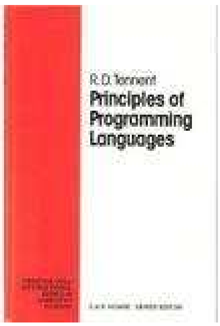
Abstract interpretation from a denotational semantics perspective

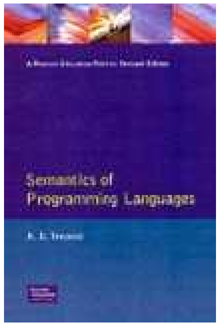
David Schmidt

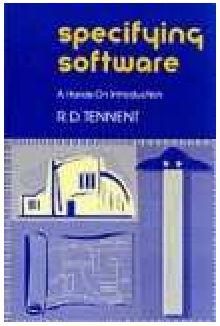
Kansas State University

`www.cis.ksu.edu/~schmidt`

Thank you, Bob Tennent...







for your contributions to programming-languages research!

The clarity and precision of your work is an inspiration, as is the care you take to ground your results in practice!

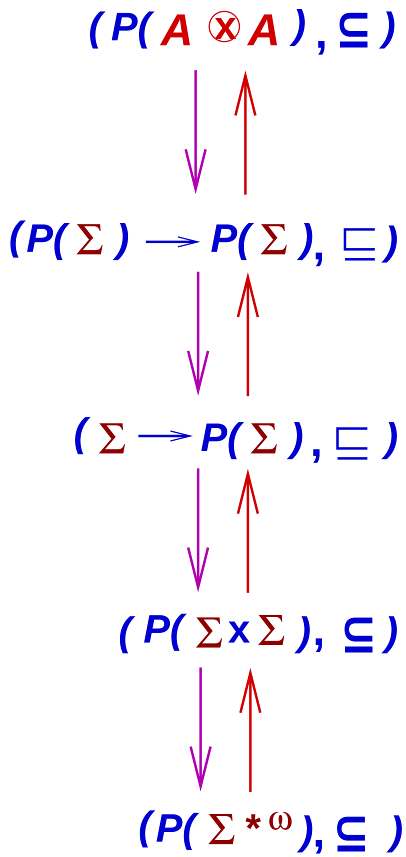
Patrick Cousot, MFPS 1997: “Denotational semantics is an abstract interpretation...”

Constructive Design of a Hierarchy of Semantics of a Transition System by Abstract Interpretation

Patrick Cousot[a]

[a]Département d'Informatique, École Normale Supérieure, 45 rue d'Ulm, 75230 Paris cedex 05, France, Patrick.Cousot@ens.fr, http://www.di.ens.fr/~cousot

We construct a hierarchy of semantics by successive abstract interpretations. Starting from the maximal trace semantics of a transition system, we derive the big-step semantics, termination and nontermination semantics, Plotkin's natural, Smyth's demoniac and Hoare's angelic relational semantics and equivalent nondeterministic denotational semantics (with alternative powerdomains to the Egli-Milner and Smyth constructions), D. Scott's deterministic denotational semantics, the generalized and Dijkstra's conservative/liberal predicate transformer semantics, the generalized/total and Hoare's partial correctness axiomatic semantics and the corresponding proof methods. All the semantics are presented in a uniform fixpoint form and the correspondences between these semantics are established through composable Galois connections, each semantics being formally calculated by abstract interpretation of a more concrete one using Kleene and/or Tarski fixpoint approximation transfer theorems.



Data domains for transition relation/function, $\tau \subseteq \Sigma \times \Sigma$

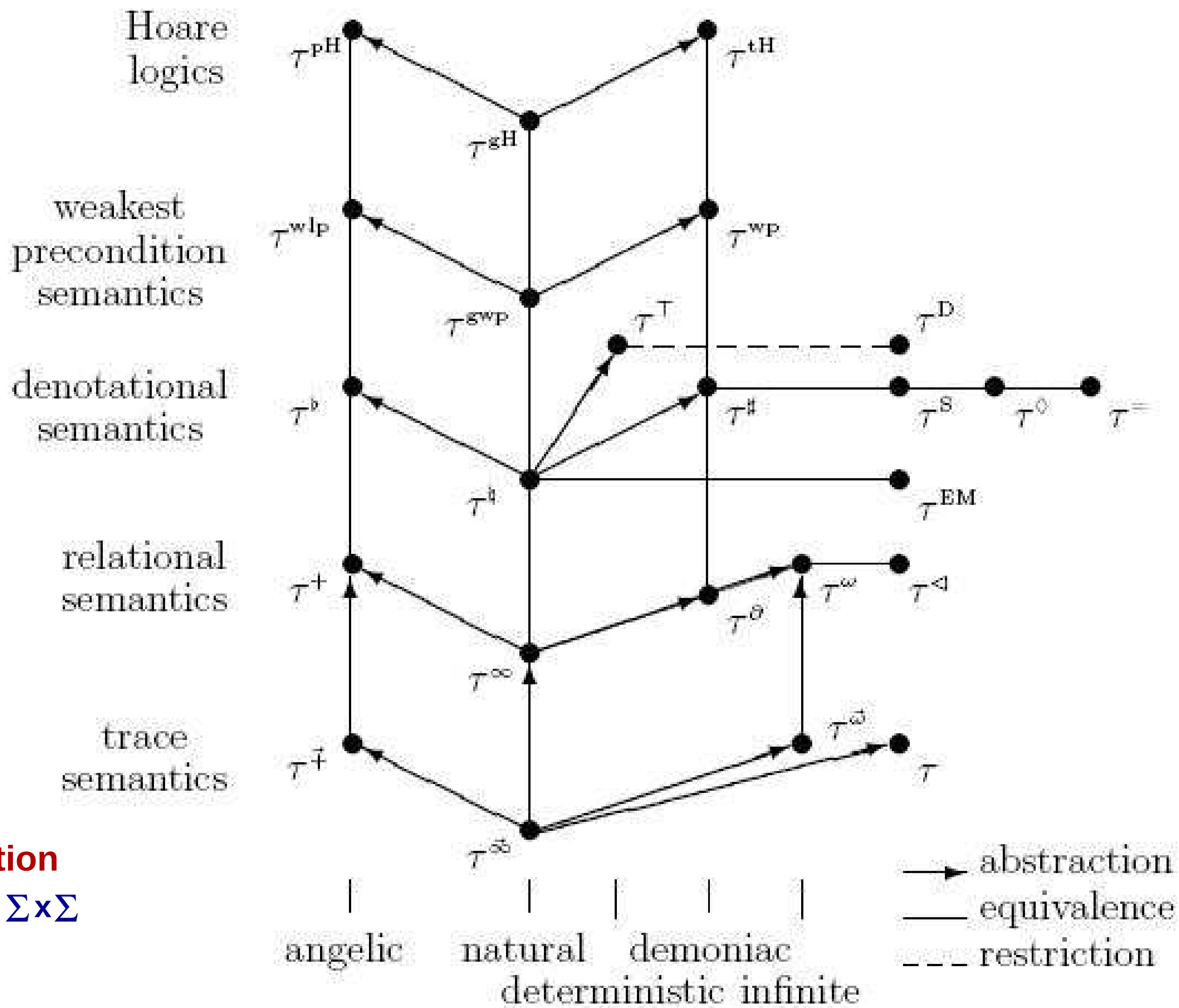
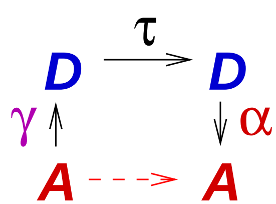


Fig. 4. The lattice of semantics

Abstract interpretation

finitely approximates a program's execution [Cousot78,Cousot^{2}77].

According to [Cousot97], it is the reinterpretation of a formal system, (τ, D), by an adjunction, $\boldsymbol{D} \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \boldsymbol{A}$, as $(\alpha \circ \tau \circ \gamma,\ A)$:



where A's elements finitely approximate D's.

Denotational semantics

defines a program's meaning extensionally (and inductively) a value from a *Scott domain* [ScottStrachey71,Tennent76].

In the sense of [Cousot97], it is a function,

$\mathcal{C} : \mathtt{Program} \rightarrow \mathrm{D}^\infty \rightarrow \mathrm{D}^\infty$,

from which one defines, for program $\mathtt{P}$, its formal system, $(\mathcal{C}[\![\mathtt{P}]\!],\ \mathrm{D}^\infty)$.

This talk shows how to use the approximation embedded within Scott domain D^∞ *to define abstract interpretation.* In this sense, "abstract interpretation is a denotational semantics...."

Background: abstract interpretation

Abstract interpretation = finite approximation

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

Q: *is the output* pos?

