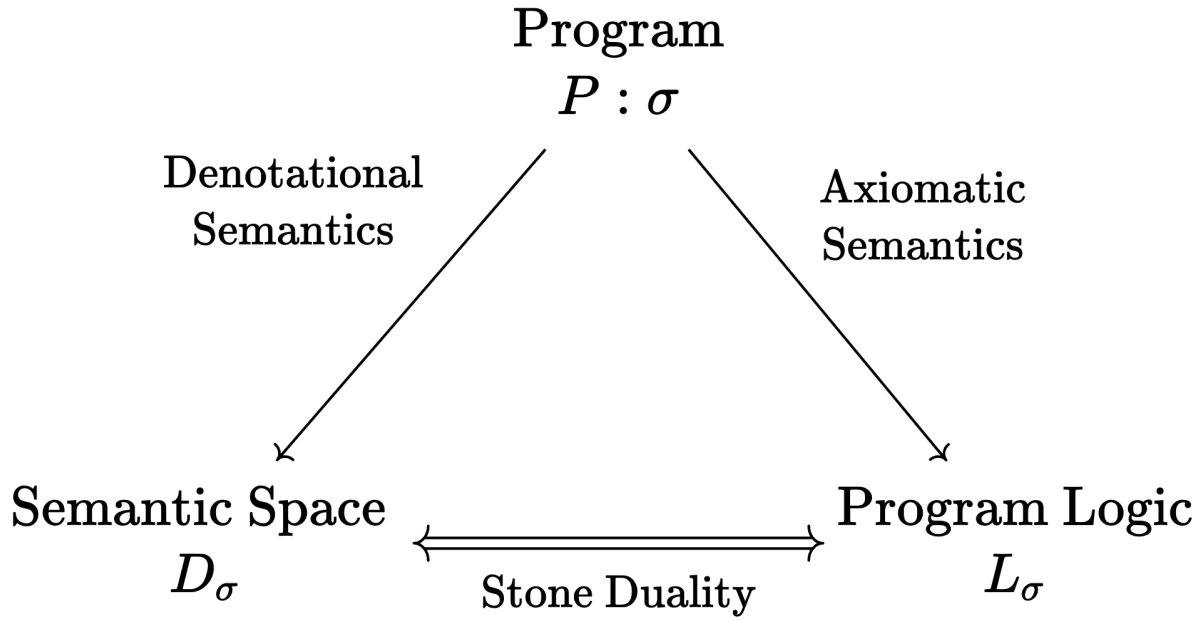
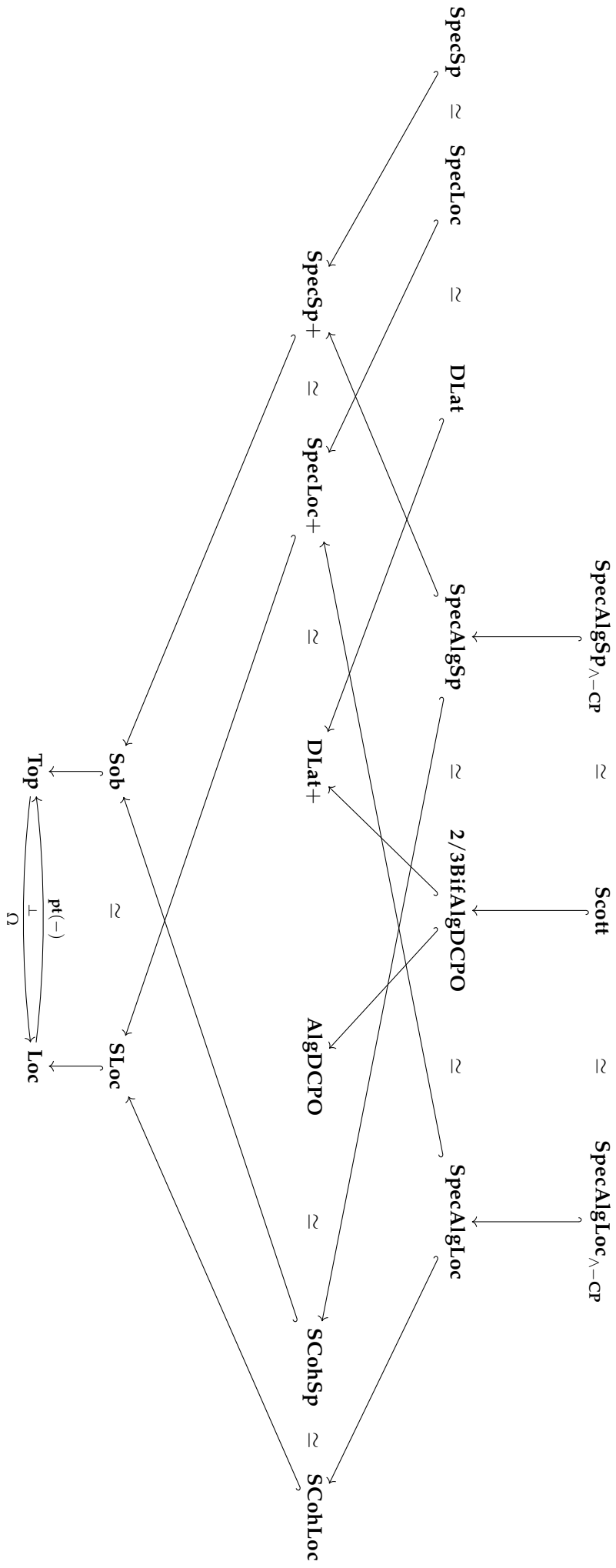
ALYSSA RENATA

