- Hi everyone, (intro, say name, MoL year, what project)
- The first half of this talk is [redacted] going to be on introduction to programming semantics with domain theory.
- Informally, we assign meaning to our programming languages so that we can reason about them. Let's see an example, with a simple programming language

Logic in Programming semantics

1 Introduction

def The programming language Imp

$$C ::= \underline{\text{if}}\ B\ \underline{\text{then}}\ C_1\ \underline{\text{else}}\ C_2 \mid \underline{\text{def}}\ n := A \mid \underline{\text{while}}\ B\ \underline{\text{do}}\ C \mid C_1 ; C_2 \mid \underline{\text{skip}}$$

$$B ::= \underline{tt} \mid \underline{ff} \mid A_1 == A_2 \mid A_1 <= A_2 \mid B_1 \,\&\, B_2 \mid B_1 \,\|\, B_2 \mid \sim B$$

$$A ::= k \in \mathbb{Z} \mid n \in \mathbb{N} \mid A_1 - A_2 \mid A_1 + A_2 \mid A_1 * A_2$$

(n: variable)

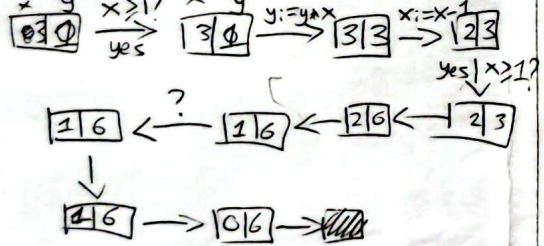
Variables will be natural numbers, for simplicity but in examples, we use x and y

You all know what boolean and arithmetic expressions look like so we focus on bigger examples

example

```
def y := 1;
while (x>1) do (
    def y := y * x;
    def x := x - 1;
)
```

Our idea of how the program runs:



If each command is a transformation of states, can we model them as functions?

the semantics of arithmetic expressions and boolean expressions are kind of what you would expect.

A first attempt at semantics

State: $\Sigma = \mathbb{N} \to \mathbb{Z}$ (maps variables to their values)

Semantics of AExp

$$[\![-]\!]_{(-)} : \Sigma \times \langle AExp \rangle \to \mathbb{Z}$$

$$[\![k]\!]_s = k \qquad [\![n]\!]_s = s(n) \qquad [\![A_1 + A_2]\!]_s = [\![A_1]\!]_s + [\![A_2]\!]_s$$

etc.

Exactly as you'd expect

Semantics of BExp

$$[\![-]\!]_{(-)} : \Sigma \times \langle BExp \rangle \to 2$$

$$[\![tt]\!] = tt \qquad [\![ff]\!] = ff \qquad [\![B_1 \,\&\, B_2]\!] = [\![B_1]\!] \wedge [\![B_2]\!]$$

Semantics of Commands

$$[\![-]\!] : \langle Command \rangle \to (\Sigma \to \Sigma)$$

$$[\![skip]\!](s) = s \quad \text{AKA} \quad [\![skip]\!] = id$$

$$[\![C_1 ; C_2]\!](s) = [\![C_2]\!]([\![C_1]\!](s)) \quad \text{AKA} \quad [\![C_2]\!] \circ [\![C_1]\!]$$

$$[\![\text{if } B \text{ then } C_1 \text{ else } C_2]\!](s) = \begin{cases} [\![C_1]\!](s) & [\![B]\!]_s = tt \\ [\![C_2]\!](s) & [\![B]\!]_s = ff \end{cases}$$

Now, a command transforms state, so it should be a function $\Sigma \to \Sigma$

Beware that it's flipped!

We will avoid plugging in parameters unless necessary.

And finally... while loops!

How do we model this mathematically?
Just repeat the thing.

$[\![\text{while } B \text{ do } C]\!](s) =$
1. check $[\![B]\!]_s$. If it's true, keep going to 2.
2. do $[\![C]\!](s)$.
3. and so on...

$$[\![\text{while } B \text{ do } C]\!](s) \begin{cases} [\![\text{while } B \text{ do } C]\!] \circ [\![C]\!](s) & [\![B]\!]_s = tt \\ s & [\![B]\!]_s = ff \end{cases}$$

Hold on... what about

while tt do skip?

$[\![\text{w } tt \text{ do skip}]\!](s)$

$$= \begin{cases} [\![\text{w } tt \text{ do skip}]\!]([\![\text{skip}]\!](s)) \\ s \end{cases}$$

Does Nothing

$= [\![\text{w } tt \text{ do skip}]\!](s)$

$= \ldots$!!

Actually the whole talk will be me writing this out. !!

Solve For $f = F_{B,C}(f)$:

Let $F_{B,C} : (\Sigma \to \Sigma) \to (\Sigma \to \Sigma)$

$$F_{B,C}(f)(s) \triangleq \begin{cases} f \circ [\![C]\!](s) & \text{if } [\![B]\!]_s = tt \\ s & [\![B]\!]_s = ff \end{cases}$$

$[\![tt]\!]_s = tt$
$[\![tt]\!]_s = ff$

Never happens

But Math has a way of dealing with this... Those are equations! for example: solve for $x = 2x - 1$

But there can be many solutions...
which one do we pick?

$\forall f \in \Sigma \to \Sigma,\ f = F_{tt,\text{skip}}(f)$

following our computational intuition, none of these!

Want: $\Sigma \to \Sigma$, not just $\Sigma \to \Sigma$.