A: abstractly interpret input domain Int by $Sign = \{neg, zero, pos, any\}$:

$\mathsf{readSign}(\mathtt{x})$
`x =` $\mathsf{succ}^\sharp(\mathtt{x})$
`if` $(\mathsf{filterNeg}(\mathtt{x})$:
  `x =` $\mathsf{negate}^\sharp(\mathtt{x}))$
 $(\mathsf{filterNonNeg}(\mathtt{x})$:
  `x =` $\mathsf{succ}^\sharp(\mathtt{x}))$ `fi`
$\mathsf{writeSign}(\mathtt{x})$

where

$$\mathsf{succ}^\sharp(pos) = pos$$
$$\mathsf{succ}^\sharp(zero) = pos$$
$$\mathsf{succ}^\sharp(neg) = any \text{ (!)}$$
$$\mathsf{succ}^\sharp(any) = any$$

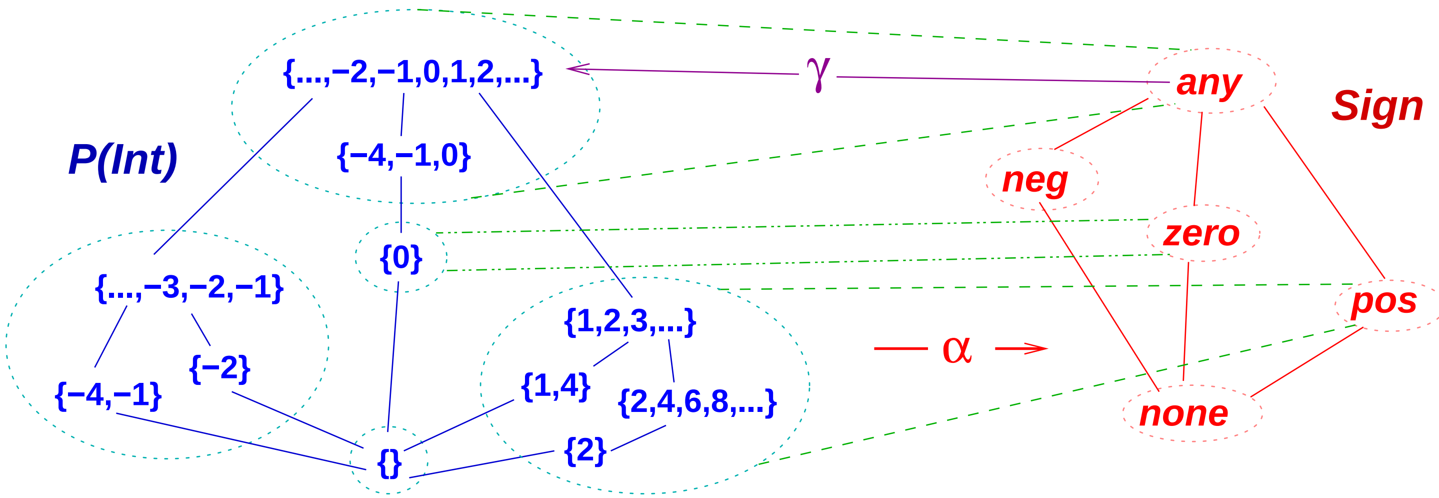
and

$$\mathsf{negate}^\sharp(neg) = pos$$
$$\mathsf{negate}^\sharp(zero) = zero$$
$$\mathsf{negate}^\sharp(pos) = neg$$
$$\mathsf{negate}^\sharp(any) = any$$

For the abstract data-test sets, $zero, neg, pos$***, we calculate:***

$\{zero \mapsto pos,\ pos \mapsto pos,\ neg \mapsto any\}$. The last result arises because $\mathsf{succ}^\sharp(neg) = any$ and $\mathsf{filterNeg}(any) = neg$ (good!) but $\mathsf{filterNonNeg}(any) = any$ (bad — we need $zero \vee pos$!), so we cannot ensure the success of the else-arm.

A Galois connection *formalizes the approximation*



$\gamma : Sign \rightarrow \mathcal{P}(Int)$

$\gamma(none) = \{\}, \quad \gamma(any) = Int$

$\gamma(neg) = \{\cdots, -3, -2, -1\}$

$\gamma(zero) = \{0\}, \quad \gamma(pos) = \{1, 2, 3, \cdots\}$

$\alpha : \mathcal{P}(Int) \rightarrow Sign$

$\alpha(S) = \sqcap\{a \mid \gamma(a) \subseteq S\}$

e.g., $\alpha\{2, 4, 6, 8, ...\} = pos$,

$\alpha\{-1, 0\} = any$, $\alpha\{0\} = zero$

$(\mathcal{P}(\mathrm{Int}), \subseteq)\langle\alpha, \gamma\rangle(Sign, \sqsubseteq)$ is a *Galois connection*:

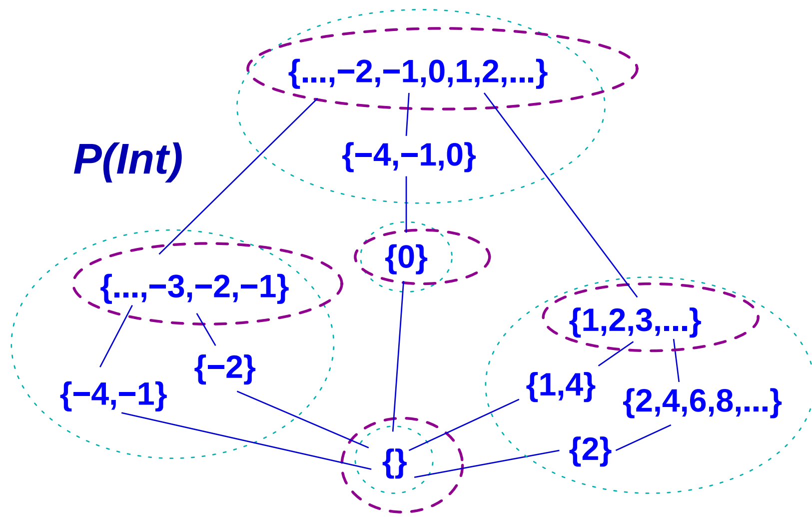
$$\alpha(S) \sqsubseteq a \text{ iff } S \subseteq \gamma(a).$$

γ interprets the elements in $Sign$, and α maps each data-test set in the *collecting domain*, $\mathcal{P}(Int)$, to the name that best describes the set [CousotCousot77].

The Galois connection defines a closure operator, $\rho = \gamma \circ \alpha : \mathcal{P}(\Sigma) \to \mathcal{P}(\Sigma)$

$\rho[\mathcal{P}(Int)] =$

$\{\{\}, \{\cdots, -2, -1\}, \{0\}, \{1, 2, \cdots\}, Int\}$

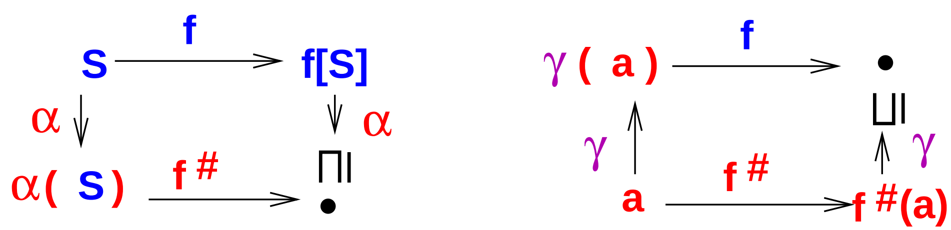


$\rho[\mathcal{P}(Int)]$ identifies the *properties* expressible in abstract domain $Sign$, and ρ maps a test set to its *minimal property*, e.g., $\rho\{1\} = \{1, 2, \cdots\}$, $\rho\{-1, 1\} = Int$, etc. Note that $\rho[\mathcal{P}(Int)]$ is *closed under intersection* (conjunction).

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.

Monotone, sound abstract functions

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



α and γ are *semi-homomorphisms*.

Example: The $succ^\sharp$ function seen earlier is sound for $succ$, e.g., for $succ : Int \rightarrow Int$, $succ[\{0\}] = \{1\}$, and $succ^\sharp(zero) = pos$.

Recall that $\rho[\mathcal{P}(\Sigma)] = \gamma[A]$ identifies the properties expressed by A.

When α is onto, we can treat $f^\sharp : A \rightarrow A$ as $f^\sharp : \rho[\mathcal{P}(\Sigma)] \rightarrow \rho[\mathcal{P}(\Sigma)]$.

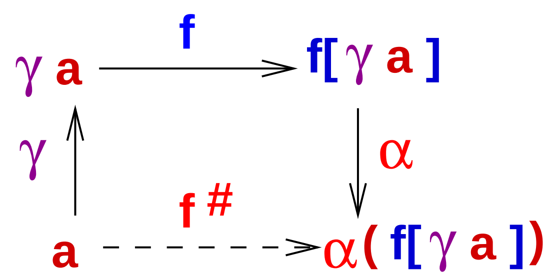
Example: $succ^\sharp\{0\} = \{1, 2, \cdots\}$.

Proposition: For all $\phi \in \rho[\mathcal{P}(\Sigma)]$, $f^\sharp$ is sound for f 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, $\iota : \mathcal{P}(\Sigma) \rightarrow \mathcal{P}(\Sigma)$.

Strongest abstract function

The strongest (most precise), sound $f^\sharp : A \to A$ for $f : \Sigma \to \Sigma$ is $f^\sharp_0 = \alpha \circ f \circ \gamma$, that is, $f^\sharp_0(a) = \alpha(f[\gamma(a)])$:



Example: The $succ^\sharp$ function seen earlier is strongest for $succ$.