DUALITY IN DOMAIN THEORY



Contents


SpecSp
SpecLoc
SpecSp+
DLat
SpecLoc+
SpecAlgSp
SpecAlgSp>−CP
DLat+
2/3BifAlgDCPO
Scott
Sob
Top
Ω
pt(−)
⊣
SLoc
Loc
AlgDCPO
SpecAlgLoc
SpecAlgLoc>−CP
SCohSp
SCohLoc
≅

1
Introduction

We give meaning to our programming languages so that we can reason about the correctness of the programs we write. This happens at an informal level: we intuitively understand an assignment statement $x := 2$ to modify some variable store x such that it has value 2, and may then reason that subsequent uses of x are synonymous with 2 until we re-assign its value.

In the study of programming language semantics, we take this process a step further by assigning a *formal* meaning to each well-formed program in the language, so that we may reason *mathematically* about programs. This is called denotational semantics. Following our intuition for now, a program represents a transformation on the machine state, so we can represent programs as functions between representations of the machine state (i.e. the machine's memory). As we will soon see however, we encounter some deep problems when trying to do this in a naive way using sets.

This failure motivates the need for domain theory, which, broadly construed, is the study of mathematical structures capable of modelling computational processes. Note that while domain theory studies the mathematical theory of such structures in and of themselves, denotational semantics refers to the application of such structures for modelling. The two fields are deeply connected[1], and heavily influence one another.

[1] they're invented by two people working together: Dana Scott & Christopher Strachey. The former developed the domain theoretic side while the latter developed the denotational side.

1.1 The Simple Imperative Programming Language **Imp**

Consider the following imperative language. It contains the standard constructions one would expect from an imperative language, including a while loop command. Unlike real languages, we only allow the assignment of numbers to variables, to keep things simple.

⟨*Command*⟩ ::= `if` ⟨*BExp*⟩ `then` ⟨*Command*⟩ `else` ⟨*Command*⟩
 | `while` ⟨*BExp*⟩ `do` ⟨*Command*⟩ | `def` ⟨*Var*⟩ `:=` ⟨*AExp*⟩
 | ⟨*Command*⟩ `;` ⟨*Command*⟩ | `skip`

⟨*BExp*⟩ ::= `tt` | `ff` | ⟨*AExp*⟩ = ⟨*AExp*⟩ | ⟨*AExp*⟩ `<=` ⟨*AExp*⟩
 | ⟨*Bool*⟩ `and` ⟨*Bool*⟩ | ⟨*Bool*⟩ `or` ⟨*Bool*⟩ | `not` ⟨*Bool*⟩