$[\![C]\!](s)$ undefined $\iff$ C does not terminate when started at state s

Sometimes terminating Example

$[\![\text{while } x >= 0 \text{ do skip}]\!](s)$

$$\begin{cases} [\![\text{while } x >= 0 \text{ do skip}]\!](s) & \text{if } s(x) \geqslant 0 \\ s & \text{if } s(x) < 0 \end{cases}$$

Some observations:

$\forall$ solutions f, $s(x) < 0 \Rightarrow f(s) = s$

$s(x) \geqslant 0 \Rightarrow f(s)$ can be anything!

"Minimal" solution: $s \mapsto \begin{cases} \text{undefined} & \text{if } s(x) \geqslant 0 \\ s & \text{if } s(x) < 0 \end{cases}$

Formally, take g.l.b/intersection of all solutions under the ordering:

$f \sqsubseteq g$ iff $\forall s \in \Sigma.\ f(s) \downarrow \Rightarrow g(s) \downarrow$ and $= f(s)$

Because the equation is trivially solvable.

Because the equation forces this.

How to make this formal?

The Nice thing is that this corresponds to our computational intuition.

We want to denote while tt do skip as the never-term program, but all of these are terminating

we should allow partial functions $\Sigma \to \Sigma$,

Let's look at an example that is sometimes terminating, for more information.

Narration Start here (maybe talk while erasing?)

Taking the greatest lower bound is not a very computationally feasible way of obtaining the minimal solution. The key to solving this lies in our original naive definition: it works when at some point $[\![B]\!]_s$ becomes false, so we do not refer to the repeated $[\![\text{while } B \text{ do } C]\!]$.

1. Start with $\bot$ ($\bot(s)$ undefined $\forall s$).
2. Compute $F_{B,C}(\bot)$
   ↳ Defined $\forall s.\ [\![B]\!]_s = ff$
3. Compute $F^2_{B,C}(\bot)$
   ↳ Defined $\forall s'.\ [\![B]\!]_{[\![C]\!](s')} = ff$
   and $\forall s.\ [\![B]\!]_s = ff$
   ⋮ etc.
   ⋮

Look at it bottom-up instead: starting knowing nothing about $[\![\text{while } B \text{ do } C]\!]$, can we gather any info about $[\![\text{while } B \text{ do } C]\!]$?

Note: $\bot \sqsubseteq F_{B,C}(\bot) \sqsubseteq F^2_{B,C}(\bot) \sqsubseteq \dots$

each iteration gives us more and more info

At limit,

$$[\![\text{while } B \text{ do } C]\!] = \lim_{n\to\infty} F^n_{B,C}(\bot)$$

Formally: $\bigsqcup_{n\in\mathbb{N}} F^n_{B,C}(\bot)$

the limit must incorporate the information from each $F^n(\bot)$, and Nothing else.

The theme here is that $\sqsubseteq$ represents some kind of information ordering. Hence $\bot$ is the zero-information element. This shows Sets are not sufficient to interpret programs, we need more structure.

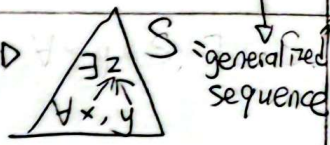
Let us now abstract from this specific scenario, in order to develop a general theory of denotational structures.

we want all limits to exist

Every sequence is directed

Directed-Complete Partial Order

2 DCPOs



S = generalized sequence

we want to start our computation of least fixpoint somewhere

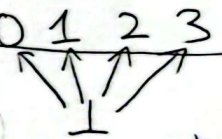
def $(D, \sqsubseteq)$ is DCPO if

- For every directed $S \subseteq D$, $\bigsqcup S$ exists.
- Least element $\bot$.

Examples

- $\Sigma \to \Sigma$. $\bigsqcup S = \bigcup$ of graphs of S
- $A \to B$, $A, B \in$ Set
- $A_\bot$, $A \in$ Set

$\mathbb{N}_\bot$:



limits trivially exist in posets of finite height

- NOT ω BUT $\omega+1$, or any successor ordinal
- Better $\Sigma = \mathbb{N} \to \mathbb{Z}$

Better because now we can build up infinite elements as limit of finite elements

e.g. $\bot \sqsubseteq (0 \mapsto 0) \sqsubseteq (0 \mapsto 0, 1 \mapsto -1) \sqsubseteq (0 \mapsto 0, 1 \mapsto -1, 2 \mapsto 2)$ ⋮

$$n \longmapsto \begin{cases} n & n \text{ even} \\ -n & n \text{ odd} \end{cases}$$

What about maps between DCPOs? They should capture the notion of computability, and not just be any function. A computable function can only operate on given information and how it behaves on more information should be consistent with its behavior at less.

def D, E DCPO, $f : D \to E$

1. f **monotone** if $x_1 \sqsubseteq_D x_2$ implies $f(x_1) \sqsubseteq_E f(x_2)$.
2. f **Scott-continuous** if $f(\bigsqcup S) = \bigsqcup f[S]$, and f monotone.

Also, it should not invent new information in the limit, since it only works with what is given.

example $\Sigma \to \Sigma$

- continuous: $s \longmapsto \left(n \longmapsto \begin{cases} n & n \text{ even} \\ s(n) & n \text{ odd} \end{cases} \right)$ (not computable! at least using Imp)
- NOT continuous: $s \longmapsto \begin{cases} s & s \text{ finite} \\ n \mapsto s(n)+10 & s \text{ infinite} \end{cases}$

Rule of Thumb for Σ: don't talk about undefinedness

theorem Let $f : D \to D$ continuous.