We can define $f^\sharp_0$ in terms of $\rho = \gamma \circ \alpha$:

$$f^\sharp_0 = \rho \circ f : \rho[\mathcal{P}(\Sigma)] \to \rho[\mathcal{P}(\Sigma)], \quad \text{e.g., } succ^\sharp_0\{0\} = \{1, 2, \cdots\}.$$

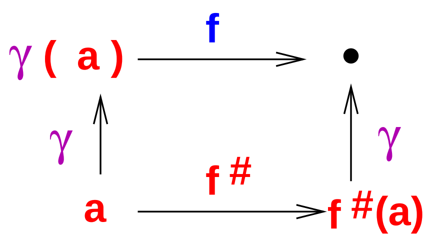
Proposition: (strongest postcondition for f): For all $\phi, \psi \in \rho[\mathcal{P}(\Sigma)]$, if $f(\phi) \subseteq \psi$, then $f^\sharp_0(\phi) \subseteq \psi$.

There is dual formulation, in terms of an interior map, ι, that generates the weakest precondition for f as $\iota \circ f^{-1}$.

Complete abstract functions

Forwards completeness

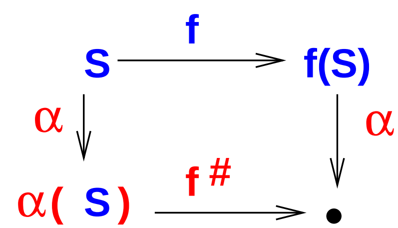
[Giacobazzi01]: $f \circ \gamma = \gamma \circ f^\sharp$



Backwards completeness

[Cousot²79,Giacobazzi00]:

$\alpha \circ f = f^\sharp \circ \alpha$



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

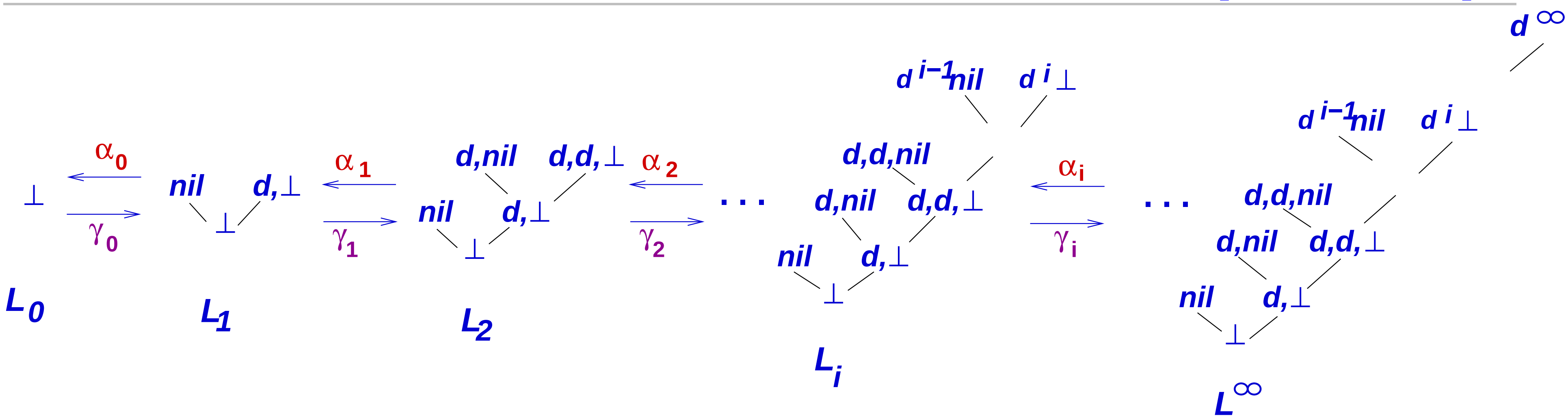
Proposition: TFAE: (i) $f_0^\sharp$ is forwards complete for f;
(ii) for all $\phi \in \rho[\mathcal{P}(\Sigma)]$, $f(\phi) \in \rho[\mathcal{P}(\Sigma)]$;
(iii) $f \circ \rho = \rho \circ f \circ \rho$.

Proposition: TFAE: (i) $f_0^\sharp$ is backwards complete for f;
(ii) for all $S_1, S_2 \in \mathcal{P}(\Sigma)$, $\rho(S_1) = \rho(S_2)$ implies $\rho(f[S_1]) = \rho(f[S_2])$;
(iii) $\rho \circ f = \rho \circ f \circ \rho$.

What do these results signify, really?

Background: denotational semantics

Inverse limit of $L^\infty \approx (\{nil\} + (D \times L^\infty)_\perp$ *(in* SFP^{ep}*)*



For $L_0 = \{\perp\}$, $L_{i+1} = (\{nil\} + (D \times L_i)_\perp$,
the embedding, projection pairs, $L_i\langle\gamma_i, \alpha_i\rangle L_{i+1}$, are defined

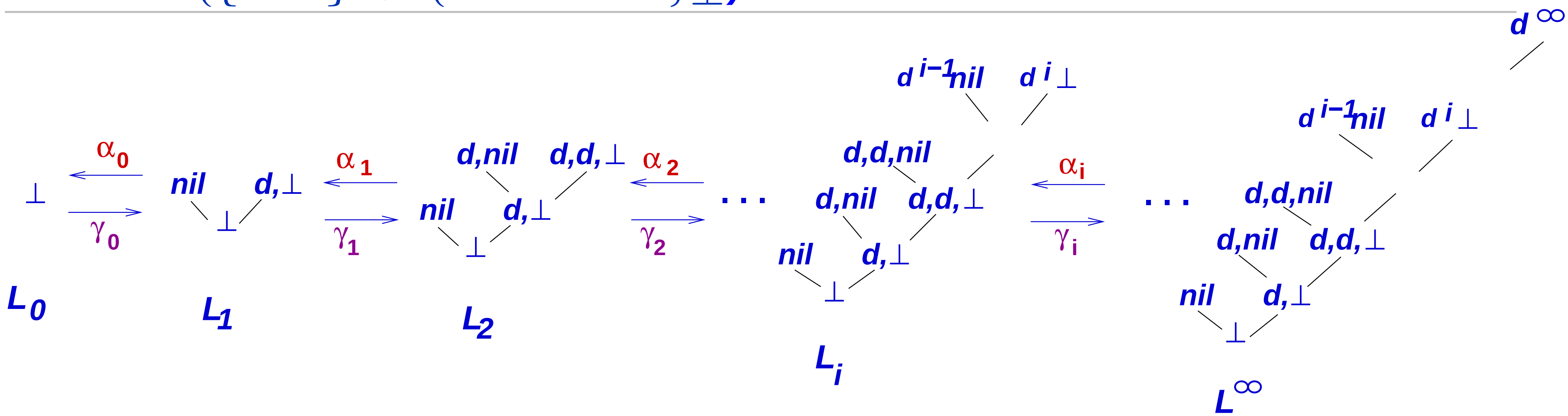
$$\gamma_0(\perp) = \perp \qquad \gamma_{i+1} = F(\gamma_i)$$
$$\alpha_0(\ell) = \perp \qquad \alpha_{i+1} = F(\alpha_i)$$

where

$$F(f)(\perp) = \perp$$
$$F(f)(nil) = nil$$
$$F(f)(d, \ell) = (d, f(\ell))$$

The e,p pairs compose into ones of form, $L_i\langle\gamma_{i,j}, \alpha_{j,i}\rangle L_j$, for $i < j$.

$L^\infty \approx (\{nil\} + (D \times L^\infty)_\perp$, *cont.*



The elements of L^∞ are tuples, $\langle \ell_i \rangle_{i \geq 0}$, such that each $\ell_i \in L_i$ and $\ell_i = \alpha_i(\ell_{i+1})$, for all $i \geq 0$.

For all $i \geq 0$, $L_i \langle \gamma_{i,\infty}, \alpha_{\infty,i} \rangle L^\infty$ are defined

$$\gamma_{i,\infty}(\ell) = \langle \alpha_{i-1,0}(\ell), \alpha_{i-1,1}(\ell), \cdots, \alpha_i(\ell), \ell, \gamma_i(\ell), \gamma_{i,i+2}(\ell), \gamma_{i,i+3}(\ell) \cdots \rangle$$

$$\alpha_{\infty,i}\langle \ell_0, \ell_1, \cdots, \ell_i, \cdots \rangle = \ell_i$$

$L^\infty \langle \gamma^\infty, \alpha^\infty \rangle (\{nil\} + (D \times L^\infty))_\perp$ forms an order-isomorphism, where

$$\gamma^\infty = \sqcup_{i \geq 0} F(\gamma_{i,\infty}) \circ \alpha_{\infty,i+1}$$

$$\alpha^\infty = \sqcup_{i \geq 0} \gamma_{i+1,\infty} \circ F(\alpha_{\infty,i})$$

A semantics definition based on L^∞

$\mathtt{d} \in \mathrm{Data}$(atomic data) $\mathtt{x} \in \mathit{Var}$(var names) $\mathtt{G} \in \mathrm{Guard}$(bool exprs)

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

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

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

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

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

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

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

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

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

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

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

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

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

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

A guard filters the store, like a logic gate; absence of store is denoted by $\underline{\perp}$.

