Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 249 (2009) 19–37
www.elsevier.com/locate/entcs

Abstract Interpretation From a Denotational-semantics Perspective

David A. Schmidt

Computing and Information Sciences Dept.
Kansas State University
Manhattan, KS 66506 USA

Abstract

The basic principles of abstract interpretation are explained in terms of Scott-Strachey-style denotational semantics: abstract-domain creation is defined as the selection of a finite approximant in the inverse-limit construction of a Scott-domain. Abstracted computation functions are defined in terms of an embedding-projection pair extracted from the inverse-limit construction. The key notions of abstract-interpretation backwards and forwards completeness are explained in terms of topologically closed and continuous maps in a coarsened version of the Scott-topology. Finally, the inductive-definition format of a language's denotational semantics is used as the framework into which the abstracted domain and abstracted computation functions are inserted, thus defining the language's abstract interpretation.

Keywords: Abstract interpretation, denotational semantics, Galois connection, Scott-domains, Scott-topology

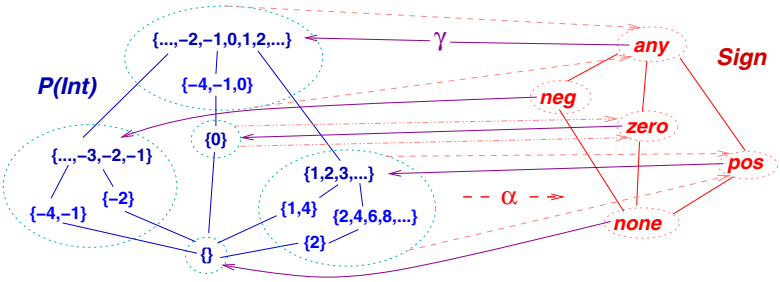
1 Introduction

Denotational semantics [19,29,31,32] and abstract interpretation [3,5,6] came to life about the same time, and their intents were complementary: denotational semantics showed how to define a program's extensional meaning independently from a machine, and abstract interpretation showed how to deduce a program's properties in advance of running the program on a machine. In a previous MFPS presentation [4], Patrick Cousot explained how abstract interpretation can derive a program's denotational semantics as an abstraction of the program's trace semantics, thus explaining denotational semantics from an abstract-interpretation perspective.

In this paper, we take the dual course: We derive a popular form of abstract interpretation from denotational semantics. Given a language's denotational semantics, defined upon a domain, D^{∞}, constructed by an inverse-limit construction,

[1] Supported by NSF ITR-0326577.
[2] Email: schmidt@cis.ksu.edu

1571-0661 © 2009 Elsevier B.V. Open access under CC BY-NC-ND license.
doi:10.1016/j.entcs.2009.07.082



Define $Sign = \{none, neg, zero, pos, any\}$ and $\gamma : Sign \to \mathcal{P}(Int)$ as
$\gamma(none) = \emptyset$; $\gamma(neg) = \{\cdots, -2, -1\}$; $\gamma(zero) = \{0\}$; $\gamma(pos) = \{1, 2, \cdots\}$; $\gamma(any) = Int$

For
$succ(i) = i + 1$
$negate(i) = -i$
$sq(i) = i * i$

define

$succ^\sharp(none) = none$	$negate^\sharp(neg) = pos$	$sq^\sharp(neg) = pos$
$succ^\sharp(neg) = any$	$negate^\sharp(zero) = zero$	$sq^\sharp(zero) = zero$
$succ^\sharp(zero) = pos$	$negate^\sharp(pos) = neg$	$sq^\sharp(pos) = pos$
$succ^\sharp(pos) = pos$	$negate^\sharp(any) = any$	$sq^\sharp(any) = any$
$succ^\sharp(any) = any$	$negate^\sharp(none) = none$	$sq^\sharp(none) = none$

Fig. 1. Abstract domain, *Sign*, and associated functions

we replace D^∞ in the semantics by one of its finite approximants, D_k, $k \geq 0$, from the inverse-limit construction. Elements of D_k are interpreted to denote *subsets* of D^∞. Functions that compute on D^∞ are projected to operate on D_k; this is done with the aid of the embedding-projection pair between D_k and D^∞. Soundness of this "abstract" denotational semantics is ensured by the embedding-projection pair. The inductive-definition format of a language's denotational semantics is used as the framework into which the abstracted domain and abstracted computation functions are inserted, thus defining the language's abstract interpretation.

We judge the quality of the abstract interpretation we have defined in terms of a coarser variant of Scott-topology, and we characterize the so-called forwards- and backwards-complete ("homomorphic") functions of abstract-interpretation theory [11,10] as the topologically closed and topologically continuous maps on the weakened Scott-topology.

In this fashion, abstract interpretation is derived from denotational semantics.

2 Background: Abstract interpretation

Abstract interpretration is approximation by computation on properties. For concrete-data domain, Σ, we select a set of property names, A, such that each $a \in A$ names a set $\gamma(a) \subseteq \Sigma$, for $\gamma : A \to \mathcal{P}(\Sigma)$. γ identifies the family of properties (data-test sets) modelled by A. Order A s.t. $a \sqsubseteq a'$ iff $\gamma(a) \subseteq \gamma(a')$ — the result should be a partial ordering.

Figure 1 displays an approximation of the concrete integers, *Int*, by sign properties named by complete lattice, *Sign*.

When γ possesses an adjoint, $\alpha : \mathcal{P}(\Sigma) \to Sign$, then there is a *Galois connection* (that is, $S \subseteq \gamma(a)$ iff $\alpha(S) \sqsubseteq a$, for all $S \in \mathcal{P}(\Sigma)$ and $a \in A$). α is the *lower adjoint* and γ is the *upper adjoint*, and we write this as $\mathcal{P}(\Sigma)\langle\alpha, \gamma\rangle Sign$. This makes $\rho = \gamma \circ \alpha$

Q: For this program,

```
readInt(x)
x = succ(x)
if x < 0 :
  x = negate(x)
else:
  x = succ(x)
writeInt(x)
```

is the output pos?

A: abstractly interpret input domain *Int* by *Sign* to see:

```
readSign(x)
x = succ♯(x)
if (filterNeg(x):
     x = negate♯(x))
  (filterNonNeg(x):
     x = succ♯(x))  fi
writeSign(x)
```

where $filterNeg : Sign \to Sign$ and $filterNonNeg : Sign \to Sign$ are defined

$filterNeg(none) = none$	$filterNonNeg(none) = none$
$filterNeg(neg) = neg$	$filterNonNeg(neg) = none$
$filterNeg(zero) = none$	$filterNonNeg(zero) = zero$
$filterNeg(pos) = none$	$filterNonNeg(pos) = pos$
$filterNeg(any) = neg$	$filterNonNeg(any) = any$

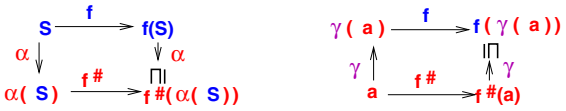
For the abstract data-test sets, $zero, pos, neg$, we calculate the outcomes; they are

$$\{zero \mapsto pos,\ pos \mapsto pos,\ neg \mapsto any\}$$

They validate that all nonnegative inputs yield positive outputs. The failure to validate the result for input neg arises because $succ^\sharp(neg) = any$ and $filterNeg(any) = neg$ (good) but $filterNonNeg(any) = any$ (bad — we need $zero \vee pos$ to deduce the needed result), so we cannot predict the success of the else-arm.

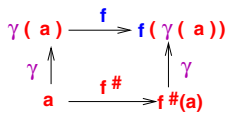
Fig. 2. An abstract interpretation using *Sign*

$f^\sharp : A \to A$ is *sound* for $f : \Sigma \to \Sigma$ iff $\alpha \circ f \sqsubseteq f^\sharp \circ \alpha$ iff $f \circ \gamma \sqsubseteq \gamma \circ f^\sharp$



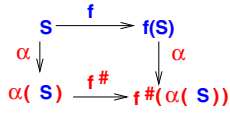
α and γ act as *semi-homomorphisms*.

Forwards completeness [10]: $f \circ \gamma = \gamma \circ f^\sharp$



γ is a homomorphism from A to $\mathcal{P}(\Sigma)$ — it preserves $f^\sharp$ as f.

Backwards completeness [6,11]: $\alpha \circ f = f^\sharp \circ \alpha$



α is a homomorphism from $\mathcal{P}(\Sigma)$ to A — it preserves f as $f^\sharp$.

Fig. 3. Sound and complete forms of abstract functions

an *upper closure operator* — $\rho : \mathcal{P}(\Sigma) \to \mathcal{P}(\Sigma)$ is monotone, extensive ($S \subseteq \rho(S)$), and idempotent ($\rho \circ \rho = \rho$).

$\rho[\mathcal{P}(Int)]$ identifies the *properties* expressible by abstract domain *Sign*, and ρ maps a test set to its *minimal property*, e.g., $\rho\{1\} = \{1, 2, \cdots\}$, $\rho\{-1, 1\} = Int$, etc. Note that $\rho[\mathcal{P}(Int)]$ is closed under intersection (conjunction).

From here on, we work with Galois connections of form, $(\mathcal{P}(\Sigma), \subseteq)\langle\alpha, \gamma\rangle(A, \sqsubseteq)$, so that $\rho = \gamma \circ \alpha$ maps sets to sets, and we assume that α is onto.