(1) $f\left(\bigsqcup_{n \in \mathbb{N}} f^n(\bot)\right) = \bigsqcup_{n \in \mathbb{N}} f^n(\bot) = \bigsqcup_{n \in \mathbb{N}} f^{n+1}(\bot)$

$\bot \sqsubseteq x \Rightarrow f^n(\bot) \sqsubseteq f^n(x) = x$

(2) $\forall x. f(x) = x \implies \bigsqcup_{n \in \mathbb{N}} f^n(\bot) \sqsubseteq x$

We are now in the position to show directed limits actually solve our problem, and is the minimal solution.

Σ is generated by the finite elements.

This is a good property computationally, since it means we can work with just the finite elements. and then computational behavior on the infinite elements is immediately determined. But first how to express finiteness?

3 Algebraic DCPOs

def Let D DCPO. $x \in D$ is **compact** iff $x \sqsubseteq \bigsqcup S \implies \exists y \in S.\ x \sqsubseteq y$

(well, if x is finite, and is surpassed by infinite limit, then it is surpassed at a finite point.)

example

- $\bot$ is compact
- Every element in "finite-height" DCPO
- Finite elements of Σ
- $2\omega + 1 \in 2\omega + 2$

(This last example shows compactness is not quite finiteness but close enough.)

Notation KD set of compact elements,

$\downarrow_K x$ set of compact elements below x

def D DCPO, is **algebraic** iff

$\forall x \in D.\ \downarrow_K x$ is directed and $x = \bigsqcup \downarrow_K x$

$x \sqsubseteq y$ iff $\downarrow_K x \subseteq \downarrow_K y$.

Specifically: $\downarrow_K x \subseteq \downarrow_K y \Rightarrow x = \bigsqcup \downarrow_K x \sqsubseteq \bigsqcup \downarrow_K y = y$

we can now define what it means to be generated by finite/compact elements.

Here's some evidence for this fact. clearly, if x is below y, then the stuff below x is also below y. For the other direction,

So since elements of D are directed joins of elements of KD, we should be able to recover D as the poset of directed subsets of KD. This is almost true, except an element of D can be obtained from multiple directed joins, so we need to choose a canonical representative. For this, we can take the maximal ones, which are the downwards closed ones.

def Let P Poset. An ideal I is a downwards closed, directed subset of P.

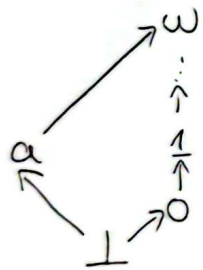
$$x \sqsubseteq y,\ y \in I \Rightarrow x \in I$$

Let $Id(P)$ denote the poset of ideals of P, ordered by subset inclusion ($\subseteq$).

theorem P poset $\Rightarrow Id(P)$ Alg DCPO, with $K(Id(P)) \cong P$

D Alg DCPO $\Rightarrow D \cong Id(KD)$

non-example



$\downarrow_K \omega$ NOT directed

I'd rather give a non-example, since naturally occuring DCPOs tend to be algebraic already.

An important construction in domain theory is the function space. We can construct our $\mathcal{E}$ as a function space, and of course we interpret commands as continuous functions $\mathcal{E} \to \mathcal{E}$.

4 Function Spaces

def Let D, E DCPO.

$$[D \to E] = \{ f : D \to E \mid f \text{ Scott continuous} \}$$

$$f \sqsubseteq_{[D \to E]} g \text{ iff } \forall x \in D.\ f(x) \sqsubseteq_E g(x)$$

Strict version:

$$[D \to_\perp E] = \{ f : [D \to E] \mid f(\perp) = \perp \}$$

The idea here is that some programming languages are strict: if the input is non-terminating then so is the output since it computes the input first. e.g. Java, Python, NOT Haskell.

example

- Instead of $\mathcal{E} = \mathbb{N} \to \mathbb{Z}$, $\mathcal{E} = [\mathbb{N}_\perp \to_\perp \mathbb{Z}_\perp]$
- $[\![-]\!] : \langle \text{Command} \rangle \to [\mathcal{E} \to \mathcal{E}]$

Not strict, want $[\![\text{defn} := A]\!](\perp) = (n \mapsto [\![A]\!]_\perp)$,

theorem?? D, E Algebraic $\Rightarrow [D \to E]$ Algebraic

we would like to maintain algebraicity of the function spaces, but is this true? Let us first analyse the compact elements of the function space

Not strict because we want meaningful computation on partial elements

Taking inspiration from Σ, the compact elements should be generated by some kind of input output pairs. We have that as well for general function spaces, but monotonicity forces us to define a little bit extra.

def Let $D, E \in \mathrm{Alg}\mathbf{DCPO}$, $d \in \mathcal{K}D$, $e \in \mathcal{K}E$.
the **compact pair function** $\langle d; e\rangle : [D \to E]$

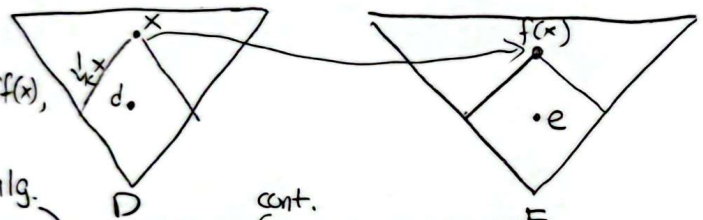
$$\langle d; e\rangle(x) = \begin{cases} e & \text{if } x \sqsupseteq d \\ \bot & \text{otherwise} \end{cases}$$

Note $\langle d; e\rangle$ not even continuous unless d compact.