$\langle AExp \rangle ::= n \in \mathbb{Z} \mid \langle Var \rangle \mid \langle AExp \rangle$ - $\langle AExp \rangle \mid \langle AExp \rangle$ + $\langle AExp \rangle \mid \langle AExp \rangle * \langle AExp \rangle$

Naturally, we expect the arithmetic expressions to denote integers, and the boolean expressions to denote one of the truth values $2 = \{tt, ff\}$. The operations forming our expressions above should have their expected denotation: the + means addition and so forth. The only question now is, what should the variables denote in an arithmetic expression? Clearly, the intention is that the programmer is allowed to set the value of a variable to a number using the assignment command. Therefore, our machine state has to keep track of this information, which we then use to figure out what the variables should denote. Since this is the only aspect of the machine state accessible to the programmer, we can simply *model* states as functions

$$\Sigma := \langle Var \rangle \to \mathbb{Z}$$

assigning an integer to each variable.

Then, given a state $s \in \Sigma$, we can give a denotation for each such expression, as shown on the right. In general, we write $[\![E]\!]$ for the denotation of an expression E.

$[\![-]\!]_s : \langle BExp \rangle \to 2$
$[\![\texttt{tt}]\!]_s = tt$
$[\![\texttt{ff}]\!]_s = ff$
$[\![B_1 \texttt{ and } B_2]\!]_s = [\![B_1]\!]_s \wedge [\![B_2]\!]_s$
$[\![B_1 \texttt{ or } B_2]\!]_s = [\![B_1]\!]_s \vee [\![B_2]\!]_s$
$[\![\texttt{not } B]\!]_s = \neg [\![B]\!]_s$
$[\![A_1 = A_2]\!]_s = ([\![A_1]\!]_s == [\![A_2]\!]_s)$
$[\![A_1 \texttt{<=} A_2]\!]_s = ([\![A_1]\!]_s \leq [\![A_2]\!]_s)$

$[\![-]\!]_s : \langle AExp \rangle \to \mathbb{Z}$
$[\![n \in \mathbb{Z}]\!]_s = n$
$[\![x]\!]_s = s(x)$
$[\![A_1 - A_2]\!]_s = [\![A_1]\!]_s - [\![A_2]\!]_s$
$[\![A_1 + A_2]\!]_s = [\![A_1]\!]_s + [\![A_2]\!]_s$
$[\![A_1 * A_2]\!]_s = [\![A_1]\!]_s \times [\![A_2]\!]_s$

Now, we attempt to model the commands. As previously mentioned, we can view a command as something which modifies the current state, so we should be able to denote commands as functions $\Sigma \to \Sigma$. Let's try that, starting with the simplest commands.

The `skip` command does nothing, so it should be denoted by the identity function on Σ.

$$[\![\texttt{skip}]\!] = id_\Sigma$$

The `def` command assigns the value of an arithmetic expression A to variable x. So it should first evaluate A, and then take the state whose x-value has been modified to be $[\![A]\!]$.

$$[\![\texttt{def } x \texttt{ := } A]\!] = (s \mapsto s[[\![A]\!]_s / x])$$

The sequencing command runs one command after another, which is just composing the two modifications:

$$[\![C_1; C_2]\!] = [\![C_2]\!] \circ [\![C_1]\!]$$

The `if` command evaluates a Boolean expression B and evaluates one command or the other depending on whether B evaluated to true or false.

$$[\![\texttt{if } B \texttt{ then } C_1 \texttt{ else } C_2]\!] = (s \mapsto \text{if } [\![B]\!]_s \text{ then } [\![C_1]\!](s) \text{ else } [\![C_2]\!](s))$$

Finally, the `while` command checks a boolean expression B, and then evaluates the command C if it is true, then checks B, and then evaluates C, and so on until B is not true. How can we model this mathematically? Well here's a clever trick. This is just the same as

saying check B, and if it is true, run C and then run the exact same `while` command again.

$$[\![\texttt{while } B \texttt{ do } C]\!] = (s \mapsto \text{if } [\![B]\!]_s \text{ then } [\![\texttt{while } B \texttt{ do } C]\!] \circ [\![C]\!]\,(s) \text{ else } s)$$