Computation functions, $f : \Sigma \to \Sigma$, are *soundly approximated by* $f^\sharp : A \to A$ iff $\alpha(f[S]) \sqsubseteq f^\sharp(\alpha(S))$, for all $S \in \mathcal{P}(\Sigma)$ (equivalently, iff $f[\gamma(a)] \subseteq \gamma(f^\sharp(a))$, for all $a \in A$), where we define $f[S] = \{f(s) \mid s \in S\}$, as usual. Figure 2 applies the functions from Figure 1 to interpret a program that computes upon *Int* so that it soundly computes upon *Sign*, which represents the data-test sets of interest.

Recall that $\rho[\mathcal{P}(\Sigma)] = \gamma[A]$ identifies the properties expressed by A. When α is onto, we can treat $f^\sharp : A \to A$ as $f^\sharp : \rho[\mathcal{P}(\Sigma)] \to \rho[\mathcal{P}(\Sigma)]$, for example, $succ^\sharp\{0\} = \{1, 2, \cdots\}$.

Proposition 2.1 *For all $\phi \in \rho[\mathcal{P}(\Sigma)]$, $f^\sharp : \rho[\mathcal{P}(\Sigma)] \to \rho[\mathcal{P}(\Sigma)]$ is sound for $f : \Sigma \to \Sigma$ iff $f(\phi) \subseteq f^\sharp(\phi)$.*

There is also the dual notion, *underapproximating soundness*, where $f(\phi) \supseteq f^\sharp(\phi)$; this is best developed with an interior map, $\iota : \mathcal{P}(\Sigma) \to \mathcal{P}(\Sigma)$. We leave this for later in the paper.

The most precise (*strongest*) $f^\sharp$ for function f is defined $f_0^\sharp = \alpha \circ f \circ \gamma$. It is strongest in the sense that $f_0^\sharp \sqsubseteq f^\sharp$ for all $f^\sharp$ that are sound for f.

We can define $f_0^\sharp : \rho[\mathcal{P}(\Sigma)] \to \rho[\mathcal{P}(\Sigma)]$ in terms of $\rho = \gamma \circ \alpha$ as $f_0^\sharp = \rho \circ f$. For example, for the *Sign* domain and its closure map, ρ, $succ_0^\sharp\{0\} = \{1, 2, \cdots\}$ and $succ_0^\sharp\{\cdots, -2, -1\} = \{\cdots, -2, -1, 0, 1, 2, \cdots\}$.

Proposition 2.2 *(strongest postcondition for f): For all $\phi, \psi \in \rho[\mathcal{P}(\Sigma)]$, if $f(\phi) \subseteq \psi$, then $f_0^\sharp(\phi) \subseteq \psi$.*

When f is approximated exactly by $f^\sharp$ such that $f \circ \gamma = \gamma \circ f^\sharp$, we say f is *forwards complete* [10]. When f is approximated exactly such that $\alpha \circ f = f^\sharp \circ \alpha$, we say f is *backwards complete* [11,27]. See Figure 3. In Figure 1, *sq* is backwards but not forwards complete; *negate* is both backwards and forwards complete, and *succ* is neither.

Define $f_0^\sharp = \rho \circ f : \rho[\mathcal{P}(\Sigma)] \to \rho[\mathcal{P}(\Sigma)]$ as before.

Proposition 2.3 *[10] The following are equivalent:*

- *$f_0^\sharp$ is forwards complete for f*
- *for all $\phi \in \rho[\mathcal{P}(\Sigma)]$, $f(\phi) \in \rho[\mathcal{P}(\Sigma)]$*
- *$f \circ \rho = \rho \circ f \circ \rho$*

Proposition 2.4 *[6,11] The following are equivalent:*

- *$f_0^\sharp$ is backwards complete for f*
- *for all $S_1, S_2 \in \mathcal{P}(\Sigma)$, $\rho(S_1) = \rho(S_2)$ implies $\rho(f[S_1]) = \rho(f[S_2])$*
- *$\rho \circ f = \rho \circ f \circ \rho$.*

An abstract function, $f^\sharp$, that is forwards or backwards complete for f is also strongest for f, so it is unclear at this point exactly what is gained from these notions. We will resolve this question later in the paper.

3 Background: Denotational semantics

One might explain *denotational semantics* as the interpretation of a program's phrases as values from *Scott-domains*. We treat a Scott-domain as "an SFP object," that is, as the inverse limit of a sequence of finite-cardinality bcpos, related by embedding-projection pairs [14,25]. Figure 4 presents the Scott-domain

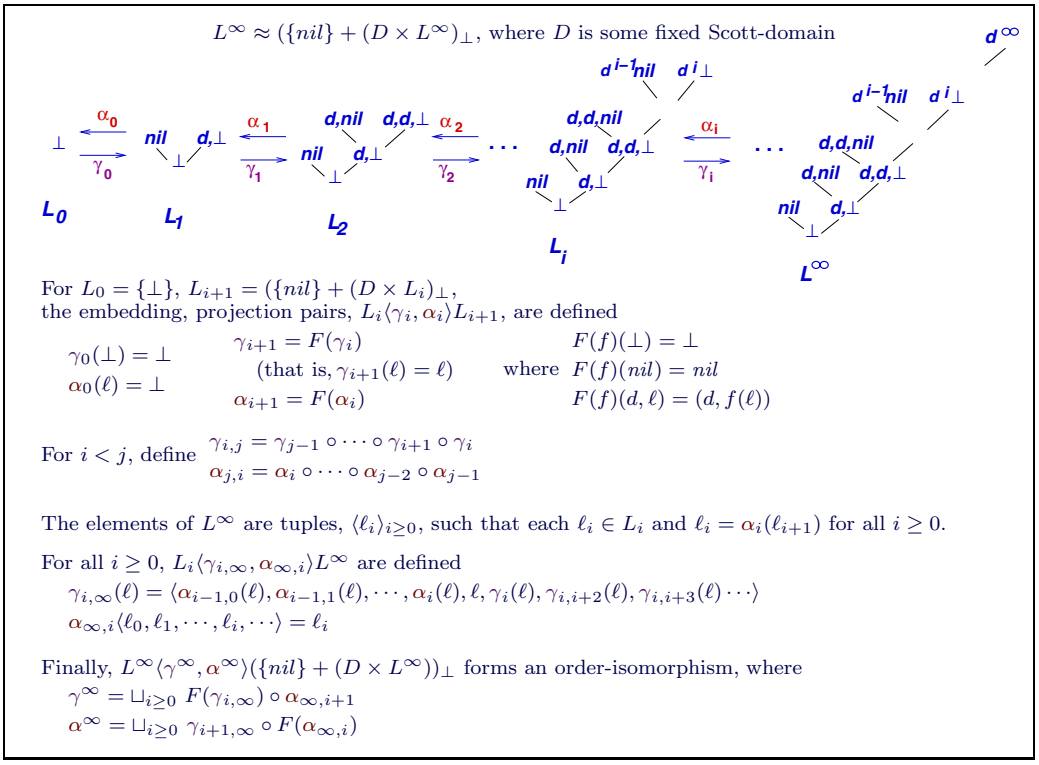


Fig. 4. Inverse limit of $L = (\{nil\} + (D \times L))_\perp$

of partial, total, finite, and infinite lists corresponding to the domain equation, $L = (\{nil\} + (D \times L))_\perp$.[3] For each $i \geq 0$, the corresponding embedding-projection pair defines a Galois connection, $L_i\langle\gamma_i, \alpha_i\rangle L_{i+1}$, as does $L_i\langle\gamma_{i,\infty}, \alpha_{\infty,i}\rangle L^\infty$. (Here, the γ maps are *lower* adjoints.)

Figure 5 shows a denotational semantics for a while-language based on L^∞. A store is a mapping from a set of variables, Var, to values in L^∞. Absence of store is denoted by $\underline{\perp}$. The language uses a guarded-if construction, where a guard, $\mathtt{G}_j$, filters the input store to its guarded command, $\mathtt{C}_j$, and the results of all $\mathtt{G_j : C_j}$ pairs are joined. When the guards of an if-command are mutually exclusive, the semantics is the usual one. We use this formulation to ease the transition into abstract interpretation, which treats analysis of software much like analysis of hardware circuits (cf. Figure 2).

The while-command is a tail-recursive guarded-if, such that `while B do C` has a denotation equal to `if` $(\neg$`B : skip`$)$, $($`B : C; (while B do C)`$)$ `fi`.

Here is an example: let $\sigma_0 = [[\![\mathtt{x}]\!] \mapsto nil]$. Then,

[3] As usual, $+$ represents disjoint union, $\times$ is product, and $_\perp$ is lifting.