Indeed, any continuous function can be generated as a join of pair functions

prop $f : D \to E$ continuous $\Rightarrow$ LUB of compact pair function.

Proof



$\forall x, e \sqsubseteq f(x)$,

$$e \sqsubseteq f(x) \overset{\text{alg.}}{=} f(\bigsqcup \downarrow_K x) \overset{\text{cont.}}{=} \bigsqcup \{ f(d) \mid d \in \downarrow_K x \}$$

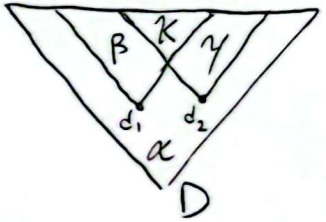
$\Rightarrow \exists d \in \downarrow_K x.\ e \sqsubseteq f(d)$ So $\langle d; e\rangle \sqsubseteq f$. $\square$

However, this join is not necessarily directed. The argument goes as follows:

BUT: This LUB is not (necessarily) directed.

Let $\langle d_1; e_1\rangle, \langle d_2; e_2\rangle \sqsubseteq f$. Need g s.t.
① $g \in \downarrow_K f$ ② $\langle d_1; e_1\rangle, \langle d_2; e_2\rangle \sqsubseteq g$

$$g(x) = \begin{cases} \bot & x \in \alpha \\ e_1 & x \in \beta \\ e_2 & x \in \gamma \\ ??? & x \in k \end{cases}$$

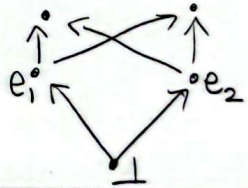


Does not exist!

① $e \in \downarrow_K f(x)$ ② $e_1, e_2 \sqsubseteq e$

Obvious candidate: ~~$e_1 \sqcup e_2$~~

example



So, $[D \to E]$ is not necessarily algebraic. However, it's not conceptually hard to accept that if two elements are consistent with each other, i.e. have an upper bound, then they should have a least upper bound, thus ruling out this example. Okay, this is more of a "why not" than a "why".

Based on this "why not" reasoning let us define the extra condition anyway. We call the DCPOs with this condition Scott Domains.

That's sort of concludes the first half of the talk. Before we end, let's look at the domain-ified version of the semantics of IMP.

(Show on laptop)

Then, it is easy to show Scott domains are closed under function spaces.

def E Scott Domain if

- E algebraic DCPO
- $\forall e_1, e_2 \in E.\ e_1 \triangle e_2 \Rightarrow \exists e_1 \sqcup e_2$
  $\hookrightarrow \exists e.\ e_1 \sqsubseteq e \sqsupseteq e_2$

Theorem If D alg. DCPO and E Scott, then [D → E] Scott.

Prop $K[D \to E] = \left\{ \bigsqcup_{i \in I} \langle d_i ; e_i \rangle \;\middle|\; d_i, e_i \text{ compact}, \triangle_{i \in I} \langle d_i ; e_i \rangle \right\}$

($\bigsqcup_{i \in I}$: finite; $\triangle_{i \in I} \langle d_i ; e_i \rangle$: $\exists g.\ \forall i.\ \langle d_i ; e_i \rangle \sqsubseteq g$)

Using this, we can now also characterize the compact elements of [D → E]

Proof:

$f \in K[D \to E]$ is join of pair func.

Scottiness $\Rightarrow$ is directed join of finite join of pair func.

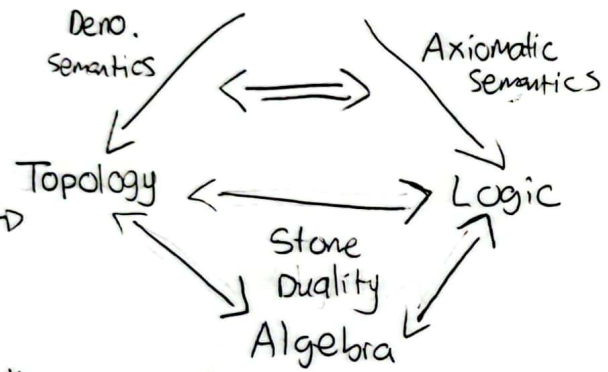
compactness $\Rightarrow$ f = finite join of pair func. □

Please take note of this result! We will use it quite a bit in the second half of the talk.

Welcome Back from the break!

In the first half, we defined a semantics for our programming language IMP, and so now we can reason about IMP in terms of it's semantics. But, what if we wanted to ~~reason~~ formalize this reasoning? In this second half, we will see that domains are really topological structures, and that there is a framework called Stone duality that establishes a correspondance between logics and their Semantic spaces, allowing us to derive formal logics of programs.

Programs



Draw this first!

usually: Logic ⟹ Algebra ⟹ Topology

Today: Topology ⟹ Algebra ⟹ Logic

In Logic applications, the path we take is usually Logic ⇓ Algebra ⇓ Top.

We will take the reverse path today, since we know the spaces we are interested in, but not their logic.

We can think of a space X as a set of things, and with a set of finitely observable properties.