This is all well and good, until you try to evaluate a program that doesn't/shouldn't terminate on certain inputs, like

$$\texttt{while } x \texttt{ >= } 0 \texttt{ do skip}$$

If you are a diligent human being evaluating this program according to the definition we just gave, then what happens is that you will also not terminate, as long as x starts off larger than 0. Indeed, let's try it with $x = 10$:

$$\begin{aligned}
&[\![\texttt{while } x \texttt{ >= } 0 \texttt{ do skip}]\!]\,([x \mapsto 10]) \\
=\ &\text{if } 10 \geq 0 \text{ then } [\![\texttt{while } x \texttt{ >= } 0 \texttt{ do skip}]\!] \circ [\![\texttt{skip}]\!]\,([x \mapsto 10]) \text{ else } [x \mapsto 10] \\
=\ &[\![\texttt{while } x \texttt{ >= } 0 \texttt{ do skip}]\!] \circ [\![\texttt{skip}]\!]\,([x \mapsto 10]) \\
=\ &[\![\texttt{while } x \texttt{ >= } 0 \texttt{ do skip}]\!]\,([x \mapsto 10]) \\
=\ &\ldots
\end{aligned}$$

The first immediate observation is that a command does not always denote a *total* function $\Sigma \to \Sigma$, since as the above example shows, some commands may be undefined under certain starting states. Hence, we have to expand our space of denotations to include partial functions. The second observation is that, viewing the definitions we gave as a rewrite rule indeed yields undefinedness. However, viewing it as an equation where the $[\![\texttt{while } B \texttt{ do } C]\!]$ value is an unknown to solve for, then we can in fact find functions $\Sigma \to \Sigma$ satisfying the given equation. For example, considering our example before, the identity function is one such solution. However, it is not the unique solution: we can also consider the function which increments states with x-value above 0:

$$s \mapsto \begin{cases} s & \text{if } s(x) < 0 \\ s[s(x) + 1/x] & \text{otherwise} \end{cases}$$

This suggests our endeavor was not rotten to the core: the definitions we gave are satisfiable, just that we don't have a canonical way of picking the right solution. In fact, the move to partial functions fixes this problem. The key idea is that we can view partial functions as *approximations* of the total functions. More generally, given two partial functions $f, g : \Sigma \rightharpoonup \Sigma$, if $g(s)$ is defined and equal to $f(s)$ whenever $f(s)$ is defined, then we say that f approximates[2] g. In other words, g carries all the information that f does, and possibly more. We denote this relation as $f \sqsubseteq g$, which turns $\Sigma \rightharpoonup \Sigma$ into a poset.

Let us now re-inspect our example of `while x >= 0 do skip` more closely. We see that any solution to the corresponding equation must map states with x-value < 0 to the same state, but that it is free to do anything on other states! This suggests that the information of what

[2] This is not standard use of the word "approximate" in domain theory, for the notion of approximation is usually defined as a more restrictive relation on elements of a domain. But for now it gets the point across.

the solution does to states with x-value ≥ 0 is irrelevant to solving the equation. Therefore, we might as well take the minimal solution, which is the partial function

$$f(s) = \begin{cases} s & \text{if } s(x) < 0 \\ \text{undefined} & \text{otherwise} \end{cases}$$

In mathematical terms, this corresponds to taking the meet or greatest lower bound of all solutions to the equation. The nice thing is that the minimal solution seems to correspond to our computational intuition: the function f is only defined on states with x-value where our intuition tells us it should terminate (and vice versa). However, the method by which we obtained the minimal solution is not very computational, in the sense that it does not correspond to our original idea of computing $[\![\texttt{while } B \texttt{ do } C]\!]$ recursively in terms of itself. In our original definition, the denotation is only well-defined on a state s if the recursion is well-founded: i.e. at some point in the recursive process, we take the "else" case which does not contain a recursive call to $[\![\texttt{while } B \texttt{ do } C]\!]$, and so we are able to stop the recursion.

We can view this bottom-up instead of top-down: knowing nothing about the next recursive call, what are the states s on which we can already compute $[\![\texttt{while } B \texttt{ do } C]\!](s)$? It's exactly those states s for which $[\![B]\!]_s$ is false, since those don't contain the recursive call. Now, repeating this process: now knowing this information on the next recursive call, what else can we compute? Well it will be those states s' s.t. $s = [\![C]\!](s')$ for some s with $[\![B]\!]_s$. We can keep this iterative process going, and in the "limit" compute the minimal solution.