$\mathtt{d} \in \mathit{Data}$(atomic data) $\quad \mathtt{x} \in \mathit{Var}$(variable names) $\quad \mathtt{G} \in \mathit{Guard}$(boolean expressions)

$\mathtt{E} \in \mathit{Expression} ::= \mathtt{x} \mid \mathtt{tl\ E} \mid \mathtt{cons\ d\ E}$

$\mathtt{C} \in \mathit{Command} ::= \mathtt{x} = \mathtt{E} \mid \mathtt{C_1; C_2} \mid \mathtt{if}\ (\mathtt{G_i} : \mathtt{C_i})_{i \in I}\ \mathtt{fi} \mid \mathtt{while\ G\ do\ C}$

Domain of stores: $\sigma \in \Sigma = \mathit{Var} \to L^\infty$

$\mathcal{G} : \mathit{Guard} \to \Sigma \to \Sigma_\perp$

$\mathcal{G}[\![\mathtt{G}]\!]\sigma = \sigma$ when $\mathtt{G}$ holds true in σ; $\quad \mathcal{G}[\![\mathtt{G}]\!]\sigma = \underline{\perp}$ otherwise

$\mathcal{E} : \mathit{Expression} \to \Sigma \to L^\infty$

$\mathcal{E}[\![\mathtt{x}]\!]\sigma = \mathit{lookup}\ [\![\mathtt{x}]\!]\ \sigma \quad$ where $\mathit{lookup}\ v\ \sigma = \sigma(v)$

$\mathcal{E}[\![\mathtt{tl\ E}]\!]\sigma = \mathit{tail}\ (\mathcal{E}[\![\mathtt{E}]\!]\sigma) \quad$ where $\mathit{tail}(v) = \mathit{cases}\ \gamma^\infty(v)\ \mathit{of} \begin{cases} \perp : \alpha^\infty(\perp) \\ \mathit{nil} : \alpha^\infty(\perp) \\ (d, \ell) : \ell \end{cases}$

$\mathcal{E}[\![\mathtt{cons\ d\ E}]\!]\sigma = \mathit{cons}\ \mathtt{d}\ (\mathcal{E}[\![\mathtt{E}]\!]\sigma) \quad$ where $\mathit{cons}\ d\ \ell = \alpha^\infty(d, \ell)$

$\mathcal{C} : \mathit{Command} \to \Sigma \to \Sigma_\perp$

$\mathcal{C}[\![\mathtt{x = E}]\!]\sigma = \mathit{update}\ [\![\mathtt{x}]\!]\ (\mathcal{E}[\![\mathtt{E}]\!]\sigma)\ \sigma \quad$ where $\mathit{update}\ v\ \ell\ \sigma = \sigma + [v \mapsto \ell]$

$\mathcal{C}[\![\mathtt{C_1; C_2}]\!] = \mathcal{C}[\![\mathtt{C_2}]\!] \circ \mathcal{C}[\![\mathtt{C_1}]\!] \quad$ Note : $\circ$ forces strictness: $g \circ f(\sigma) = \underline{\perp}$ when $f(\sigma) = \underline{\perp}$

$\mathcal{C}[\![\mathtt{if}\ (\mathtt{G}_i : \mathtt{C}_i)_{i \in I}\ \mathtt{fi}]\!] = \bigsqcup_{i \in I} \mathcal{C}[\![\mathtt{C}_i]\!] \circ \mathcal{G}[\![\mathtt{G}_i]\!]$

$\mathcal{C}[\![\mathtt{while\ G\ do\ C}]\!] = \mathit{lfp}\ \lambda f.\ (\mathcal{G}[\![\neg \mathtt{G}]\!]) \ \sqcup \ (f \circ \mathcal{C}[\![\mathtt{C}]\!] \circ \mathcal{G}[\![\mathtt{G}]\!])$

Fig. 5. Denotational semantics for while-language based on L^∞

$$
\begin{aligned}
&\mathcal{C}[\![\mathtt{if\ (isNil\ x :\ x = cons\ d0\ x)\ (isNonNil\ x :\ x = x)\ fi}]\!]\sigma_0 \\
&= (\mathcal{C}[\![\mathtt{x = cons\ d0\ x}]\!] \circ \mathcal{G}[\![\mathtt{isNil\ x}]\!])\sigma_0 \ \sqcup \ (\mathcal{C}[\![\mathtt{x = x}]\!] \circ \mathcal{G}[\![\mathtt{isNonNil\ x}]\!])\sigma_0 \\
&= \mathcal{C}[\![\mathtt{x = cons\ d0\ x}]\!]\sigma_0 \ \sqcup \ \mathcal{C}[\![\mathtt{x = x}]\!]\underline{\perp} \\
&= (\mathit{update}\ [\![\mathtt{x}]\!]\ (\mathcal{E}[\![\mathtt{cons\ d0\ x}]\!]\sigma_0)\ \sigma_0) \ \sqcup \ \underline{\perp} \ = \ [[\![\mathtt{x}]\!] \mapsto (\mathtt{d0}, \mathit{nil})]
\end{aligned}
$$

The example shows how $\mathcal{G}[\![\mathtt{isNil\ x}]\!]$ passes σ_0 forwards because the guard holds true for the store, whereas $\mathcal{G}[\![\mathtt{isNonNil\ x}]\!]$ passes $\underline{\perp}$.

4 Collecting domains

Reconsidering the L_k domains in Figure 4, we note that an element like $(d, \perp)$ denotes a list that has d as its head element and an unknown tail, that is, $(d, \perp)$ approximates the set, $\{(d, \ell) \mid \ell \in L^\infty\} \subseteq L^\infty$. In this sense, each L_k is an approximation domain, like the ones used for abstract interpretation (cf. *Sign* in Figure 1).

We can formalize this intuition. The *collecting domain*, $\mathcal{P}(L^\infty)$, defines all data-test sets that might be used with a program written in the language defined in Figure 5. If we "crown" L^∞ with a $\top$ element, we have a Galois connection between the collecting domain and complete lattice, ${L^\infty}^\top$; see Figure 6. Element $\top \in {L^\infty}^\top$ denotes contradictory (literally, no) information content and maps to the empty data set in $\mathcal{P}(L^\infty)^{op}$. In contrast, $\perp \in {L^\infty}^\top$ denotes all possible test data. One might also restrict the collecting domain to be just the totally defined lists or just the finite, total lists.

The Figure shows how the Galois connection composes with an embedding-projection pair, $L_k^\top \langle \gamma_{k,\infty}, \alpha_{\infty,k} \rangle {L^\infty}^\top$, where L_k is also crowned. The Galois con-

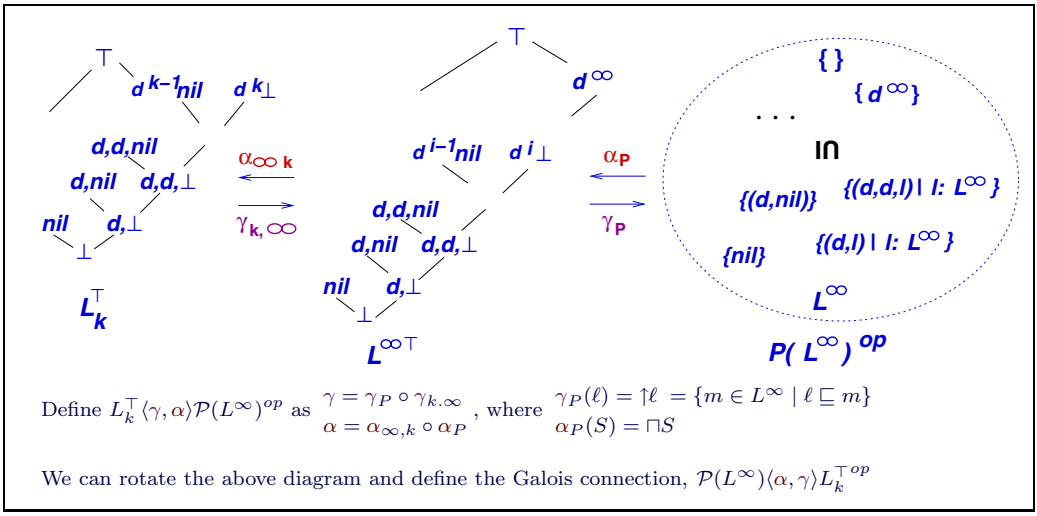
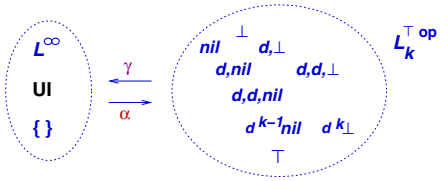


Fig. 6. Collecting domain (data-test sets), $\mathcal{P}(L^\infty)^{op}$, for L^∞ and associated Galois connections

nection that results, $L_k^\top\langle\gamma,\alpha\rangle\mathcal{P}(L^\infty)^{op}$, is significant: If we "rotate" it, we have a Galois connection suitable for abstract interpretation,

$\mathcal{P}(L^\infty)\langle\alpha,\gamma\rangle L_k^{\top op}$:



In this way, we have extracted a useful, crucial abstract interpretation from the Scott-domain's inverse-limit construction.

An element, $(d^n, \bot) \in L_k^{\top op}$, represents those lists having at least n-many elements, for $0 \leq n \leq k$, and (d^n, nil) represents a list that has exactly n elements. As noted, $\bot \in L_k^{\top op}$ stands for all lists; $\top \in L_k^{\top op}$ for none. We can repeat the style of abstract testing in Figure 2 for a program that computes on lists by using elements of $L_k^{\top op}$ as inputs. The next section develops this idea.