Example: let $\sigma_0 = [[\![\mathtt{x}]\!] \mapsto \mathit{nil}]$. Then,

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

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

$$= \ \mathcal{C}[\![\mathtt{x} = \mathtt{cons\ d0\ x}]\!]\sigma_0 \ \sqcup \ \mathcal{C}[\![\mathtt{x} = \mathtt{x}]\!]\underline{\perp}$$

$$= \ (\mathit{update}\ [\![\mathtt{x}]\!]\ (\mathcal{E}[\![\mathtt{cons\ d0\ x}]\!]\sigma_0)\ \sigma_0) \ \sqcup \ \underline{\perp} \ = \ [[\![\mathtt{x}]\!] \mapsto (\mathtt{d0}, \mathit{nil})]$$

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

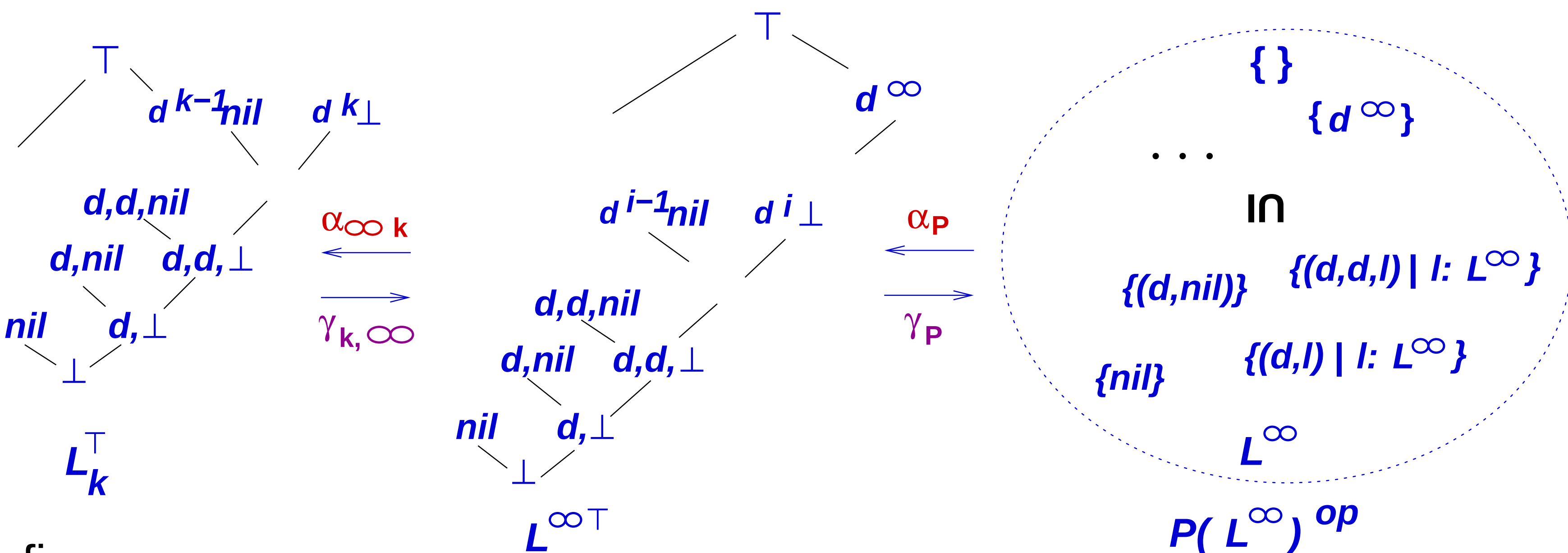
The while-command is a tail-recursive guarded-if, such that $\mathcal{C}[\![\mathtt{while\ B\ do\ C}]\!]$ equals $\mathcal{C}[\![\mathtt{if}\ (\neg\mathtt{B} : \mathtt{skip}),\ (\mathtt{B} : \mathtt{C}; (\mathtt{while\ B\ do\ C}))\ \mathtt{fi}]\!]$.

We write the semantics this way, because abstract-interpretation methodology treats programs as circuits and calculates information flows through them.

From denotational semantics to abstract interpretation

...the bridge is the collecting domain, $\mathcal{P}(\mathrm{L}^\infty)$

Intuition: an element, like $(d, \bot)$, approximates/describes the set, $\{(d, l) \mid l \in \mathrm{L}^\infty\} \in \mathcal{P}(\mathrm{L}^\infty)$:



Define

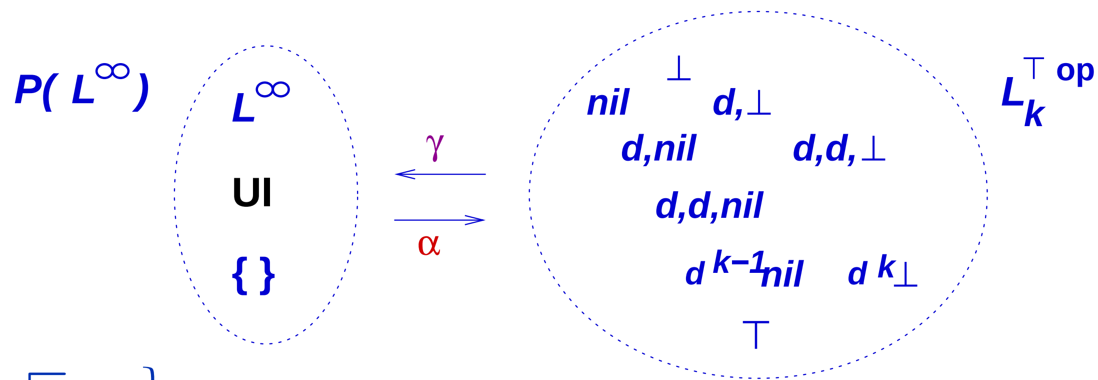
$\mathrm{L}_k^\top \langle \gamma, \alpha \rangle \mathcal{P}(\mathrm{L}^\infty)^{op}$ as $\begin{array}{l} \gamma = \gamma_P \circ \gamma_{k.\infty} \\ \alpha = \alpha_{\infty,k} \circ \alpha_P \end{array}$, where $\begin{array}{ll} \gamma_P(\ell) & = \uparrow\ell \\ & = \{m \in \mathrm{L}^\infty \mid \ell \sqsubseteq m\} \\ \alpha_P(S) & = \sqcap S \end{array}$

Each $l \in \mathrm{L}_k$ "names" the data-test set, $\gamma(l) = \uparrow l \in \mathcal{P}(\mathrm{L}^\infty)$

just like $pos \in \mathit{Sign}$ names $\{1, 2, \cdots\}$!

The rotated diagram yields a Cousot-style Galois connection — the notion of approximation is one and the same

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



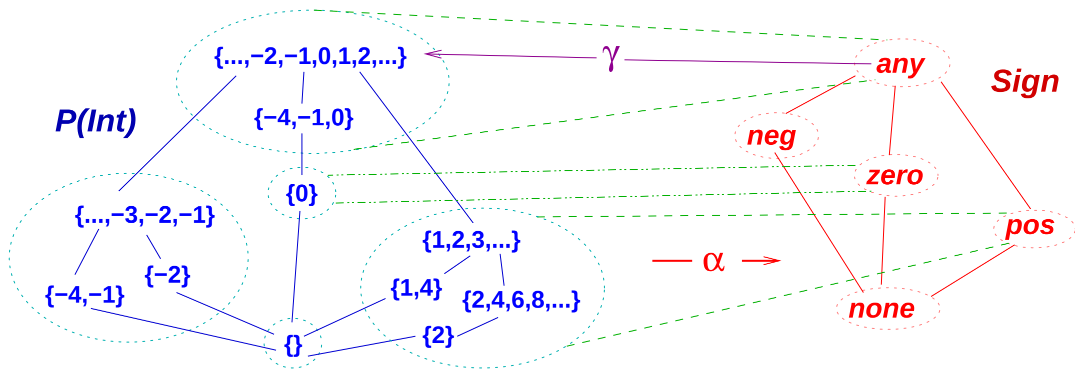
$\gamma(l) = \uparrow l = \{m \in L^\infty \mid l \sqsubseteq m\}$

$\alpha(S) = \bigsqcup_{L_k^\top}\{l \in L_k^\top \mid S \subseteq \gamma(l)\}$

- $(d^n, \bot) \in L_k^{\top\,op}$ names those lists having at least n-many elements; (d^n, nil) represents a list that has exactly n elements.
- $\bot \in L_k^{\top\,op}$ stands for all lists; $\top \in L_k^{\top\,op}$ for none.

One might also restrict the collecting domain to be just the *totally defined* lists or just the *finite, total* lists.

