \times M \to M$ and use it to define $\nabla_L : L \times L \to L$ by the formula:

$$l_1 \ \nabla_L \ l_2 = \gamma(\alpha(l_1) \ \nabla_M \ \alpha(l_2))$$

Let (L, α, γ, M) be a Galois connection and let $\nabla_M : M \times M \to M$ be an upper bound operator. Then the formula

$$l_1 \ \nabla_L \ l_2 = \gamma(\alpha(l_1) \ \nabla_M \ \alpha(l_2))$$

defines an upper bound operator $\nabla_L : L \times L \to L$. It defines a widening operator if one of the following two conditions are fulfilled:

(i) M satisfies the Ascending Chain Condition, or

(ii) (L, α, γ, M) is a Galois insertion and $\nabla_M : M \times M \to M$ is a widening operator.

If (L, α, γ, M) is a Galois insertion such that $\gamma(\bot_M) = \bot_L$, and if $\nabla_M : M \times M \to M$ is a widening operator, then the widening operator $\nabla_L : L \times L \to L$ defined by $l_1 \ \nabla_L \ l_2 = \gamma(\alpha(l_1) \ \nabla_M \ \alpha(l_2))$ satisfies

$$\mathit{lfp}_{\nabla_L}(f) = \gamma(\mathit{lfp}_{\nabla_M}(\alpha \circ f \circ \gamma))$$

for all monotone functions $f : L \to L$.

Acknowledgements

The main source of these slides was Chapter 4 of Principles of Program Analysis, a book I wrote with Flemming Nielson and Hanne Riis Nielson. The corrected, 2nd printing was published by Springer in 2005.

Thank you