1 Some Preliminaries

def A topological space X:

- a set $|X|$
- set of subsets $\mathcal{O}X \subseteq \mathcal{P}X$
- $\mathcal{U} \in \mathcal{O}X$ is called open set

satisfying:

- $\mathcal{U}, \mathcal{V}$ open $\Rightarrow \mathcal{U} \cap \mathcal{V}$ open — finitely observe both. Note: can't finitely observe infinitely many
- $\forall i \in I.\ \mathcal{U}_i$ open $\Rightarrow \bigcup_{i \in I} \mathcal{U}_i$ open — finitely observe one.
- $\emptyset, |X|$ open — trivial to observe: either never or always.

def A category C:

- ~~obj~~ class of objects $|C|$
- $\forall c_1, c_2 \in C$, set $\mathrm{Hom}_C(c_1, c_2)$ of morphisms closed under composition and identity.

the intuition here is that if I observe property $\mathcal{U}$ of some element y, I want to observe a corresponding property on the inverse image of y.

We now see that the Scott topology can replace the ordering, since we can recover it from the topology. The idea is that x has less information than y iff any property observed on x is also observed on y.

examples

- Top : Objects are topological spaces, morphisms are continuous functions: $f : X \to Y$

  $\mathcal{U}$ open in $Y \Rightarrow f^{-1}[\mathcal{U}]$ open in X

- DCPO : morphisms are Scott-continuous functions.
- Alg DCPO : full subcategory of algebraic DCPO
- SDom : —— " —— Scott Domains

Really, continuous maps are maps between properties.

2 Scott Topology

def Let D DCPO. $\mathcal{U} \subseteq D$ is open in the Scott Topology if

- $\mathcal{U}$ is upwards closed: $x \in \mathcal{U} \;\&\; x \sqsubseteq y \Rightarrow y \in \mathcal{U}$.
- $\mathcal{U}$ is inaccessible by directed joins:

$$\bigsqcup S \in \mathcal{U} \Rightarrow \exists x \in S.\ x \in \mathcal{U}.$$

Denote the topology by ΣD or ΩD.

prop Let D be DCPO. Then

$$x \sqsubseteq_D y \text{ iff } \left(\forall \mathcal{U} \text{ Scott open}, x \in \mathcal{U} \Rightarrow y \in \mathcal{U}\right)$$

proof

($\Rightarrow$) obvious. (since this is just a reshuffling of saying Scott open sets are upwards closed.)

($\Leftarrow$) Take $\mathcal{U}_y = \{z \in D \mid z \not\sqsupseteq y\}$. (the set of things not below y.)

$y \notin \mathcal{U}_y \Rightarrow x \notin \mathcal{U}_y \iff x \sqsubseteq y$. □

prop Let $f : D \to E$, D, E DCPO

f Scott-continuous iff $f : (D, \Sigma D) \to (E, \Sigma E)$ continuous.

(To further justify this topology, I note that the topologically continuous functions are exactly the Scott continuous ones.)

3 Stone Duality.

$$(|X|, \Omega X) \longmapsto \Omega X$$

$$\text{Pt } A \longleftarrow A$$

Topology — Algebra

For DCPOs, the notion of "finitely observable" is something we can express very nicely: the idea being that if y has more information than x, any property observed of x is also observed of y. (if $\mathcal{U}$ observable at some infinite limit, it can already be observed at a finite point)

In it's most basic form, Stone Duality is asking:

"How much topology can we do using only the open sets, seen as an algebraic structure?" i.e. given just ΩX, can we recover the points of X?

One attempt to reconstruct the points is by considering a set of properties that seem to completely characterize a single point. Before we do this, let's axiomatise the algebraic structure of the open sets

Now, we have an "obvious" topological structure on $pt A$. The extent to which we can do topology using only the open sets ΩX answered by comparing $pt A$ with X.

def A **locale** is a (distributive) lattice with infinitary joins:

$$(A, \leq, \bigvee, \wedge, 0, 1) \qquad \mathbf{Loc}$$

We have a category of locales where maps $f: B \to A$ are frame homomorphisms from A to B.

note the flip! we do this

Example $(\Omega X, \subseteq, \bigcup, \cap, \emptyset, |X|)$

If $f: X \to Y$ is continuous, $f^{-1}[-] : \Omega Y \to \Omega X$ is a frame homomorphism, so a locale map ΩX to ΩY.

def let A locate. $F \subseteq A$ is a (proper) completely prime filter or abstract point

- $1 \in F$, $0 \notin F$ $\quad (x \in |X|, x \notin \emptyset)$
- F upwards closed. $\quad (U \subseteq V, x \in U \Rightarrow x \in V)$
- $a, b \in F \Rightarrow a \wedge b \in F$ $\quad (x \in U, x \in V \Rightarrow x \in U \cap V)$
- $\bigvee_{i \in I} a_i \in F \Rightarrow \exists a_i \in F.$ $\quad \left(x \in \bigcup_{i \in I} U_i \Rightarrow x \in U_i\right)$

so now we can define more generally subsets of a frame that seem to determine a point.

Denote $pt A = \{\text{abstract points of } A\}$

"Obvious" topology on $pt A$:

$\forall a \in A. \ \{F \in pt A \mid a \in F\}$ open.

def Let $X \in \mathbf{Top}$. If the map

$$X \longrightarrow pt\,\Omega X, \quad x \longmapsto \{U \in \Omega X \mid x \in U\}$$

is a homeomorphism, we say X is sober

i.e. you don't hallucinate or forget points.