The Sign domain is derived from a Scott-domain:



$\mathrm{N} = \{1\}_\perp \oplus \mathrm{N}$ where $\oplus$ denotes disjoint sum with merged $\perp$s

$S = (\mathrm{N} + \{0\} + \mathrm{N})_\perp$

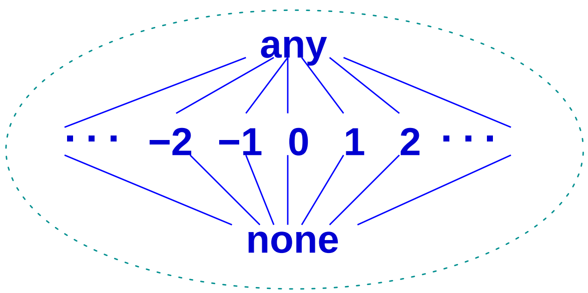
S denotes the integers partitioned into the negatives, zero, and the positives. The approximating domain,

$S_1 = (\mathrm{N}_0 + \{0\} + \mathrm{N}_0)_\perp$, where $\mathrm{N}_0 = \{\perp\}$, defines $\mathit{Sign} = S_1^{\top\,\mathrm{op}}$.

The collecting domain, $\mathcal{P}(\mathit{Int})$, holds sets of *total values* from S^∞.

We obtain better-precision signs-analyses from domains S_k, $k > 1$, which distinguish individual integers, e.g, $S_2^{\top\,\mathrm{op}} = \{\top, \mathit{neg}, -1, \mathit{zero}, 1, \mathit{pos}, \perp\}$.

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:

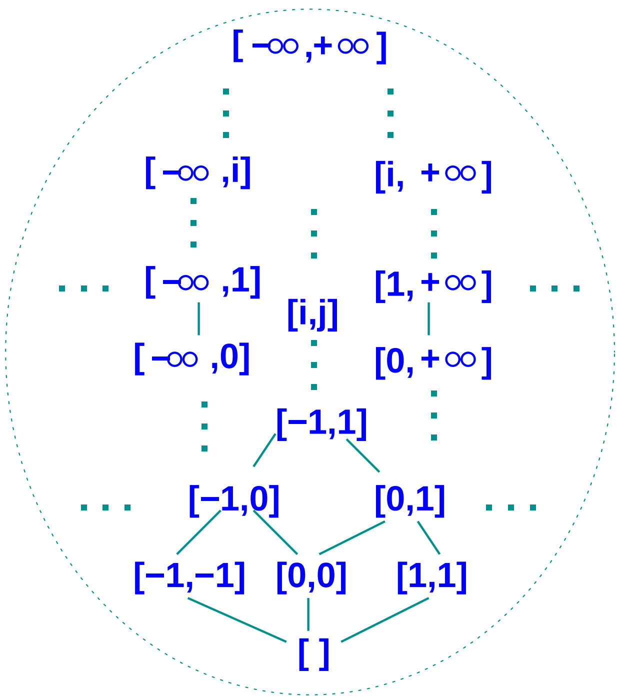


Const, *for constant-propagation analysis.*

Vars are analyzed to see if they are uninitialized ($\perp$), a constant (n), or hold multiple values ($\top$). This domain is $\mathrm{N}^{\infty\,\top op}$, where $\mathrm{N} = (\{0\} + \mathrm{N})_\perp$.

$\mathsf{Interval}$, *for tracking the range of values a variable is assigned.* Like Const, this domain is itself an inverse limit; its opposite is not in SFP.

Later, we will look at *relational abstract domains*, which abstract the entire store, $Var \to \Sigma$, rather than just the data domain, Σ.



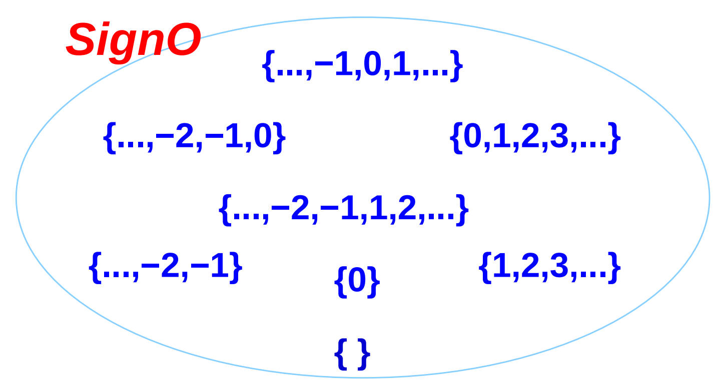
A bit of topology...

Domains like L^∞ are objects in category $\mathcal{SFP}$, and each $l \in L_k$ is a *finite element* that names the property, $\gamma(l) = \uparrow l \subseteq L^\infty$ — a *Scott-basic-open set*.

For $\rho = \gamma \circ \alpha$, $\rho[\mathcal{P}(L^\infty)]$ are all Scott-basic opens. This family is closed under intersection (because γ is an upper adjoint).

We can close $\rho[\mathcal{P}(L^\infty)]$ under unions, making a topology on L^∞ (coarser than L^∞'s Scott topology).

But this is exactly the *disjunctive completion* construction of abstract interpretation [Cousot[2]94], which is used to increase precision of an analysis!



But what are the topologically continuous functions in this setting?

A bit of logic...

Every abstract interpretation, A, has a logic of properties it can validate:

$$\psi ::= a \in A \mid f(\psi_i)_{0<i\leq n}$$

where $f : \Sigma^n \to \Sigma$ is a *logical operator*: $f[\gamma(\psi_i)]_{0<i\leq n} \in \gamma[A]$ that is, f maps properties to properties, on the nose

Fact: f is a logical operator iff it is forwards complete.

Since γ is an upper adjoint, $\sqcap$ is a logical operator; when $\gamma[A]$ defines a topology, $\sqcup$ is a logical operator. Here is the logic for $Sign$:

$$\phi ::= a \in Sign \mid \phi_1 \sqcap \phi_2 \mid negate(\phi), \quad \text{where } negate(n) = -n$$

But this developement is just Abramsky's *domain theory in logical form* [Abramsky91], where a domain's logic uses the finite/atomic elements of L^∞ as its A!

Jensen similarly defined *abstract interpretation in logical form* [Jensen92], where A is a *finite subset* of L^∞'s finite elements.

A language's abstract interpretation is its semantics where $A = L_k^{\top\,op}$ replaces L^∞

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

Galois connections:

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

$\Sigma = Var \to L^\infty$: $\mathcal{P}(\Sigma)\langle\alpha_{Var},\gamma_{Var}\rangle\Sigma^\sharp$ (indexed product)

$\Sigma_{\underline{\perp}}$: $\mathcal{P}(\Sigma_{\underline{\perp}})\langle\alpha_{\underline{\perp}},\gamma_{\underline{\perp}}\rangle\Sigma^\sharp$ (merges $\underline{\perp}$ with $\perp \in \Sigma^\sharp$)

$\mathcal{G}^\sharp : \text{Guard} \to \Sigma^\sharp \to \Sigma^\sharp$

$\mathcal{G}^\sharp[\![\texttt{G}]\!] = \alpha_{\underline{\perp}} \circ \mathcal{G}[\![\texttt{G}]\!] \circ \gamma_\Sigma$

$\mathcal{E}^\sharp : \text{Expression} \to \Sigma^\sharp \to L_k^{\top\,op}$

$\mathcal{E}^\sharp[\![\texttt{x}]\!]\sigma = \text{lookup}^\sharp\ [\![\texttt{x}]\!]\ \sigma$

where $\text{lookup}^\sharp\ v = \alpha \circ \text{lookup}\ v \circ \gamma_{Var}$, that is, $\text{lookup}^\sharp\ v\ \sigma = \sigma(v)$

$\mathcal{E}^\sharp[\![\texttt{tl E}]\!]\sigma = \text{tail}^\sharp(\mathcal{E}^\sharp[\![\texttt{E}]\!]\sigma)$

that is, $\text{tail}^\sharp(a,\ell) = \ell;\ \ \text{tail}^\sharp(\text{nil}) = \perp = \text{tail}^\sharp(\perp)$

$\mathcal{E}^\sharp[\![\texttt{cons a E}]\!]\sigma = \text{cons}^\sharp\ \texttt{a}\ (\mathcal{E}^\sharp[\![\texttt{E}]\!]\sigma)$

that is, $\text{cons}^\sharp\ a\ \ell = (a,\ell)$

abstract interpretation, cont.