To express this formally, we need two ingredients: first, an element of $\Sigma \rightharpoonup \Sigma$ which represents the empty approximation, and second, a function which given the denotation of the next recursive call, computes the current denotation of $[\![\texttt{while } B \texttt{ do } C]\!]$. The empty approximation contains no information, so it stands to reason that it should approximate any element of $\Sigma \rightharpoonup \Sigma$, albeit in a vacuous sense. Indeed $\Sigma \rightharpoonup \Sigma$ has a least element $\bot$, also called the bottom element, which is the partial function undefined on all inputs. For the second ingredient, this has to be the definition of $[\![\texttt{while } B \texttt{ do } C]\!]$, except with the recursive call replaced by the argument. Hence, we obtain a function $F_{B,C} : (\Sigma \rightharpoonup \Sigma) \to (\Sigma \rightharpoonup \Sigma)$, defined as

$$F_{B,C}(f) = (s \mapsto \text{if } [\![B]\!]_s \text{ then } f \circ [\![C]\!](s) \text{ else } s)$$

Solutions of the equation expressed by $[\![\texttt{while } B \texttt{ do } C]\!]$ correspond to fixpoints[3] of $F_{B,C}$ and we are seeking to compute the least fixpoint. Expressing this computation amounts to constructing the following sequence of approximations

$$\bot \sqsubseteq F_{B,C}(\bot) \sqsubseteq F_{B,C}(F_{B,C}(\bot)) \sqsubseteq \ldots \sqsubseteq F^n_{B,C}(\bot) \sqsubseteq \ldots$$

and taking the denotation of the while loop to be the "limit" of this sequence.

[3] i.e. those elements f s.t. $f = F_{B,C}(f)$.

To illustrate this process, we consider our original example, where the minimal solution can be computed in one step. For a second example on the other extreme end, we consider the factorial program, for which we only obtain the fixpoint after infinitely[4] many steps.

[4] This doesn't mean that its not feasible, because for any particular starting state s we can still find an approximation which is defined on s after finitely many steps.

Example 1.1.1. *In our original example, the corresponding higher-order function is*

$$F(f)(s) = \begin{cases} f(s) & \text{if } s(x) \geq 0 \\ s & \text{otherwise} \end{cases}$$

Therefore, taking $f = \bot$, we have

$$F(\bot)(s) = \begin{cases} s & \text{if } s(x) < 0 \\ \text{undefined} & \text{otherwise} \end{cases}$$

But then we also have that

$$F^2(\bot)(s) = \begin{cases} s & \text{if } s(x) < 0 \\ \text{undefined} & \text{otherwise} \end{cases}$$

Hence, $F(\bot) = F(F(\bot)) = F(F(F(\bot))) = \ldots$, so the limit of the sequence $F^n(\bot)$ is $F(\bot)$.

Example 1.1.2. *The higher-order function associated with the while-loop on the right is*

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

$$F(f)(s) = \begin{cases} f(s[y/s(y) \times s(x)][x/s(x) - 1]) & \text{if } s(x) \geq 1 \\ s & \text{otherwise} \end{cases}$$

Clearly, the first iteration will be undefined except when the x-value is less than 1.

$$F(\bot)(s) = \begin{cases} s & \text{if } s(x) < 1 \\ \text{undefined} & \text{otherwise} \end{cases}$$

In the second iteration, we get also the value for when $s(x) = 1$.

$$\begin{aligned} & F(F(\bot))(s) \\ &= \begin{cases} s & \text{if } s(x) < 1 \\ F(\bot)(s[y/s(y) \times s(x)][x/s(x) - 1]) & \text{if } s(x) \geq 1 \end{cases} \\ &= \begin{cases} s & \text{if } s(x) < 1 \\ s[y/s(y) \times s(x)][x/s(x) - 1] & \text{if } s(x) = 1 \\ \text{undefined} & \text{otherwise} \end{cases} \\ &= \begin{cases} s & \text{if } s(x) < 1 \\ s[y/s(y) \times 1][x/0] & \text{if } s(x) = 1 \\ \text{undefined} & \text{otherwise} \end{cases} \end{aligned}$$