Analogously, A is spatial if $A \cong \Omega\, pt A$.

theorem (Stone Duality) Equivalence of Categories

The sober spaces are the ones on which the points can be forgotten, i.e. point-free topology.

$$\begin{array}{ccc} \mathbf{Sob} & & \mathbf{SLoc} \\ X & \longmapsto & \Omega X \\ pt A & \longleftarrow & A \end{array}$$

The next question to ask is: we have some spaces already, i.e. the Scott topology on domains. Are they sober? And if yes, can we characterise them algebraically/logically?

Let us now tackle the first question:

4 Stone Duality for Scott Domains

Q1. Is Scott Sober?

In general No! (He's hungover)

Yes, for Algebraic DCPOs.

Compact elements = Anchors to reality.

Prop Let $D \in Alg\,DCPO$. Then open sets of form $\uparrow c \quad \forall c \in KD$ generate all open sets.

proof $U = \bigcup \{\uparrow c \mid c \in U \cap KD\}$. □

Theorem $D \in Alg\,DCPO$ is sober.

proof sketch By above prop, $F \in pt\Sigma D$ determined by which $\uparrow c$ it contains. So,

$$F \Rightarrow K_F = \{c \in KD \mid \uparrow c \in F\} \Rightarrow \bigsqcup K_F$$

claim: this is directed ↗ □

Moreover, we can characterize the compact elements as the completely prime opens.

def Let $A \in Loc$. $a \in A$ is **completely prime** if

$$a \leq \bigvee_{i \in I} b_i \Rightarrow \exists b_i \geq a.$$

Note the similarity to compactness indeed, it is like a super compactness

prop $a \in A$ is completely prime iff a is compact:

$$a \leq \bigvee^{\uparrow}_{i \in I} b_i \Rightarrow \exists b_i \geq a.$$

and prime:

$$a \leq b_1 \vee b_2 \Rightarrow a \leq b_1 \text{ or } a \leq b_2.$$

Now,

$$\uparrow c \subseteq \bigcup_{i \in I} U_i \Rightarrow c \in \bigcup_{i \in I} U_i \Rightarrow c \in U_i \Rightarrow \uparrow c \subseteq U_i.$$

On the other hand, completely prime $U \in \Sigma D \Rightarrow \uparrow U$ abstract point, compact by similar reasoning.

Specific to Scott Domain property:

CP-'opens' closed under $\cap$, since

$$\uparrow c_1 \cap \uparrow c_2 = \begin{cases} \uparrow (c_1 \sqcup c_2) & \text{if } c_1 \triangle c_2 \\ \varnothing & \text{otherwise} \end{cases}$$

Moreover, from this we can sort of get a finitary algebra/logic. This property is called being spectral, where the finitary/compact opens generate the opens.

def Let X be topological space.
X is spectral if

1. X is sober
2. $K\mathcal{O}X$ generates all open sets.
3. $K\mathcal{O}X$ closed under finite intersection. (in particular, $\emptyset \in K\mathcal{O}X$)

making $K\mathcal{O}X$ a distributive lattice.

follows from CP-opens generating all opens, since CP-opens are compact.

theorem Stone Duality restricts to an equivalence between Scott domains and distributive lattices satisfying

1. Every element is a finite join of prime elements.
2. prime elements are closed under meets.

Using this characterization, we can build a logic for reasoning about Imp! To make it fast, I will just share the end result, to make things fast.

Logic In Programming Language Semantics

By Alyssa Renata

$$[\![-]\!]_{(-)} : \Sigma \times \langle AExp \rangle \to \mathbb{Z}_\perp$$

$$\begin{aligned}
[\![k \in \mathbb{Z}]\!]_s &= k \\
[\![n \in \mathbb{N}]\!]_s &= s(n) \\
[\![A_1 \texttt{ - } A_2]\!]_s &= [\![A_1]\!]_s \,[\![\texttt{-}]\!]\, [\![A_2]\!]_s \\
[\![A_1 \texttt{ + } A_2]\!]_s &= [\![A_1]\!]_s \,[\![\texttt{+}]\!]\, [\![A_2]\!]_s \\
[\![A_1 \texttt{ * } A_2]\!]_s &= [\![A_1]\!]_s \,[\![\texttt{*}]\!]\, [\![A_2]\!]_s
\end{aligned}$$

$$[\![-]\!]_{(-)} : \Sigma \times \langle BExp \rangle \to 2_\perp$$

$$\begin{aligned}
[\![\texttt{tt}]\!]_s &= tt \\
[\![\texttt{ff}]\!]_s &= ff \\
[\![B_1 \texttt{ and } B_2]\!]_s &= [\![B_1]\!]_s \,[\![\texttt{and}]\!]\, [\![B_2]\!]_s \\
[\![B_1 \texttt{ or } B_2]\!]_s &= [\![B_1]\!]_s \,[\![\texttt{or}]\!]\, [\![B_2]\!]_s \\
[\![\texttt{not } B]\!]_s &= [\![\texttt{not}]\!]\, [\![B]\!]_s \\
[\![A_1 \texttt{ = } A_2]\!]_s &= ([\![A_1]\!]_s \,[\![\texttt{=}]\!]\, [\![A_2]\!]_s) \\
[\![A_1 \texttt{ <= } A_2]\!]_s &= ([\![A_1]\!]_s \,[\![\texttt{<=}]\!]\, [\![A_2]\!]_s)
\end{aligned}$$

$$\begin{aligned}
\perp \,[\![\texttt{and}]\!]\, b &= \perp \\
b \,[\![\texttt{and}]\!]\, \perp &= \perp \\
ff \,[\![\texttt{and}]\!]\, ff &= ff \\
ff \,[\![\texttt{and}]\!]\, tt &= ff \\
tt \,[\![\texttt{and}]\!]\, ff &= ff \\
tt \,[\![\texttt{and}]\!]\, tt &= tt
\end{aligned}$$

other operators defined similarly.