Abstract store domain: $\sigma \in \Sigma^\sharp = \mathit{Var} \to \mathrm{L}_k^{\top\,op}$

Galois connections:

$$\begin{array}{ll} \mathrm{L}^\infty: & \mathcal{P}(\mathrm{L}^\infty)\langle\alpha,\gamma\rangle \mathrm{L}_k^{\top\,op} \\ \Sigma = \mathit{Var} \to \mathrm{L}^\infty: & \mathcal{P}(\Sigma)\langle\alpha_{\mathit{Var}},\gamma_{\mathit{Var}}\rangle\Sigma^\sharp \quad , \\ \Sigma_{\underline{\perp}}: & \mathcal{P}(\Sigma_{\underline{\perp}})\langle\alpha_{\underline{\perp}},\gamma_{\underline{\perp}}\rangle\Sigma^\sharp \end{array}$$

$\mathcal{C}^\sharp : \mathrm{Command} \to \Sigma^\sharp \to \Sigma^\sharp$

$\mathcal{C}^\sharp[\![\mathtt{x} = \mathtt{E}]\!]\sigma = \mathrm{update}^\sharp\ [\![\mathtt{x}]\!]\ (\mathcal{E}^\sharp[\![\mathtt{E}]\!]\sigma)\ \sigma$

where $\mathrm{update}^\sharp[\![\mathtt{x}]\!] = \alpha_{\underline{\perp}} \circ \mathrm{update}[\![\mathtt{x}]\!] \circ (\gamma \times \gamma_{\mathit{Var}})$,

that is, $\mathrm{update}^\sharp\ v\ \ell\ \sigma = \sigma + [v \mapsto \ell]$

$\mathcal{C}^\sharp[\![\mathtt{C}_1;\mathtt{C}_2]\!] = \mathcal{C}^\sharp[\![\mathtt{C}_2]\!] \circ \mathcal{C}^\sharp[\![\mathtt{C}_1]\!]$

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

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

We utilize the appropriate maps from the Galois connections to replace operations f by $f_0^\sharp = \alpha \circ f \circ \gamma$.

For the conditional, the guards filter the abstract store, and the results join together

Let $\sigma_0 = [[\![\mathtt{x}]\!] \mapsto \bot] \in \Sigma^\sharp$, that is, $\mathtt{x}$ might be any L^∞-value at all:

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

Now,
$$\mathcal{G}^\sharp[\![\mathtt{isNil\ x}]\!])\sigma_0 = (\alpha_{\underline{\bot}} \circ \mathcal{G}[\![\mathtt{isNil\ x}]\!] \circ \gamma_{Var})\sigma_0$$
$$= \alpha_{\underline{\bot}}\{[[\![\mathtt{x}]\!] \mapsto \mathrm{nil}],\ \underline{\bot}\} = [[\![\mathtt{x}]\!] \mapsto \mathrm{nil}]$$

and,
$$\mathcal{G}^\sharp[\![\mathtt{isNonNil\ x}]\!])\sigma_0 = \alpha_{\underline{\bot}}(\{[[\![\mathtt{x}]\!] \mapsto (d, \ell)] \mid \ell \in L^\infty\} \cup \{\underline{\bot}\})$$
$$= [[\![\mathtt{x}]\!] \mapsto (d, \bot)]$$

So, $\mathcal{C}^\sharp[\![\mathtt{x} = \mathtt{cons\ d0\ x}]\!][[\![\mathtt{x}]\!] \mapsto \mathrm{nil}] \sqcup \mathcal{C}^\sharp[\![\mathtt{x} = \mathtt{x}]\!][[\![\mathtt{x}]\!] \mapsto (d, \bot)]$

$$= (\mathrm{update}^\sharp\ [\![\mathtt{x}]\!]\ (\mathcal{E}^\sharp[\![\mathtt{cons\ d0\ x}]\!][[\![\mathtt{x}]\!] \mapsto \mathrm{nil}])\ [[\![\mathtt{x}]\!] \mapsto \mathrm{nil}]) \sqcup [[\![\mathtt{x}]\!] \mapsto (d, \bot)]$$
$$= [[\![\mathtt{x}]\!] \mapsto (\mathtt{d0}, \mathrm{nil})] \sqcup [[\![\mathtt{x}]\!] \mapsto (d, \bot)]$$ Note that the $\sqcup$ operates in $L_k^{\top\mathrm{op}}$.
$$= [[\![\mathtt{x}]\!] \mapsto (\mathtt{d0} \sqcup d,\ \bot)]$$

We use $\mathcal{C}^\sharp$ *for* abstract testing

Like the previous example, we supply an abstract test input and calculate its output. For denotations of form, $f = \mathit{lfp}\, \lambda\sigma.F_{f\sigma'}$, we must ensure detectable, finite convergence of tests, $f(\sigma)$.

We use “minimal function graph” semantics [JonesMycroft86]: Starting from $f(\sigma_0)$, we generate the subsequent calls, $f(\sigma_i)$, giving a family of k *first-order* equations,

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

which we solve iteratively.

If the abstract domain for σ is not finite (e.g., Const), k is forced 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. Then, it is common to solve just $f\sigma_k = F_{f\sigma_k}$.

Example: 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 $\sigma_{d\perp}$:

Let $\sigma_{d\perp} = [\texttt{x} \mapsto (d, \perp)]$
$\sigma_{\perp} = [\texttt{x} \mapsto \perp]$

(Note: in abstract domain $L_k^{\top\,op}$, $\perp \in L_k^{\top}$ means "all lists," and $\top \in L_k^{\top}$ means "no lists.")

$\mathcal{C}^\sharp[\![\texttt{while NonNil x : x = tl x}]\!]\sigma_{d\perp} = f\sigma_{d\perp}$, where

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

We solve these two first-order equations.

The inductive definition preserves soundness and completeness

For the format, $\mathcal{E}[\![\mathtt{op}(\mathtt{E_i})]\!] = \mathsf{f}(\mathcal{E}[\![\mathtt{E_i}]\!])$, we define the abstract semantics inductively as $\mathcal{E}^\sharp[\![\mathtt{op}(\mathtt{E_i})]\!] = \mathsf{f}_0^\sharp(\mathcal{E}^\sharp[\![\mathtt{E_i}]\!])$, where $\mathsf{f}_0^\sharp = \alpha \circ \mathsf{f} \circ \gamma$.

It is easy to prove that $\mathcal{E}^\sharp$ is sound for $\mathcal{E}$.

Recall
F-completeness: For all $\mathtt{E}$, $\mathcal{E}[\![\mathtt{E}]\!] \circ \gamma = \gamma \circ \mathcal{E}^\sharp[\![\mathtt{E}]\!]$
B-completeness: For all $\mathtt{E}$, $\alpha \circ \mathcal{E}[\![\mathtt{E}]\!] = \mathcal{E}^\sharp[\![\mathtt{E}]\!] \circ \alpha$

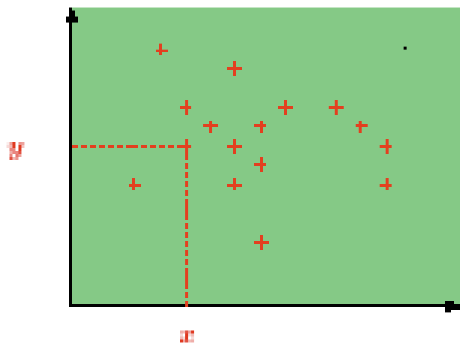
Proposition: If for every equation, $\mathcal{E}[\![\mathtt{op}(\mathtt{E_i})]\!] = \mathsf{f}(\mathcal{E}[\![\mathtt{E_i}]\!])$, $\mathsf{f}_0^\sharp$ is F- (resp. B-) complete for f, then $\mathcal{E}^\sharp$ is F- (resp. B-) complete for $\mathcal{E}$.

This result is preserved when lfp and gfp are used.

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

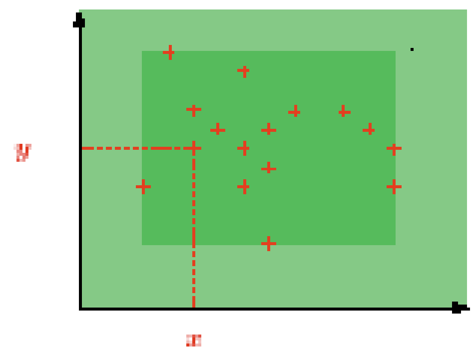
A store domain, $Var \rightarrow \Sigma$***, can be abstracted*** **pointwise *by*** $Var \rightarrow A$ ***or*** **relationally *by*** $\mathcal{P}(A^n)$

SignO: $[x \mapsto \geq 0][y \mapsto \geq 0]$



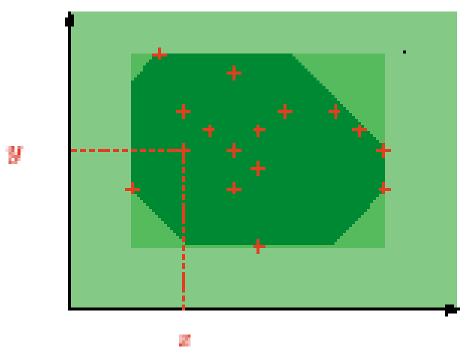
$$\begin{cases} x \geq 0 \\ y \geq 0 \end{cases}$$

Interval: $[x \mapsto [3, 27]][[y \mapsto [4, 32]]$



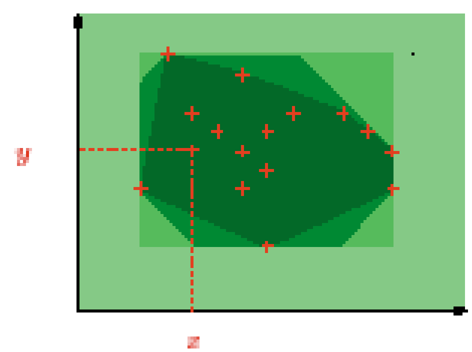
$$\begin{cases} x \in [3,\ 27] \\ y \in [4,\ 32] \end{cases}$$

Octagon: $\bigwedge_i (\pm x_i \pm y_i \leq c_i)$



$$\begin{cases} 3 \leq x \leq 27 \\ x + y \leq 88 \\ 4 \leq y \leq 32 \\ x - y \leq 61 \end{cases}$$

Polyhedra: $\bigwedge_i ((\sum_j a_{ij} \cdot x_{ij}) \leq b_i)$



$$\begin{cases} 7x + 31y \leq 325 \\ 21x + 7y \geq 0 \end{cases}$$

diagrams from *Abstract Interpretation: Achievements and Perspectives* by Patrick Cousot, Proc. SSGRR 2000.