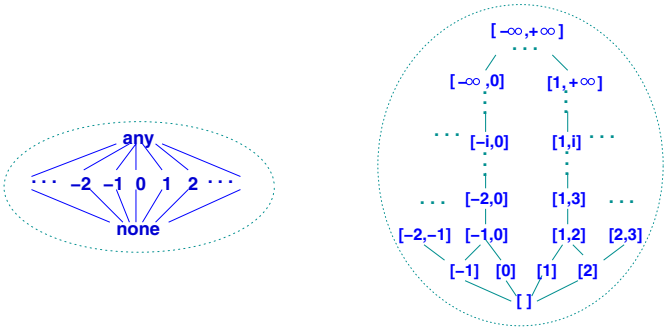
Other abstract domains can be synthesized by means of inverse limits and collecting domains. The *Sign* domain in Figure 1 is derived from these Scott-domain definitions:

$$N = \{1\}_\bot \oplus N \quad \text{where } \oplus \text{ denotes disjoint sum with merged } \bot\text{s}$$
$$S = (N + \{0\} + N)_\bot$$

S denotes the integers partitioned into the negatives, zero, and the positives. The approximating domain, $S_1 = (N_0 + \{0\} + N_0)_\bot$, where $N_0 = \{\bot\}$, defines $Sign = S_1^{\top op}$ in Figure 1. The Galois connection in Figure 1 goes between the collecting domain of sets of *total values* of S^∞ and *Sign*. We can define better-precision signs-analyses by using domains S_k, $k > 1$, which would distinguish individual integers, e.g, $S_2^{\top op} = \{\top, neg, -1, zero, 1, pos, \bot\}$.

Many abstract domains are defined this way — they are "partitions" of data-

test sets, "crowned" by a $\top$, characterized by a finite domain from an inverse-limit sequence. But here are two that are not:



The *Const* domain, shown on the left, is used for constant-propagation analysis: a program's variables are analyzed to see if they are uninitialized ($\bot$), are assigned a single, constant value ($n \in Int$), or are assigned multiple values ($\top$) [24]. Rather than an approximating domain, *Const* is $N^{\infty\top op}$, where N^{∞} is the inverse limit of $N = (\{0\} + N)_{\bot}$.

On the right is the *Interval* domain, which is employed when an analysis must determine the range of values that a variable is assigned [6]. This domain is not finite and its opposite domain cannot be constructed as an SFP object. Further, the map, $\gamma : Interval \to \mathcal{P}(Int)$, is not defined as $\gamma([a,b]) = \uparrow[a,b]$ but as $\gamma([a,b]) = \{n \in Int \mid a \leq n \leq b\}$. Because of its infinite height, this domain must be handled specially when used in an abstract interpretation; we discuss this later.

Domains like *Sign*, *Const*, and *Interval* are used to represent values from Σ; a *relational domain* is a nonfunctional domain that represents values from domain $Var \to \Sigma$. The standard example of a relational domain is the *polyhedral domain* [8], whose values describe linear relationships between variables' values in the store. For example, this set of inequalities between variables, `x`, `y`, and `z`, is an abstract value in the polyhedral domain that abstracts $Var \to \Sigma$:

$$2\mathtt{x} + 1\mathtt{y} \leq 100$$

$$4\mathtt{x} + 1\mathtt{y} + -3\mathtt{z} \leq 0$$

$$-1\mathtt{z} \leq 2$$

Such an abstract value is a conjunctive proposition of form, $\bigwedge_i((\sum_j (a_{ij} \cdot \mathtt{x}_{ij}) \leq b_i)$, and can be implemented as a set of tuples, a matrix, or a graph. It represents all stores whose variables satisfy the conjunctive proposition.

Similar to the polyhedral domain is the octagon domain [20] and the predicate-abstraction domains [13,2]. None of these readily fit the format of a finite domain, $L_k^{\top op}$, in an inverse-limit sequence (but see the remark at the end of Section 6.)

There are also the usual constructions for collecting domains for products, sums, and liftings. Figure 7 shows two such constructions, indexed product and lifting. Both constructions are common to abstract interpretation. The indexed product generates an *independent attribute analysis* [17], where a set of indexed tuples is

Let D be a Scott-domain, A its approximant, and $\mathcal{P}(D)\langle\alpha,\gamma\rangle A$ the collecting Galois connection.

Set-indexed product: $I \to D$, for set I: $\mathcal{P}(I \to D)\langle\alpha_I,\gamma_I\rangle I \to A$

where $\gamma_I(a_i)_{i\in I} = \{(d_i)_{i\in I} \mid d_i \in \gamma(a_i)\}$
$\alpha_I(S) = (\alpha\{t_i \mid t \in S\})_{i\in I}$

Compressed lift: $D_{\underline{\perp}}$: $\mathcal{P}(D \cup \{\underline{\perp}\})\langle\alpha_\perp,\gamma_{\underline{\perp}}\rangle A$ (that is, $\underline{\perp}$ in A is aliased to the existing $\perp \in A$)

where $\gamma_{\underline{\perp}}(a) = \gamma(a) \cup \{\underline{\perp}\}$
$\alpha_{\underline{\perp}}(S) = \alpha(S - \{\underline{\perp}\})$

Fig. 7. Compound Galois connections for collecting domains

abstracted to a single tuple that covers the set. The lifting construction *compresses* the $\underline{\perp}$ element with the existing $\perp$ in A and is used when an abstract interpretation ignores nontermination (which is almost always the case).

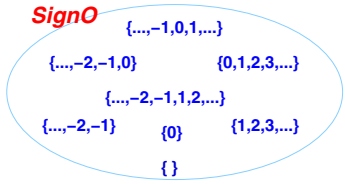
5 Some topology

The intuition that an element from an abstract domain models a set of concrete data-test elements, suggests a topological connection. Indeed, for an approximating domain, L_k, each $\gamma(\ell)$, $\ell \in L_k$, is a Scott-basic open set [12,26] — a "computable property" [30]. As before, we define the closure operator, $\rho = \gamma \circ \alpha : \mathcal{P}(L^\infty) \to \mathcal{P}(L^\infty)$, and we have that the family of sets, $\rho[\mathcal{P}(L^\infty)]$, are all Scott-basic opens and the family is closed under intersection.

It is natural to close $\rho[\mathcal{P}(L^\infty)]$ under unions to generate a topology on L^∞, one that is coarser than the Scott topology — it defines the "topology of the abstract interpretation."

This construction does indeed exist in abstract-interpretation methodology — it is called the *disjunctive completion* [7] of the abstract domain, and it is used to add additional elements to an abstract domain when more precision is needed for an analysis. For example, the *Sign* domain in Figure 1 can be completed into this domain:

$$SignO = \{none,\ neg,\ \leq 0,\ zero,\ \neq 0,\ \geq 0,\ pos,\ any\}$$:



When the domain, *SignO*, used to analyze the program in Figure 2, the analysis can validate, for the test-data sets named by *neg*, *zero*, and *pos*, that the program's output must be positive (*pos*).

6 Some logic

There is another reason why the disjunctive completion is useful. It reminds us that every abstract domain, $L_k^{\top^{op}}$, defines a "logic," where L_k's $\top$ denotes False, L_k's $\perp$ denotes True, and $L_k^{\top^{op}}$'s $\sqsubseteq$ denotes entailment. This particular logic possesses

conjunction, and the disjunctive completion adds disjunction, making the domain a frame [16].

In general, for abstract domain A, its logic is the language of assertions that can be validated using an abstract intepretation based upon A. For example, one can use abstract domain *Sign* to validate that a program's output satisfies assertion, *pos*, or assertion, $any \sqcap pos$, but the domain cannot be used to validate *isEvenValued* or $zero \sqcup pos$ (but this last assertion *can* be validated in *Sign*'s disjunctive completion, *SignO*).

To start, A's logic includes the primitive assertions, a, for every $a \in A$.

Next, define A's $\rho = \gamma \circ \alpha$. Function $f : \Sigma \to \Sigma$ is a *logical operator in A's logic* iff for all $S \in \rho[\mathcal{P}(\Sigma)]$, $f[S] \in \rho[\mathcal{P}(\Sigma)]$, that is, iff $f \circ \rho = \rho \circ f \circ \rho$, that is, iff $f_0^\sharp = \rho \circ f$ is forwards complete for f. The intuition is that f maps property sets to property sets "on the nose" and for this reason, one can use its $f_0^\sharp$ to compute exactly on the assertions in the logic. The concept of logical operator generalizes to n-ary f as well.

For example, *Sign*'s logic includes

$$\phi \ ::= \ a \mid \phi_1 \sqcap \phi_2 \mid negate_0^\sharp \, \phi, \quad \text{where } a \in Sign$$

The logic contains primitive assertions like *neg*, *zero*, etc., as well as conjunctions. Since conjunctions compute on the nose, we have that meets in *Sign* compute to intersections in $\mathcal{P}(Int)$:

$$a \sqsubseteq \phi_1 \sqcap \phi_2 \ \text{ iff } \ \gamma(a) \subseteq \gamma(\phi_1) \cap \gamma(\phi_2)$$

Since *negate* maps properties on the nose, we have that

$$a \sqsubseteq negate_0^\sharp \phi \ \text{ iff } \ \gamma(a) \subseteq negate[\gamma(\phi)]$$

and so on.

In constrast, union ($\cup$) is not a logical operator for *Sign* (although it is for *Sign*0).

The logic of the approximating domain is critical to an abstract interpretation, which must compute sound logical properties of a program in terms of the elements and operations in the abstract domain. *Only properties that belong to the abstract domain's logic may be soundly verified by the abstract interpretation.* This makes notions like Galois connection, disjunctive completion, and forwards completeness critical to the design of a useful abstract interpretation.