And so on...

$$F(F(F(\bot)))(s) = \begin{cases} s & \text{if } s(x) < 1 \\ s[y/s(y) \times 1][x/0] & \text{if } s(x) = 1 \\ s[y/s(y) \times 1 \times 2][x/0] & \text{if } s(x) = 2 \\ \text{undefined} & \text{otherwise} \end{cases}$$

In the limit, we have that

$$(\lim_{n\in\mathbb{N}} F^n(\bot))(s) = \begin{cases} s & \text{if } s(x) < 1 \\ s[y/s(y) \times s(x)!][x/0] & \text{if } s(x) \geq 1 \end{cases}$$

Since we start off by setting y to be 1, the denotation of the overall program becomes:

$$s \mapsto \begin{cases} s[y/1] & \text{if } s(x) < 1 \\ s[y/s(x)!][x/0] & \text{if } s(x) \geq 1 \end{cases}$$

In other words, this program computes the factorial of x and stores it in y.

Our investigation here reveals the inadequacy of interpreting programs in the category of sets, and requires us to make the move to certain posets, which we shall call *domains*, whose properties abstract that of the poset $\Sigma \rightharpoonup \Sigma$. Specifically, domains carry those properties that were necessary in computing the least fixpoint. This means the structures we consider are posets with certain "limits", and the maps between them are functions that preserve these limits, i.e. "continuous" functions. As we will motivate later, our intuition on what a computable function is requires that *computable functions are continuous.* This is the slogan of domain theory.

The main focus of chapter 2 will be exploring the mathematical theory of domains. As we will see, there is no single formal notion of domain, so when we say "domain" we refer vaguely to any of the suitable formal definitions we will define.

1.2 *Semantic Modelling of Procedures with* **ImpProc**

In the toy language of the previous section, a key tool in modern programming languages is missing: the ability to define and later re-use procedures. How can we semantically model such behavior? Let us agree for simplicity that defining a procedure simply assigns a command to a procedure name, without allowing any explicit input or output variables to the procedure[5]. We add to the grammar the following commands:

$$\langle \mathit{Command} \rangle ::= \ \ldots \mid \text{def } \langle \mathit{ProcName} \rangle := \langle \mathit{Command} \rangle \mid \text{run } \langle \mathit{ProcName} \rangle$$

and call this language **ImpProc**.

Let us now try to see what changes we need to make to the semantic model from the previous section. For one, we need to modify the state model Σ to allow it to store procedures, so it has to be

$$\Sigma = (\langle \mathit{Var} \rangle \to \mathbb{Z}) \times (\langle \mathit{ProcName} \rangle \to (\Sigma \rightharpoonup \Sigma)).$$

But hold on, this already seems fishy... We are asking for Σ to contain a copy of the space of all functions from $\Sigma \to \Sigma$. As we know thanks to Cantor, there is no (non-trivial) set with this property, even if we weaken the equality $=$ to an isomorphism $\cong$. However, we have

[5] Of course, one can still put the inputs in some variables and have the procedure read those variables, and similarly the procedure can pass an output by assigning it to some pre-determined variable(s). So there is really no loss of expressivity here.

previously established sets to be insufficient for the modelling of programming languages anyway. The solution then is to consider Σ itself as a domain. Its defining criterion is that it should satisfy the equation above (up to isomorphism). As we have also seen before, this can be rephrased as asking for a least fixpoint of

$$F(D) := (\langle Var \rangle \to \mathbb{Z}) \times (\langle ProcName \rangle \to (D \rightharpoonup D))$$

In this scenario, we have to consider F as an endofunctor on the "category of domains"[6]. As we will see, suitable categories of domains will allow us to systematically compute the least fixpoint of a functor by a similar "limiting process" as we did in the previous section. We call equations such as the one above *domain equations*.

[6] In quotes because as we mentioned, there are multiple suitable notions of domain, none of which are the canonically correct choice in all scenarios.