$$[\![-]\!] : \langle Command \rangle \to [\Sigma \to \Sigma]$$

$$\begin{aligned}
[\![\texttt{skip}]\!] &= id_\Sigma \\
[\![C_1 \texttt{;} C_2]\!] &= [\![C_2]\!] \circ [\![C_1]\!]
\end{aligned}$$

$$[\![\texttt{def } n \texttt{ := } A]\!](s) = \begin{cases} \perp_\Sigma & \text{if } [\![A]\!]_s = \perp_{\mathbb{N}_\perp} \\ s[n := [\![A]\!]_s] & \text{otherwise} \end{cases}$$

$$[\![\texttt{if } B \texttt{ then } C_1 \texttt{ else } C_2]\!](s) = \begin{cases} \perp & \text{if } [\![B]\!]_s = \perp_{2_\perp} \\ C_1(s) & \text{if } [\![B]\!]_s = tt \\ C_2(s) & \text{if } [\![B]\!]_s = ff \end{cases}$$

$$[\![\texttt{while } B \texttt{ do } C]\!] = \bigsqcup_{n \in \omega} F^n_{B,C}(\perp)$$

where $s[n := k]$ denotes the map[15]

$$x \mapsto \begin{cases} \perp & \text{if } x = \perp \\ k & \text{if } x = n \\ s(x) & \text{otherwise} \end{cases}$$

morally, should be

$$x \mapsto \begin{cases} k & \text{if } x = n \\ s(x) & \text{otherwise} \end{cases}$$

and $F_{B,C} : [\Sigma \to \Sigma] \to [\Sigma \to \Sigma]$ is the continuous function defined by

$$F_{B,C}(f) = s \mapsto \begin{cases} \perp & \text{if } [\![B]\!]_s = \perp_{2_\perp} \\ f \circ [\![C]\!](s) & \text{if } [\![B]\!]_s = tt \\ s & \text{if } [\![B]\!]_s = ff \end{cases}$$

Definition 4.2.2. Let $L(\Sigma)$ be the domain prelocale generated by generators of the form

$$\{(x \nrightarrow k) \mid x \in \mathbb{N}_\perp, k \in \mathbb{Z}\}.$$

sorry, these should be $\rightsquigarrow$.

satisfying additional rules:

$$\frac{x \sqsubseteq x'}{(x \to k) \leq (x' \to k)}\ \Sigma\text{-}\leq$$

$$\frac{k_1 \neq k_2}{(\mathbb{N}_\perp \rightsquigarrow k_1) \wedge (x \rightsquigarrow k_2) = 0}\ \Sigma\text{-}\mathbb{N}_\perp \qquad \frac{k_1 \neq k_2 \quad n \in \mathbb{N}}{(n \rightsquigarrow k_1) \wedge (n \rightsquigarrow k_2) = 0}\ \Sigma\text{-}n$$

$$\frac{}{\mathsf{P}(\mathbb{N}_\perp \rightsquigarrow k)}\ \Sigma\text{-P-}\rightsquigarrow \qquad \frac{\forall i, j \in I. n_i \neq n_j}{\mathsf{P}(\bigwedge_{i \in I}(n_i \rightsquigarrow k_i))}\ \Sigma\text{-P-}\wedge$$

By the earlier characterisation of K[D->E], and the prime open-to-compact-element correspondence, we know the prime opens are of the above form.

Rules for Commands

$$\frac{}{[\phi]\ \texttt{skip}\ [\phi]}\ \text{H-skip} \qquad \frac{[\phi]\ C_1\ [\theta] \qquad [\theta]\ C_2\ [\psi]}{[\phi]\ C_1 ; C_2\ [\psi]}\ \text{H-;}$$

$$\frac{[\phi]\ B\ [\texttt{tt}] \qquad [\phi]\ C_1\ [\psi]}{[\phi]\ \texttt{if}\ B\ \texttt{then}\ C_1\ \texttt{else}\ C_2\ [\psi]}\ \text{H-ite-tt} \qquad \frac{[\phi]\ B\ [\texttt{ff}] \qquad [\phi]\ C_2\ [\psi]}{[\phi]\ \texttt{if}\ B\ \texttt{then}\ C_1\ \texttt{else}\ C_2\ [\psi]}\ \text{H-ite-ff}$$

$$\frac{[\phi]\ B\ [\texttt{tt}] \qquad [\phi]\ C\ [\theta] \qquad [\theta]\ \texttt{while}\ B\ \texttt{do}\ C\ [\psi]}{[\phi]\ \texttt{while}\ B\ \texttt{do}\ C\ [\psi]}\ \text{H-while-tt} \qquad \frac{[\phi]\ B\ [\texttt{ff}]}{[\phi]\ \texttt{while}\ B\ \texttt{do}\ C\ [\phi]}\ \text{H-while-ff}$$