Of course, the above development can be read as "domain logic" as presented by Abramsky [1], where a domain like L^∞ is generated from a set of atomic (finite) elements, which are the primitive propositions in the logic, closed under frame-like axioms. And Jensen observed that one can use a finite subset of the atomic elements to define an abstract domain that approximates L^∞, much in the style that we used L_k. Jensen's methodology is called *abstract interpretation in logical form* [15].

It appears possible to use Jensen's framework to describe the relational domains outlined in Section 4, but we do not try to do so here.

Abstract store domain: $\sigma \in \Sigma^\sharp = Var \to L_k^{\top\, op}$

Collecting Galois connections for Scott-domains,

L^∞:	$\mathcal{P}(L^\infty)\langle\alpha,\gamma\rangle L_k^{\top\, op}$
$\Sigma = Var \to L^\infty$:	$\mathcal{P}(\Sigma)\langle\alpha_{Var},\gamma_{Var}\rangle\Sigma^\sharp$, as defined from Figures 5, 6, and 7.
$\Sigma_\perp$:	$\mathcal{P}(\Sigma_\perp)\langle\alpha_\perp,\gamma_\perp\rangle\Sigma^\sharp$

$\mathcal{G}^\sharp : Guard \to \Sigma^\sharp \to \Sigma^\sharp$
$\quad \mathcal{G}^\sharp[\![\mathtt{G}]\!] = \alpha_\perp \circ \mathcal{G}[\![\mathtt{G}]\!] \circ \gamma_{Var}$

$\mathcal{E}^\sharp : Expression \to \Sigma^\sharp \to L_k^{\top\, op}$
$\quad \mathcal{E}^\sharp[\![\mathtt{x}]\!]\sigma = lookup^\sharp\ [\![\mathtt{x}]\!]\ \sigma$
$\qquad$ where $lookup^\sharp\ v = \alpha \circ lookup\ v \circ \gamma_{Var}$, that is, $lookup^\sharp\ v\ \sigma = \sigma(v)$
$\quad \mathcal{E}^\sharp[\![\mathtt{tl\ E}]\!]\sigma = tail^\sharp(\mathcal{E}^\sharp[\![\mathtt{E}]\!]\sigma)$
$\qquad$ where $tail^\sharp = \alpha \circ tail \circ \gamma$, that is, $tail^\sharp(a,\ell) = \ell$; $tail^\sharp(nil) = \perp = tail^\sharp(\perp)$
$\quad \mathcal{E}^\sharp[\![\mathtt{cons\ a\ E}]\!]\sigma = cons^\sharp\ \mathtt{a}\ (\mathcal{E}^\sharp[\![\mathtt{E}]\!]\sigma)$
$\qquad$ where $cons^\sharp(a,v) = \alpha \circ cons\ a \circ \gamma$, that is, $cons^\sharp\ a\ \ell = (a,\ell)$

$\mathcal{C}^\sharp : Command \to \Sigma^\sharp \to \Sigma^\sharp$
$\quad \mathcal{C}^\sharp[\![\mathtt{x} = \mathtt{E}]\!]\sigma = update^\sharp\ [\![\mathtt{x}]\!]\ (\mathcal{E}^\sharp[\![\mathtt{E}]\!]\sigma)\ \sigma$
$\qquad$ where $update^\sharp[\![\mathtt{x}]\!] = \alpha_\perp \circ update[\![\mathtt{x}]\!] \circ (\gamma \times \gamma_{Var})$, that is, $update^\sharp\ v\ \ell\ \sigma = \sigma + [v \mapsto \ell]$
$\quad \mathcal{C}^\sharp[\![\mathtt{C_1; C_2}]\!] = \mathcal{C}^\sharp[\![\mathtt{C_2}]\!] \circ \mathcal{C}^\sharp[\![\mathtt{C_1}]\!]$
$\quad \mathcal{C}^\sharp[\![\mathtt{if}\ (\mathtt{G}_i : \mathtt{C}_i)_I\ \mathtt{fi}]\!] = \bigsqcup_{i\in I} \mathcal{C}^\sharp[\![\mathtt{C}_i]\!] \circ \mathcal{G}^\sharp[\![\mathtt{G}_i]\!]$
$\quad \mathcal{C}^\sharp[\![\mathtt{while\ B\ do\ C}]\!] = lfp\ \lambda f.\ \mathcal{G}^\sharp[\![\neg\mathtt{G}]\!] \sqcup (f \circ \mathcal{C}^\sharp[\![\mathtt{C}]\!] \circ \mathcal{G}^\sharp[\![\mathtt{G}]\!])$

Fig. 8. Abstract interpretation derived from $\mathcal{P}(L^\infty)\langle\alpha,\gamma\rangle L_k^{\top\, op}$

7 Sound and complete abstract semantics

Recall from Section 2 that a Galois connection of form, $\mathcal{P}(C)\langle\alpha,\gamma\rangle A$, defines the modelling of test-data sets from C as elements of A. Computation on $a \in A$ by $f : C \to C$ is modelled by a $f^\sharp : A \to A$ such that $f(\gamma(a)) \subseteq \gamma(f^\sharp(a))$. The most precise such $f^\sharp$ is $\alpha \circ f \circ \gamma$ (where, for $S \subseteq C$, $f[S] = \{f(c) \mid c \in S\}$).

The Galois connection induces an abstract interpretation of a language's denotational semantics: replace domain C by A and replace functions, $f : C \to C$ by $f^\sharp : A \to A$. An induction proof shows that the resulting valuation, $\mathcal{C}^\sharp[\![\mathtt{C}]\!]$, is sound for $\mathcal{C}[\![\mathtt{C}]\!]$, for all phrases, $\mathtt{C}$, in the language. Figure 8 shows the abstract denotational semantics that results from the Galois connection, $\mathcal{P}(L^\infty)\langle\alpha,\gamma\rangle L_k^{\top\, op}$, and the two constructions from Figure 7.

The Figure shows that an abstract interpretation is itself just a denotational semantics, where functions, f, are replaced by their sound approximations, $f^\sharp = \alpha \circ f \circ \gamma$. This style of abstract interpretation was first proposed by Donzeau-Gouge [9] and Neilson [21,22,23].

Here is an example abstract denotation: Let $\sigma_0 = [[\![\mathtt{x}]\!] \mapsto \perp] \in \Sigma^\sharp$, that is, $\mathtt{x}$ might be any L^∞-value at all (because $\gamma(\perp) = L^\infty$):

$$\mathcal{C}^\sharp[\![\mathtt{if\ (isNil\ x :\ x = cons\ d0\ x),\ (isNonNil\ x :\ x = x)\ fi}]\!]\sigma_0$$

$$= (\mathcal{C}^\sharp[\![\mathtt{x = cons\ d0\ x}]\!] \circ \mathcal{G}^\sharp[\![\mathtt{isNil\ x}]\!])\sigma_0 \ \sqcup\ (\mathcal{C}^\sharp[\![\mathtt{x = x}]\!] \circ \mathcal{G}^\sharp[\![\mathtt{isNonNil\ x}]\!])\sigma_0$$

Now,

$$\mathcal{G}^\sharp[\![\texttt{isNil x}]\!])\sigma_0 = (\alpha_{\underline{\perp}} \circ \mathcal{G}[\![\texttt{isNil x}]\!] \circ \gamma_{Var})\sigma_0$$
$$= (\alpha_{\underline{\perp}} \circ \mathcal{G}[\![\texttt{isNil x}]\!])\{[[\![\texttt{x}]\!] \mapsto \ell] \mid \ell \in L^\infty\}$$
$$= \alpha_{\underline{\perp}}\{[[\![\texttt{x}]\!] \mapsto \mathit{nil}],\ \underline{\perp}\} = [[\![\texttt{x}]\!] \mapsto \mathit{nil}]$$

The abstracted guard calculates the abstract store that covers all stores that satisfy `isNil x`. A similar calculation demonstrates that $\mathcal{G}^\sharp[\![\texttt{isNonNil x}]\!])\sigma_0 = \alpha_{\underline{\perp}}(\{[[\![\texttt{x}]\!] \mapsto (d, \ell)] \mid \ell \in L^\infty\} \cup \{\underline{\perp}\}) = [[\![\texttt{x}]\!] \mapsto (d, \perp)]$. We complete the derivation:

$$\mathcal{C}^\sharp[\![\texttt{x = cons d0 x}]\!][[\![\texttt{x}]\!] \mapsto \mathit{nil}] \ \sqcup\ \mathcal{C}^\sharp[\![\texttt{x = x}]\!][[\![\texttt{x}]\!] \mapsto (d, \perp)]$$
$$= \ (\mathit{update}^\sharp\ [\![\texttt{x}]\!]\ (\mathcal{E}^\sharp[\![\texttt{cons d0 x}]\!][[\![\texttt{x}]\!] \mapsto \mathit{nil}])\ [[\![\texttt{x}]\!] \mapsto \mathit{nil}])\ \sqcup\ [[\![\texttt{x}]\!] \mapsto (d, \perp)]$$
$$= \ [[\![\texttt{x}]\!] \mapsto (\texttt{d0}, \mathit{nil})] \sqcup [[\![\texttt{x}]\!] \mapsto (d, \perp)]$$
$$= \ [[\![\texttt{x}]\!] \mapsto (\texttt{d0} \sqcup d,\ \mathit{nil} \sqcup_{L_k^{\top^{op}}} \perp)] \ = \ [[\![\texttt{x}]\!] \mapsto (\texttt{d0} \sqcup d,\ \perp)]$$

The outcomes are joined, precision is lost, and the result is an abstract store that maps `x` to a non-nil list whose head is $\texttt{d0} \sqcup d$ and whose tail is unknown (i.e., might be any L^∞-value at all).