Some modellings of a relational store value from the octagon abstract domain:

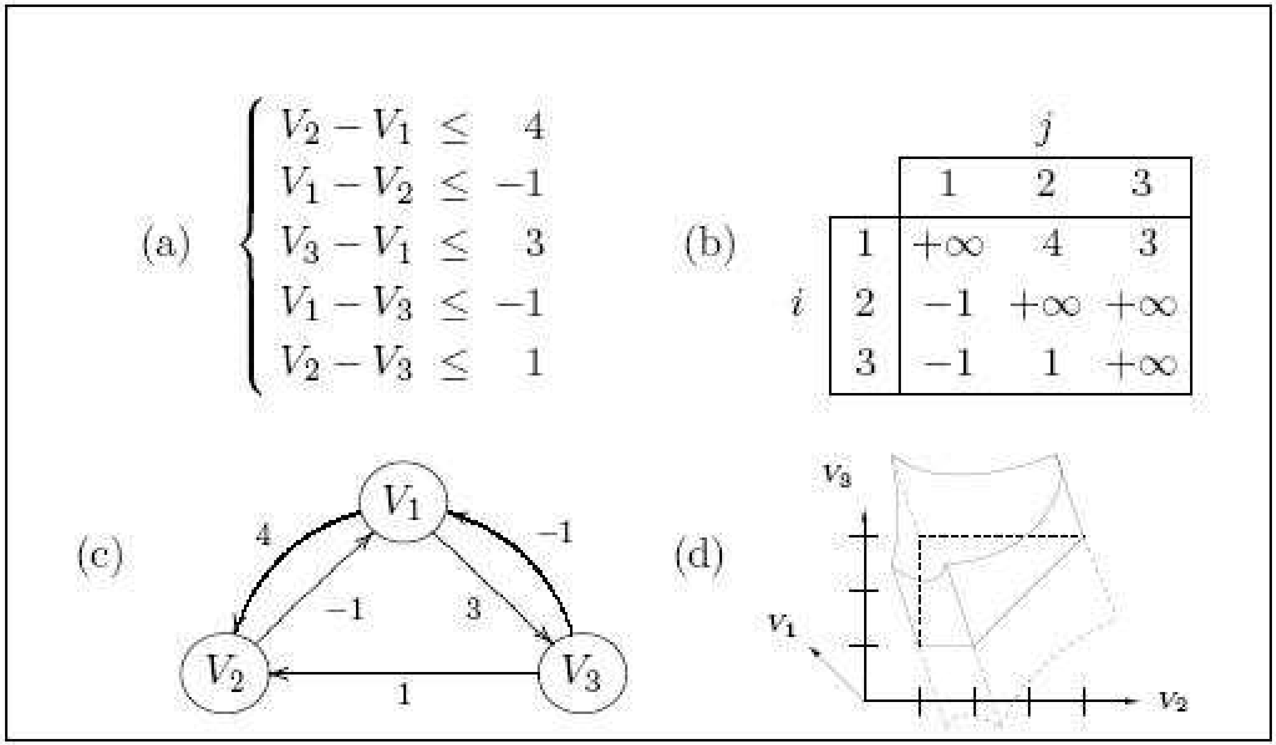


Figure 2. A potential constraint conjunction (a), its corresponding DBM **m** (b), potential graph $\mathcal{G}(\mathbf{m})$ (c), and potential set concretization $\gamma^{Pot}(\mathbf{m})$ (d).

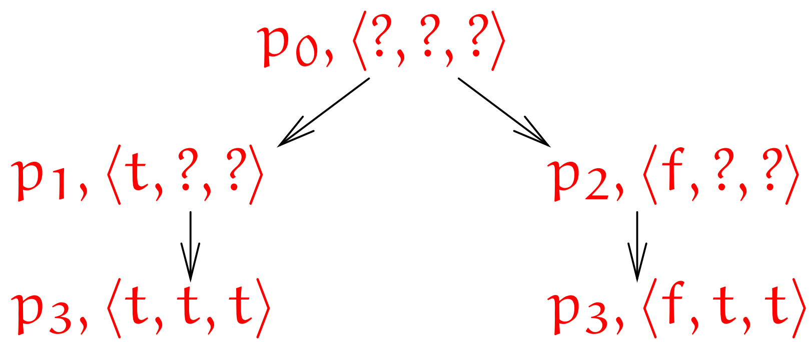
diagram from *The octagon abstract domain*, by Antoine Miné, *J. Symbolic and Higher-Order Computation* 2006

Octogan and polyhedral values can *perhaps* be explained in terms of Abramsky-Jensen “abstract interpretation in logical form.”

Predicate abstraction *uses an ad-hoc relational domain, based on predicates in the program*

Example: prove that $z \geq x \wedge z \geq y$ at p_3:

```
p0 : if x < y
p1 :   then z = y
p2 :   else z = x
p3 : exit
```



The store is abstracted to a relational domain that denotes the values of these predicates, taken from the source program,

$$\phi_1 = x < y \qquad \phi_2 = z \geq x \qquad \phi_2 = z \geq y$$

The predicates are evaluated at the program's points as one of $\{t, f, ?\}$. (Read $?$ as $t \vee f$.)

At all occurrences of p_3 in the abstract trace, $\phi_2 \wedge \phi_3$ holds.

When a goal is undecided, **domain refinement** *becomes necessary*

Prove $\phi_0 \equiv \mathtt{x} \geq \mathtt{y}$ at p_4:

p_0 : `if !(x >= y)`
p_1 : `then { i = x;`
  p_2 : `x = y;`
  p_3 : `y = i;`
p_4 : `}`

$p_0, \langle ? \rangle$ → $p_1, \langle f \rangle$; $p_4, \langle t \rangle$

$p_1, \langle f \rangle$
$p_2, \langle f \rangle$
$p_3, \langle t \rangle$
$p_4, \langle ? \rangle$

To decide the goal, we refine the ad-hoc domain:

$wp(\mathtt{y} = \mathtt{i}, \mathtt{x} \geq \mathtt{y}) = (\mathtt{x} \geq \mathtt{i}) \equiv \phi_1$. We add ϕ_1 and try again:

$p_0, \langle ?, ? \rangle$ → $p_1, \langle f, ? \rangle$; $p_4, \langle t \rangle$

$p_1, \langle f, ? \rangle$
$p_2, \langle f, t \rangle$
$p_3, \langle t, t \rangle$ — because $\mathtt{x} \not\geq \mathtt{y}$ and $\mathtt{x} \geq \mathtt{i}$ imply $\mathtt{y} > \mathtt{i}$ implies $\mathtt{x}_{new} \geq \mathtt{i}$
$p_4, \langle t, t \rangle$

But incremental predicate refinement cannot synthesize many interesting loop invariants. For this example:

```
p0 : i = n; x = 0;
p1 : while  i != 0  {
  p2 : x = x + 1;  i = i - 1;
     }
p3 : goal: x = n
```

The initial predicate set, $P_0 \equiv \{i = 0, x = n\}$, does not validate the loop body.

The first refinement suggests we add $P_1 \equiv \{i = 1, x = n - 1\}$ to the program state, but this fails to validate a loop that iterates more than once.

Refinement stage j adds predicates $P_j \equiv \{i = j, x = n - j\}$; the refinement process continues forever!