$$\frac{[\phi]\ A\ [\alpha] \qquad \mathsf{T}(\alpha)}{[\phi]\ \texttt{def}\ n\ \texttt{:=}\ A\ [\bigwedge_{k\in\alpha}(n \rightsquigarrow k)]}\ \text{H-def} \qquad \frac{[\mathbb{N}_\perp \rightsquigarrow k]\ A\ [k]}{[\mathbb{N}_\perp \rightsquigarrow k]\ \texttt{def}\ n\ \texttt{:=}\ A\ [\mathbb{N}_\perp \rightsquigarrow k]}\ \text{H-def-}\mathbb{N}_\perp$$

$$\frac{n_i \neq n}{[\bigwedge_{i\in I}(n_i \rightsquigarrow k_i)]\ \texttt{def}\ n\ \texttt{:=}\ A\ [n_i \rightsquigarrow k_i]}\ \text{H-def-}\wedge \qquad \frac{n' \neq n}{[\mathbb{N}_\perp \rightsquigarrow k]\ \texttt{def}\ n\ \texttt{:=}\ A\ [n' \rightsquigarrow k]}\ \text{H-def-}\mathbb{N}_\perp{}'$$

Generic Rules

$$\frac{}{[\Phi]\ M\ [1]}\ \text{H-1} \qquad \frac{}{[0]\ M\ [\Psi]}\ \text{H-0}$$

$$\frac{[\Phi]\ M\ [\Psi] \qquad [\Phi]\ M\ [\Theta]}{[\Phi]\ M\ [\Psi \wedge \Theta]}\ \text{H-}\wedge \qquad \frac{[\Phi]\ M\ [\Psi] \qquad [\Theta]\ M\ [\Psi]}{[\Phi \vee \Theta]\ M\ [\Psi]}\ \text{H-}\vee \qquad \frac{\Phi \leq \Phi' \qquad [\Phi']\ M\ [\Psi'] \qquad \Psi' \leq \Psi}{[\Phi]\ M\ [\Psi]}\ \text{H-}\leq$$

Generic Rules

$$\frac{}{[\Phi]\ M\ [1]}\ \text{H-1} \qquad \frac{}{[0]\ M\ [\Psi]}\ \text{H-0}$$

$$\frac{[\Phi]\ M\ [\Psi] \qquad [\Phi]\ M\ [\Theta]}{[\Phi]\ M\ [\Psi \wedge \Theta]}\ \text{H-}\wedge \qquad \frac{[\Phi]\ M\ [\Psi] \qquad [\Theta]\ M\ [\Psi]}{[\Phi \vee \Theta]\ M\ [\Psi]}\ \text{H-}\vee \qquad \frac{\Phi \leq \Phi' \qquad [\Phi']\ M\ [\Psi'] \qquad \Psi' \leq \Psi}{[\Phi]\ M\ [\Psi]}\ \text{H-}\leq$$

Rules for Arithmetic Expressions

$$\frac{}{[\phi]\ k \in \mathbb{Z}\ [k]}\ \text{H-}\mathbb{Z} \qquad \frac{}{[\bigwedge_{i \in I}(n_i \rightsquigarrow k_i)]\ n_i\ [k_i]}\ \text{H-var-}\bigwedge \qquad \frac{}{[\mathbb{N}_\perp \rightsquigarrow k]\ n_i\ [k]}\ \text{H-var-}\perp$$

$$\frac{[\phi]\ A_1\ [\alpha] \qquad [\phi]\ A_2\ [\beta]}{[\phi]\ A_1\ \texttt{-}\ A_2\ [\alpha\ \texttt{-}\ \beta]}\ \text{H-}\texttt{-} \qquad \frac{[\phi]\ A_1\ [\alpha] \qquad [\phi]\ A_2\ [\beta]}{[\phi]\ A_1\ \texttt{+}\ A_2\ [\alpha\ \texttt{+}\ \beta]}\ \text{H-}\texttt{+} \qquad \frac{[\phi]\ A_1\ [\alpha] \qquad [\phi]\ A_2\ [\beta]}{[\phi]\ A_1\ *\ A_2\ [\alpha\ *\ \beta]}\ \text{H-}*$$

Rules for Boolean Expressions

$$\frac{}{[\phi]\ \texttt{tt}\ [\texttt{tt}]}\ \text{H-}\texttt{tt} \qquad \frac{}{[\phi]\ \texttt{ff}\ [\texttt{ff}]}\ \text{H-}\texttt{ff}$$

$$\frac{[\phi]\ B\ [\chi]}{[\phi]\ \texttt{not}\ B\ [\texttt{not}\ \chi]}\ \text{H-}\texttt{not} \qquad \frac{[\phi]\ B_1\ [\chi] \qquad [\phi]\ B_2\ [\kappa]}{[\phi]\ B_1\ \texttt{and}\ B_2\ [\chi\ \texttt{and}\ \kappa]}\ \text{H-}\texttt{and} \qquad \frac{[\phi]\ B_1\ [\chi] \qquad [\phi]\ B_2\ [\kappa]}{[\phi]\ B_1\ \texttt{or}\ B_2\ [\chi\ \texttt{or}\ \kappa]}\ \text{H-}\texttt{or}$$

$$\frac{[\phi]\ A_1\ [\alpha] \qquad [\phi]\ A_2\ [\beta]}{[\phi]\ A_1\ \texttt{=}\ A_2\ [\alpha\ \texttt{=}\ \beta]}\ \text{H-}\texttt{=} \qquad \frac{[\phi]\ A_1\ [\alpha] \qquad [\phi]\ A_2\ [\beta]}{[\phi]\ A_1\ \texttt{<=}\ A_2\ [\alpha\ \texttt{<=}\ \beta]}\ \text{H-}\texttt{<=}$$