The previous derivation demonstrates how an abstract intepretation is used in practice: a family of tests, covering the data sets of interest, are supplied to a program, and the outputs are calculated by derivation. Using this approach, one naturally wishes to unfold a higher-order abstract denotation of form, $f = \mathit{lfp}\ \lambda\sigma.F_{f\sigma'}$. But we must ensure finite and detectable termination of the unfolding and calculation.

A semantically sound technique for bounding the unfolding is explained in terms of "minimal function graph" semantics [18]: Starting from term, $f(\sigma_0)$, we generate the subsequent calls (unfoldings), $f(\sigma_i)$, in the process constructing a family of k *first-order* equations,

$$f\sigma_0 = F_{f\sigma_1}$$
$$f\sigma_1 = F_{f\sigma_2}$$
$$\cdots$$
$$f\sigma_k = F_{f\sigma_j},\ \text{for some } j \leq k$$

which we can solve iteratively and can detect convergence. The equation set is guaranteed to be finite if the abstract domain from which σ is taken is finite (e.g., *Sign*, or $L_k^{\top^{op}}$).

If the abstract domain is not finite (e.g., *Const*), k can be forced to be finite by making the argument sequence, $\sigma_0, \sigma_1, \cdots, \sigma_k$, into a chain so that the domain's *finite-height* ensures a finite equation set. This is done by unfolding call $f(\sigma_i)$ until a call, $f(\sigma_i{}')$, is uncovered. This generates a new first-order equation for $f(\sigma_{i+1})$, where $\sigma_{i+1} = \sigma_i \sqcup \sigma_i{}'$. Since $f(\sigma_i{}') \sqsubseteq f(\sigma_{i+1})$, the solution to the former can be safely used in place of the latter. The abstract domain's finite height bounds the quantity of the generated equation set.

The use of $\sigma_{i+1} = \sigma_i \sqcup \sigma_i{}'$ does not suffice for an abstract domain like *Interval*, which possesses infinitely ascending chains. In this situation, $\sqcup$ is replaced by a

monotonic, extensive *widening function* that is guaranteed to generate chains of finite height only [5]. For the *Interval* domain, its widening function is defined $widen(\sigma_i, \sigma')$, where σ_i is the ith element in the chain under construction, and σ' is newly appearing in a call, $f(\sigma')$:

$$widen([], [c,d]) = [c,d]$$
$$widen([a,b],[c,d]) = [a,b], \text{ if } a \leq c \text{ and } d \leq b$$
$$widen([a,b],[c,d]) = [-\infty,b], \text{ if } c < a \text{ and } d \leq b$$
$$widen([a,b],[c,d]) = [a,+\infty], \text{ if } a \leq c \text{ and } b < d$$
$$widen([a,b],[c,d]) = [-\infty,+\infty], \text{ if } c < a \text{ and } b < d$$

Widening operations are also required when working with polyhedral domains.

When a chain of arguments is built during the process of generating the set of first-order equations it is common to solve just this one equation,

$$f\sigma_k = F_{f\sigma_k}$$

where σ_k is the last element in the generated chain.

Here is an example, for domain *Sign* and the semantics in Figure 8: For $\mathcal{C}^\sharp[\![\texttt{while NonNil x : x = tl x}]\!] = f$, where
$f(\sigma) = \mathcal{G}^\sharp[\![\texttt{Nil x}]\!]\sigma \sqcup f(\mathcal{C}^\sharp[\![\texttt{x = tl x}]\!](\mathcal{G}^\sharp[\![\texttt{NonNil x}]\!]\sigma))$,
we calculate an abstract test with σ_{db}: Let $\sigma_{db} = [\texttt{x} \mapsto (d, \bot)]$ and $\sigma_b = [\texttt{x} \mapsto \bot]$. (Please recall, in abstract domain $L_k^{\top\,op}$, that $\bot \in L_k^\top$ means "all lists," and $\top \in L_k^\top$ means "no lists.") $\mathcal{C}^\sharp[\![\texttt{while NonNil x : x = tl x}]\!]\sigma_{db} = f\sigma_{db}$, where

$$\begin{aligned} f\sigma_{db} &= \mathcal{G}^\sharp[\![\texttt{Nil x}]\!]\sigma_{db} \sqcup f(\mathcal{C}^\sharp[\![\texttt{x = tl x}]\!](\mathcal{G}^\sharp[\![\texttt{NonNil x}]\!]\sigma_{db}) \\ &= [\texttt{x} \mapsto \top] \sqcup f(\mathcal{C}^\sharp[\![\texttt{x = tl x}]\!]\sigma_{db}) \\ &= f\sigma_b \\ f\sigma_b &= \mathcal{G}^\sharp[\![\texttt{Nil x}]\!]\sigma_b \sqcup f(\mathcal{C}^\sharp[\![\texttt{x = tl x}]\!](\mathcal{G}^\sharp[\![\texttt{NonNil x}]\!]\sigma_b) \\ &= [\texttt{x} \mapsto nil] \sqcup f(\mathcal{C}^\sharp[\![\texttt{x = tl x}]\!]\sigma_{db}) \\ &= [\texttt{x} \mapsto nil] \sqcup f\sigma_b \end{aligned}$$

We solve these two first-order equations.

As noted earlier, the inductive definition format ensures soundness. This is because, for all phrase forms, E, we use the format, $\mathcal{E}[\![\texttt{op(E}_\texttt{i}\texttt{)}]\!] = f(\mathcal{E}[\![\texttt{E}_\texttt{i}]\!])$ to define E's semantics, and we define the abstract semantics inductively as $\mathcal{E}^\sharp[\![\texttt{op(E}_\texttt{i}\texttt{)}]\!] = f_0^\sharp(\mathcal{E}^\sharp[\![\texttt{E}_\texttt{i}]\!])$, where $f_0^\sharp = \alpha \circ f \circ \gamma$.

Soundness is stated as $\mathcal{E}[\![\texttt{E}]\!] \circ \gamma = \gamma \circ \mathcal{E}^\sharp[\![\texttt{E}]\!]$ (equivalently stated as $\alpha \circ \mathcal{E}[\![\texttt{E}]\!] = \mathcal{E}^\sharp[\![\texttt{E}]\!] \circ \alpha$). It is easy to prove that $\mathcal{E}^\sharp$ is sound for $\mathcal{E}$.

Recall the two notions of completeness:

forwards completeness: For all E, $\mathcal{E}[\![\texttt{E}]\!] \circ \gamma = \gamma \circ \mathcal{E}^\sharp[\![\texttt{E}]\!]$
backwards completeness: For all E, $\alpha \circ \mathcal{E}[\![\texttt{E}]\!] = \mathcal{E}^\sharp[\![\texttt{E}]\!] \circ \alpha$

As proved by Cousot and Cousot [6], both forms of completeness are preserved by least- and greatest-fixed-point constructions, as well as by function composition and by inductive definition on syntax: If for every equation, $\mathcal{E}[\![\mathtt{op}(\mathtt{E_i})]\!] = f(\mathcal{E}[\![\mathtt{E_i}]\!])$, $f_0^\sharp$ is forwards (resp. backwards) complete for f, then $\mathcal{E}^\sharp$ is forwards (resp. backwards) complete for $\mathcal{E}$.

When there is not completeness, the inductive definition of $\mathcal{E}^\sharp$ is sound but may be weaker than the strongest abstract interpretation: $\mathcal{E}^\sharp[\![\mathtt{E}]\!] \sqsupseteq \alpha \circ \mathcal{E}[\![\mathtt{E}]\!] \circ \gamma$.

It is puzzling that there are *two* forms of completeness. Both imply strongest postcondition properties, but the two notions are inequivalent [10]. What are they exactly? We learn the answer by considering the topology induced from the Galois connection: a forwards-complete function is is a topologically closed map and a backwards-complete function is a topologically continuous map.

8 Topological characterization of completeness

Topology plays a key role in denotational semantics. To solve the domain equation, $D = D \to D$, Scott needed to limit the cardinality of functions on D. Topological continuity was the appropriate criterion: For complete lattice L, Scott defined L's open sets to be those subsets of L that are (i) upwards closed and (ii) closed under tails of chains. [4] The functions that are topologically continuous for the Scott-topology of L are exactly the chain-continuous functions on L. Continuity limited the cardinality of $D \to D$ so that the recursive domain equation had a solution.

Consider the Scott-topology on an algebraic bcpo: D is algebraic if there is a subset, $F_D \subseteq D$, of finite elements [5] such that for every $d \in D$, $d = \sqcup\{e \in F_D \mid e \sqsubseteq d\}$ Each $e \in F_D$ defines the property of "having e-information level," and the basic open sets for D's Scott-topology are $\{\uparrow e \mid e \in F_D\}$. [6]

How does this relate to abstract interpretation? For abstract domain, $L_k^{\top\, op}$, its elements name properties that are used in an abstract interpretation: Each $\ell \in L_k^\top$ names the set, $\uparrow\ell \subseteq L^\infty$, a Scott-basic open set in L^∞. Indeed, the collection, $\gamma[L_k^\top]$, is a family that is closed under intersection but not necessarily under union. If we close under union, we have a topology on L^∞, coarser than the Scott-topology.