The loop invariant is $x = n - i$:-)

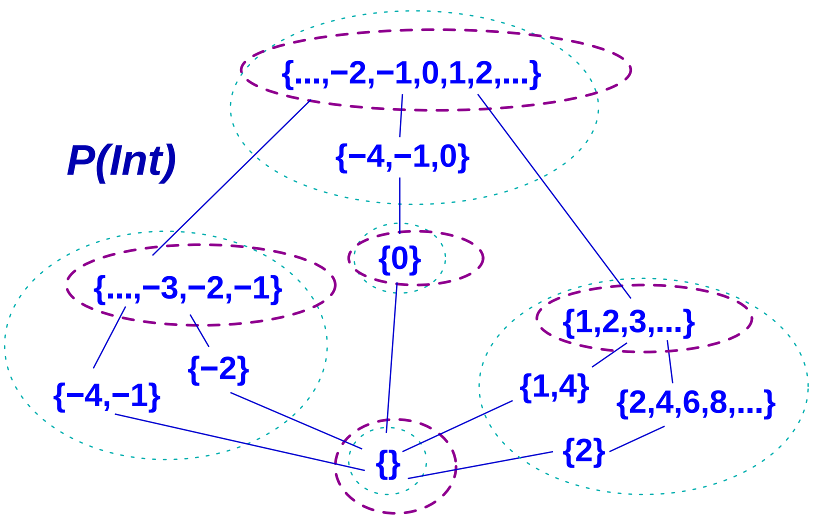
Explaining abstract-interpretation completeness with (a bit of) Scott topology

Open sets are computable properties [Smyth]

For an algebra cpo, D, its Scott-basic-open sets are $\uparrow e$, for each finite element, $e \in \mathrm{D}$. *Read* $\mathrm{d} \in \uparrow e$ *as "*d *has property* $\uparrow e$*."*

But abstract intepretation is *finite computation on properties*; for an abstract domain, like $Sign$, $\gamma[Sign]$ (or, $\rho[\mathcal{P}(Sign)]$) identifies the computable properties.

Alas, $\rho[\mathcal{P}(Sign)]$ is closed under intersections (not necessarily unions). Also, there exist abstract domains A that possess *only* a γ but no α (and no ρ) [Cousot^{2}92].



Let's weaken some definitions

For abstract domain A and $\gamma : A \to \mathcal{P}(\Sigma)$, define Σ's *property family* as $\mathcal{F}_\Sigma = \gamma[A]$.

For each $U \in \mathcal{F}_\Sigma$, its 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. If $\mathcal{F}_\Sigma$ is an open family, then its complement is a closed family (and vice versa).

When γ is the upper adjoint of a Galois connection, then $\mathcal{F}_\Sigma$ is a closed family.

Intuition: closed families are used for overapproximating, postcondition abstract interpretations; open families are used for underapproximating, precondition abstract interpretations.

Property preservation

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 topologically open and closed maps.

Let $\mathcal{F}_\Sigma$ be a closed family, and let $\rho : \mathcal{P}(\Sigma) \to \mathcal{P}(\Sigma)$ be the associated closure operator.

For $f : \Sigma \to \Sigma$, define $f_0^\sharp : \mathcal{P}(\Sigma) \to \mathcal{P}(\Sigma)$ as $f_0^\sharp = \rho \circ f$, as usual.

Fact: $f_0^\sharp$ is forwards complete for f iff f is $\mathcal{F}_\Sigma$ preserving, that is, iff f is a topologically closed map.

Property reflection (continuity)

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

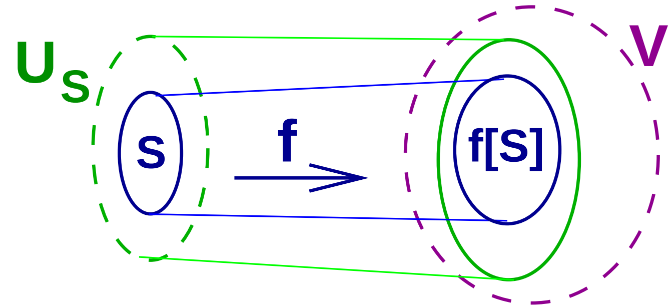
- For $c \in \Sigma$, $f : \Sigma \to \Sigma$ is *continuous at* c iff for all $V_{f(c)} \in \mathcal{F}_\Sigma$, there exists some $U_c \in \mathcal{F}_\Sigma$ such that $f[U_c] \subseteq V_{f(c)}$.
- For $S \subseteq \Sigma$, f is *continuous at* S iff for all $V_{f[S]} \in \mathcal{F}_\Sigma$, there 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 item is needed because $\mathcal{F}_\Sigma$ might not be an open family.

If $\mathcal{F}_\Sigma$ is a topology, then all three notions are equivalent.

reflection, cont.

f is continuous at $S \subseteq \Sigma$:



If $f[S] \subseteq V \in \mathcal{F}_\Sigma$, then there exists $U_S \in \mathcal{F}_\Sigma$ such that $f[U_S] \subseteq V$.

Proposition:

1. f is $\mathcal{F}_\Sigma$-reflecting iff f is continuous at S, for all $S \subseteq \Sigma$.
2. If $\mathcal{F}_\Sigma$ is an open family, then f is $\mathcal{F}_\Sigma$-reflecting iff f is continuous at c, for all $c \in \Sigma$.
3. $f : \Sigma \to \Sigma$ is $\sim\mathcal{F}_\Sigma$-reflecting iff f is $\mathcal{F}_\Sigma$-reflecting.

reflection, concl.

For $S, S' \subseteq \Sigma$, write $S \leq_{\mathcal{F}_\Sigma} S'$ iff for all $K \in \mathcal{F}_\Sigma$, $S \subseteq K$ implies $S' \subseteq K$. Write $S \equiv_{\mathcal{F}_\Sigma} S'$ iff $S \leq_{\mathcal{F}_\Sigma} S'$ and $S' \leq_{\mathcal{F}_\Sigma} S$. That is, S and S' share the same properties.

Definition: $f : \Sigma \to \Sigma$ is *backwards-$\mathcal{F}_\Sigma$-complete* iff for all $S, S' \subseteq \Sigma$, $S \equiv_{\mathcal{F}_\Sigma} S'$ implies $f[S] \equiv_{\mathcal{F}_C} f[S']$ cf. Slide 12.

Proposition: If f is $\mathcal{F}_\Sigma$-reflecting, then it is backwards-$\mathcal{F}_\Sigma$-complete.

Lemma: If $\mathcal{F}_\Sigma$ is a closed family, then TFAE:

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

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

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

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

So, abstract-interpretation backwards completeness is topological continuity.

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.

We use 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 $\iota \circ f^{-1} : \mathcal{F}_\Sigma \to \mathcal{F}_\Sigma$.

Define *F-$\mathcal{F}_\Sigma$-completeness:* $f^{-1} \circ \iota = \iota \circ f^{-1} \circ \iota$
B-$\mathcal{F}_\Sigma$-completeness: $\iota \circ f^{-1} = \iota \circ f^{-1} \circ \iota$

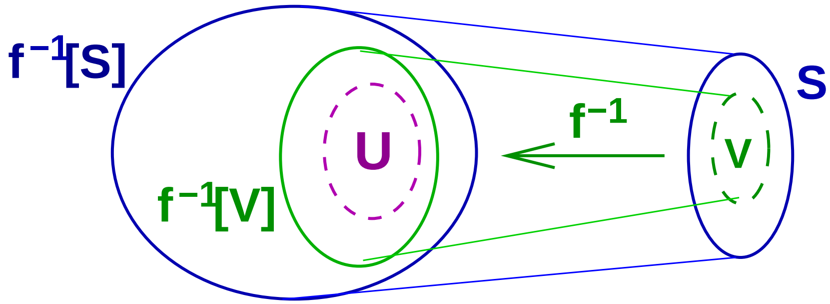
Fact: f^{-1} is $\mathcal{F}_\Sigma$-preserving iff f^{-1} is F-$\mathcal{F}_\Sigma$-complete iff f is $\sim\mathcal{F}_\Sigma$-reflecting iff f is $\mathcal{F}_\Sigma$-reflecting.

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

Definition: $f^{-1} : \mathcal{P}(\Sigma) \to \mathcal{P}(\Sigma)$ 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 $f^{-1}[V] \supseteq U$.

f^{-1} is dual continuous at $S \subseteq \Sigma$:



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

But I don't know for what this might be useful! (-:

The "topology" induced from an abstract interpretation is coarser than the Scott topology

Reconsider L^∞ and its approximant, L_k, which denotes a closed family.

- There is a Scott-continuous function, $f : L^\infty \rightarrow L^\infty$, that is not L_k-backwards complete for all $k > 0$. Define f as $f(d^k, nil) = nil$, for all $k \geq 0$, and $f(\ell) = \bot$, otherwise; this is Scott-continuous. Consider $f^{-1}\{nil\}$; it is all total, finite lists in L^∞, and for no finite $e \in L^\infty$ 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 $f_k : L^\infty \rightarrow L^\infty$ 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.

Concluding remarks

There is a lot of classical denotational semantics employed in abstract-interpretation theory and practice....

References **This talk:** `www.cis.ksu.edu/~schmidt/papers`

1. S. Abramsky. Domain theory in logical form. *Ann.Pure Appl.Logic* (51) 1991.
2. P. Cousot. Constructive design of a hierarchy of semantics of a transition system by abstract interpretation. *TCS* (277) 2002.
3. P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs. Proc. 4th. ACM Prin. Prog. Lang. 1977.
4. R. Giacobazzi and E. Quintarelli. Incompleteness, counterexamples, and refinements in abstract model checking. SAS'01, LNCS 2126.
5. T. Jensen. Abstract Interpretation in Logical Form. PhD Thesis, Imperial College, 1992.
6. D.A. Schmidt. Abstract interpretation from a topological perspective. Submitted for publication, 2009.
7. R.D. Tennent. The denotational semantics of programming languages. *Comm. ACM* (19) 1976.