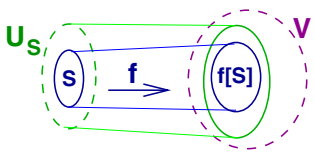
One defines a topology so to ask, "what are the continuous functions?" In the case of the family, $\gamma[L_k^{\top\, op}]$, we ask "what are the open/closed functions?" and "what are the continuous functions?". Even when $\gamma[L_k^{\top\, op}]$ is not a topology, we will see that those functions that preserve members of $\gamma[L_k^{\top\, op}]$ are exactly the forwards-complete functions of abstract-interpretation theory, and those functions that reflect members of $\gamma[L_k^{\top\, op}]$ are exactly the backwards-complete functions. We now develop these intuitions.

Let $\mathcal{P}(\Sigma)\langle\alpha, \gamma\rangle A$ be a Galois connection for Scott-domain Σ. Let $\mathcal{F}_\Sigma = \gamma[A] = (\gamma \circ \alpha)[\Sigma] \subseteq \mathcal{P}(\Sigma)$ define the properties of interest within Σ. For each $U \in \mathcal{F}_\Sigma$, its

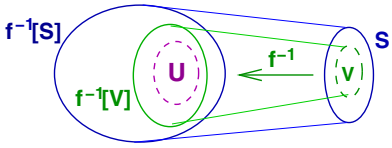
[4] That is, for every chain, $C = \{c_0, c_1, \cdots c_i, \cdots\} \subseteq L$, when $\sqcup C \in U$, for open set $U \subseteq L$, then there exists some $c_k \in C$ such that $c_k \in U$ also. This means C's tail, from c_k onwards, is in U.

[5] $e \in D$ is *finite* iff for all chains $C \subseteq D$, $e \sqsubseteq \sqcup C$ implies $e \sqsubseteq c$ for some $c \in C$.

[6] where $\uparrow e = \{d \in D \mid e \sqsubseteq d\}$ and $\uparrow S = \cup\{\uparrow e \mid e \in S\}$



(i) For $S \subseteq \Sigma$, f is *continuous at* S iff whenever $f[S] \subseteq V \in \mathcal{F}_\Sigma$, there exists some $U \in \mathcal{F}_\Sigma$ such that $S \subseteq U$ and $f[U] \subseteq V$.



(ii) For $S \subseteq \Sigma$, $f^{-1} : \mathcal{P}(\Sigma) \to \mathcal{P}(\Sigma)$ is *dual continuous at* S iff whenever $f^{-1}[S] \supseteq U \in \mathcal{F}_\Sigma$, then there exists $V \in \mathcal{F}_\Sigma$, $V \subseteq S$, such that $f^{-1}[V] \supseteq U$.

Fig. 9. Continuity and dual continuity at a set

complement is $\sim U = \Sigma - U$; for $\mathcal{F}_\Sigma$, its *complement family*, $\sim\mathcal{F}_\Sigma$, is $\{\sim U \mid U \in \mathcal{F}_\Sigma\}$.

$\mathcal{F}_\Sigma$ is an *open family* if it is closed under unions, and it is a *closed family* if it is closed under intersections. Every open family has an *interior* operation, ι, which computes the largest property contained within a set: $\iota : \mathcal{P}(\Sigma) \to \mathcal{F}_\Sigma$ is defined $\iota(S) = \cup\{U \in \mathcal{F}_\Sigma \mid U \subseteq S\}$. Dually, every closed family has a *closure* operation, ρ, which computes the smallest property covering a set: $\rho : \Sigma \to \mathcal{F}_\Sigma$ is defined $\rho(S) = \cap\{K \in \mathcal{F}_\Sigma \mid S \subseteq K\}$. If $\mathcal{F}_\Sigma$ is an open family, then its complement is a closed family (and vice versa). When we define $\mathcal{F}_\Sigma = \gamma[A]$ and γ is the upper adjoint of a Galois connection, then $\mathcal{F}_\Sigma$ is a closed family.

For $f : \Sigma \to \Sigma$, define $f : \mathcal{P}(\Sigma) \to \mathcal{P}(\Sigma)$ as $f[S] = \{f(s) \mid s \in S\}$, and define $f^{-1} : \mathcal{P}(\Sigma) \to \mathcal{P}(\Sigma)$ as $f^{-1}(T) = \{s \in \Sigma \mid f(s) \in T\}$, as usual.

f is $\mathcal{F}_\Sigma$-*preserving* iff for all $U \in \mathcal{F}_\Sigma$, $f[U] \in \mathcal{F}_\Sigma$. In such a case, $f : \mathcal{F}_\Sigma \to \mathcal{F}_\Sigma$ is well defined. This generalizes the notions of open and closed mappings on a topology. Since $\mathcal{F}_\Sigma = \gamma[A]$ is a closed family, we have immediately that $f : \Sigma \to \Sigma$ *is forwards complete iff it is* $\mathcal{F}_\Sigma$ *preserving, that is, iff it is a topologically closed map.*

We can characterize backwards completeness similarly. But first, we must generalize the definition of continuity so that it applies to property families that might not be topologies.

Let U_s (respectively, U_S) denote a member of $\mathcal{F}_\Sigma$ such that $s \in U_s$ (resp., $S \subseteq U_S$):

- For $c \in \Sigma$, $f : \Sigma \to \Sigma$ is *continuous at* c iff for all $V_{f(c)} \in \mathcal{F}_\Sigma$, there exists some $U_c \in \mathcal{F}_\Sigma$ such that $f[U_c] \subseteq V_{f(c)}$.
- For $S \subseteq \Sigma$, f is *continuous at* S iff for all $V_{f[S]} \in \mathcal{F}_\Sigma$, there exists some $U_S \in \mathcal{F}_\Sigma$ such that $f[U_S] \subseteq V_{f[S]}$.
- f is $\mathcal{F}_\Sigma$-*reflecting* iff for all $V \in \mathcal{F}_\Sigma$, $f^{-1}(V) \in \mathcal{F}_\Sigma$, that is, f^{-1} is $\mathcal{F}_\Sigma$-preserving.

The second definition is needed because $\mathcal{F}_\Sigma$ might not be an open family. Figure 9, part (i), diagrams the notion of continuity at a set. If $\mathcal{F}_\Sigma$ is a topology, then all three notions are equivalent.

We retain these fundamental results:

Proposition 8.1 *[28]*

(i) *f is $\mathcal{F}_\Sigma$-reflecting iff f is continuous at S, for all $S \subseteq \Sigma$.*

(ii) *If $\mathcal{F}_\Sigma$ is an open family, then f is $\mathcal{F}_\Sigma$-reflecting iff f is continuous at s, for all $s \in \Sigma$.*

(iii) *$f : \Sigma \to \Sigma$ is $\sim\mathcal{F}_\Sigma$-reflecting iff f is $\mathcal{F}_\Sigma$-reflecting.*

For $S, S' \subseteq \Sigma$, write $S \leq_{\mathcal{F}_\Sigma} S'$ iff for all $K \in \mathcal{F}_\Sigma, S \subseteq K$ implies $S' \subseteq K$. This is the *specialization ordering* from topology. Write $S \equiv_{\mathcal{F}_\Sigma} S'$ iff $S \leq_{\mathcal{F}_\Sigma} S'$ and $S' \leq_{\mathcal{F}_\Sigma} S$.

The following definition is the usual one for abstract-interpretation backwards completeness: $f : \Sigma \to \Sigma$ is *backwards-$\mathcal{F}_\Sigma$-complete* iff for all $S, S' \subseteq \Sigma$, $S \equiv_{\mathcal{F}_\Sigma} S'$ implies $f[S] \equiv_{\mathcal{F}_\Sigma} f[S']$.

Lemma 8.2 *[28] Let ρ be the closure operator for closed family, $\mathcal{F}_\Sigma$. The following are equivalent:*

(i) *f is backwards-$\mathcal{F}_\Sigma$-complete;*

(ii) *for all $S \subseteq \Sigma$, $f[S] \equiv_{\mathcal{F}_\Sigma} f[\rho(S)]$;*

(iii) *$\rho \circ f = \rho \circ f \circ \rho$.*

Item (iii) lets us conclude:

Theorem 8.3 For closed family, $\mathcal{F}_\Sigma$, $f : \Sigma \to \Sigma$ is backwards-$\mathcal{F}_\Sigma$-complete iff f is $\mathcal{F}_\Sigma$-reflecting, that is, iff it is topologically continuous.

For domain L^∞ and its finite approximants, L_k, let's consider the relationship between the Scott-continuous functions, $f : L^\infty \to L^\infty$, and the backwards-complete functions for each $\mathcal{P}(L^\infty)\langle\alpha^k, \gamma^k\rangle L_k^{\top^{op}}$, $k \geq 0$. First, all functions f are trivially L_0-backwards complete (that is, backwards complete for $\mathcal{P}(L^\infty)\langle\alpha^0, \gamma^0\rangle L_0^{\top^{op}}$). Since the collection of property sets defined by $\gamma^k[L_k]$ is a subset of those for $\gamma^{k+1}[L_{k+1}]$, any L_k-backwards complete f is L_j-backwards complete for $j < k$.

Consider the domain defined in Figure 6:

- There is a Scott-continuous function, $f : L^\infty \to L^\infty$, that is not L_k-backwards complete for all $k > 0$. Define f as follows: $f(d^k, nil) = nil$, for all $k \geq 0$, and $f(\ell) = \bot$, otherwise. This function is Scott-continuous. Consider $f^{-1}\{nil\}$; it is exactly all the total, finite lists in L^∞, and for no finite element $e \in L^\infty$ does this set equal $\uparrow e$. (Nor does the union of the upclosed sets of finite elements in any L_k equal $f^{-1}(nil)$ — the union of the basic opens of *all* finite lists in L^∞ are required.)
- For each $k > 0$, there is a monotone, L_k-backwards complete function that is not Scott-continuous. For k, define $f_k : L^\infty \to L^\infty$ as follows: $f(\bot) = \bot$; for $j < k$, $f_k(d^j, nil) = (d^j, nil)$ and $f_k(d^j, \bot) = (d^j, \bot)$. For $j \geq k$, $f_k(d^j, nil) = (d^k, \bot)$; $f_k(d^j, \bot) = (d^k, \bot)$. Finally, define $f_k(d^\infty) = d^\infty$. This makes f_k monotone and backwards complete but Scott-discontinuous. The result does not change when the sets defined by L_k are closed under union.

These results are not surprising, because the property family for each L_k-domain is coarser than the Scott topology for the corresponding domain. They *are* frustrating, however, because they show how difficult it is to establish a homomorphism property from a concrete to an abstract denotational semantics.

Now, what about open families? Let $\mathcal{F}_\Sigma$ be open (closed under unions) and $\iota : \mathcal{P}(\Sigma) \to \mathcal{F}_\Sigma$ be its interior map.

In abstract interpretation, one uses an open family to perform an underapproximating *precondition analysis*: for $f : \Sigma \to \Sigma$, define $f^{-1} : \mathcal{P}(\Sigma) \to \mathcal{P}(\Sigma)$ as $f^{-1}(S) = \{s \in \Sigma \mid f(s) \in S\}$, as usual.

The strongest (*weakest precondition*) abstract function for f^{-1} is $\iota \circ f^{-1} : \mathcal{F}_\Sigma \to \mathcal{F}_\Sigma$.

This makes forwards-$\mathcal{F}_\Sigma$-completeness defined as $f^{-1} \circ \iota = \iota \circ f^{-1} \circ \iota$ and backwards-$\mathcal{F}_\Sigma$-completeness defined as $\iota \circ f^{-1} = \iota \circ f^{-1} \circ \iota$.

It is easy to understand forwards completeness: f^{-1} is $\mathcal{F}_\Sigma$-preserving iff f^{-1} is forwards-$\mathcal{F}_\Sigma$-complete iff f is $\sim\mathcal{F}_\Sigma$-reflecting iff f is $\mathcal{F}_\Sigma$-reflecting. This is the classic pre- post-condition duality of predicate transformers.

Backwards completeness for an open family and f^{-1} is a "dual continuity" property. Say that $f^{-1} : \mathcal{P}(\Sigma) \to \mathcal{P}(\Sigma)$ is *dual continuous* at $S \subseteq \Sigma$ iff for all $U \in \mathcal{F}_\Sigma$, if $f^{-1}[S] \supseteq U$ then there exists $V \in \mathcal{F}_\Sigma$, $V \subseteq S$, such that $f^{-1}[V] \supseteq U$. Figure 9, part (ii), depicts dual continuity at a set.

Theorem 8.4 *f^{-1} is dual continuous for all $S \subseteq \Sigma$ iff f^{-1} is backwards-$\mathcal{F}_\Sigma$-complete, that is, $\iota \circ f^{-1} = \iota \circ f^{-1} \circ \iota$.*

9 Conclusion

Abstract interpretation and denotational semantics share foundations and applications, and the interaction between the two areas is intricate. The inverse-limit construction and its associated Scott-topology show how to derive abstract domains as structural approximations of inverse-limit-defined Scott-domains. The inductive format of denotational semantics definitions ensures the soundness of the resulting abstract interpretation, where abstract domain replaces Scott-domain. In this manner, abstract interpretation can be explained from the perspective of denotational semantics.

Acknowledgement

Robert Tennent's depth of insight and clarity of presentation have been a continuing source of inspiration, and this paper is dedicated to him on the occasion of his 65th birthday.

The referees are thanked for their helpful comments.

References

[1] S. Abramsky. Domain theory in logical form. *Ann.Pure Appl.Logic*, 51:1–77, 1991.

[2] T. Ball, A. Podeksi, and S. Rajamani. Boolean and cartesian abstraction for model checking C programs. *J. Software Tools for Technology Transfer*, 5:49–58, 2003.

[3] P. Cousot. *Méthodes itératives de construction et d'approximation de points fixes d'opérateurs monotones sur un treillis, analyse sémantique de programmes.* PhD thesis, University of Grenoble, 1978.

[4] P. Cousot. Constructive design of a hierarchy of semantics of a transition system by abstract interpretation. *Theoretical Computer Science*, 277:47–103, 2002.

[5] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs. In *Proc. 4th ACM Symp. on Principles of Programming Languages*, pages 238–252. ACM Press, 1977.

[6] P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *Proc. 6th ACM Symp. on Principles of Programming Languages*, pages 269–282. ACM Press, 1979.

[7] P. Cousot and R. Cousot. Higher-order abstract interpretation. In *Proceedings IEEE Int. Conf. Computer Lang.*, 1994.

[8] P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In *Proc. 5th ACM Symp. on Principles of Programming Languages*, pages 84–96. ACM Press, 1978.

[9] V. Donzeau-Gouge. Denotational definition of properties of program's computations. In S. Muchnick and N.D. Jones, editors, *Program Flow Analysis: Theory and Applications*. Prentice-Hall, 1981.

[10] R. Giacobazzi and E. Quintarelli. Incompleteness, counterexamples, and refinements in abstract model checking. In *Static Analysis Symposium*, LNCS 2126, pages 356–373. Springer Verlag, 2001.

[11] R. Giacobazzi, F. Ranzato, and F. Scozzari. Making abstract interpretations complete. *J. ACM*, 47:361–416, 2000.

[12] G. Gierz, K.H. Hofmann, K. Keimel, J.D. Lawson, M.W. Mislove, and D.S. Scott. *Continuous Lattices and Domains*. Cambridge Univ. Press, 2003.

[13] S. Graf and H. Saidi. Construction of abstract state graphs with pvs. In *Proc. Conf. Computer Aided Verification*, pages 72–83. Springer LNCS 1254, 1997.

[14] C. Gunter. *Semantics of Programming Languages*. MIT Press, Cambridge, MA, 1992.

[15] T. Jensen. *Abstract Interpretation in Logical Form*. PhD thesis, Imperial College, London, 1992.

[16] P. Johnstone. *Stone Spaces*. Cambridge University Press, 1986.

[17] N.D. Jones and S. Muchnick. Flow analysis and optimization of LISP-like structures. In *Proc. 6th. ACM Symp. Principles of Programming Languages*, pages 244–256, 1979.

[18] N.D. Jones and A. Mycroft. Data flow analysis of applicative programs using minimal function graphs. In *Proc. 13th ACM Symp. on Principles of Prog. Languages*, pages 296–306, 1986.

[19] R. Milne and C. Strachey. *A Theory of Programming Language Semantics*. Chapman and Hall, 1976.

[20] A. Miné. The octagon abstract domain. *J. Higher-Order and Symbolic Computation*, 19:31–100, 2006.

[21] F. Nielson. A denotational framework for data flow analysis. *Acta Informatica*, 18:265–287, 1982.

[22] F. Nielson. Program transformations in a denotational setting. *ACM Trans. Prog. Languages and Systems*, 7:359–379, 1985.

[23] F. Nielson and H. R. Nielson. *Two-Level Functional Languages*. Cambridge University Press, 1992.

[24] F. Nielson, H.R. Nielson, and C. Hankin. *Principles of Program Analysis*. Springer Verlag, 1999.

[25] G. Plotkin. Domains. Lecture notes, Univ. Pisa/Edinburgh, 1983.

[26] J.C. Reynolds. Notes on a lattice-theoretic approach to the theory of computation. Technical report, Computer Science, Syracuse University, 1972.

[27] D.A. Schmidt. Comparing completeness properties of static analyses and their logics. In *Asian Symp. Prog. Lang. Systems (APLAS'06)*, LNCS 4279, pages 183–199. Springer Verlag, 2006.

[28] D.A. Schmidt. Abstract interpretation from a topological perspective. Technical report 2009-1, Computing and Information Science, Kansas State University, 2009.

[29] D.S. Scott and C. Strachey. Toward a mathematical semantics for computer languages. In J. Fox, editor, *Proceedings of Symposium on Computers and Automata*, pages 19–46. Microwave Research Institute Symposia Series: Volume 21, Polytechnic Institute of Brooklyn, 1971.

[30] M.B. Smyth. Powerdomains and predicate transformers: a topological view. In *Proc. ICALP'83*, LNCS 154, pages 662–675. Springer, 1983.

[31] R.D. Tennent. The denotational semantics of programming languages. *Comm. ACM*, 19:437–453, 1976.

[32] R.D. Tennent. Language design methods based on semantic principles. *Acta Informatica*, 8:97–112, 1977.