Our restriction to considering only the continuous maps is precisely what distinguishes "the category of domains" from the category of sets. There are too many set-theoretic functions! They grow exponentially. On the other hand, there are few enough continuous functions that we can find domains whose continuous functions can be suitably embedded inside itself. Moreover, the continuous maps are sufficient since they include the computable functions, as per our slogan.

1.3 *Reasoning About Programs*

The use of "limits" and "continuity" suggests that there are topological ideas underlying our notion of domains. Indeed, the order-theoretic development of domain theory in chapter 2 can be recast in terms of topological spaces, and this is what we will do in the beginning of chapter 3. The starting observation for this topologization is that if $\sqsubseteq$ is an ordering of the information content in domain elements, then we can analyze this relation in terms of the properties exhibited by elements:

$$x \sqsubseteq_D y \iff \text{for every property } P \text{ of } x,\ y \text{ also exhibits } P.$$

Taking the properties to be subsets of a domain D, this says that any property has to be upwards-closed with respect to $\sqsubseteq_D$, and the upwards-closed subsets of D forms a topology on D. However, this is not the topology we want - we have to further restrict to the set of properties which are "finitely observable", so that our notion of "limit" and "continuity" correspond to the actual topological notions. More about this is said in chapter 3.

The topologization of domains allows us to apply the techniques of Stone duality, which says that we have dual correspondences between certain (categories of) spaces and (categories of) logical theories. Equipped with the view that a category of domains is a category of spaces, Stone duality gives us a logical system for reasoning about programs. This application of Stone duality to domains is originally due to [Abr91], although that paper is somewhat lacking in exposition about the underlying duality itself. In this report, we aim to give

a more well-rounded introduction to domain theory and specifically its duality-theoretic aspects. Most of chapter 3 will be about sowing the seeds of Stone duality with an eye towards domains. This chapter fills in the missing exposition on domain-theoretic duality needed to properly understand [Abr91]. In chapter 4, we reap the benefits of what we sowed to create "off-the-shelf" program logics whose structure is essentially determined by the underlying domains. In this chapter, we will focus more on exposition and less on proofs since the proofs can be found in [Abr91]. We also give a concrete and simple example by constructing a program logic for the imperative language introduced in this chapter, and compare it with Hoare logic, which is the standard program logic for reasoning about imperative programs.

1.4 Prerequisites

We assume that the reader is familiar with imperative programming languages, at least to the extent of being able to understand the languages we introduced in this chapter. On the mathematical side, we assume the reader is familiar with the basic language and definitions of category theory, specifically functors, limits/colimits and adjunctions. We also assume familiarity with point-set topological notions, the most prominent being compactness which plays a big role in making everything work.

2
Domain Theory

2.1 Directed-complete Partial Orders

Following the motivations laid out by the previous introductory chapter, we introduce a category of posets with certain limits. In a poset, a limit of a sequence of increasing elements can be understood as its least upper bound[1] (LUB) or join, so we are really asking for posets with certain LUBs.

[1] AKA a colimit, in categorical terminology.

Working with sequences is finicky, and will often require us to make arbitrary choices. To avoid this, we introduce instead a generalization of sequences called directed subsets, and ask for least upper bounds of such subsets to exist. It can be shown that the existence of LUBs for sequences[2] correspond to the existence of LUBs for directed subsets.

[2] of arbitrary ordinal length, not just countable.

Definition 2.1.1. Let $(D, \sqsubseteq)$ be a partially ordered set. A non-empty subset $A \subseteq D$ is *directed* if for every $x, y \in A$ there is $z \in A$ s.t. $x \sqsubseteq z$ and $y \sqsubseteq z$. We also say that a least upper bound $\bigsqcup A$ is a *directed join* if A is a directed subset.

In addition to the limits, the computation of least fixed-points also relies on the existence of the bottom element, so we enforce this as well.

Definition 2.1.2. A poset $(D, \sqsubseteq)$ is *directed complete* if every directed set A in D has a least upper bound, denoted $\bigsqcup A$, and D has a least element $\bot$.

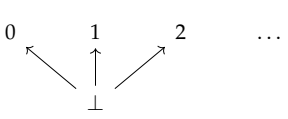
We make the abbreviation DCPO for directed-complete partially ordered sets.

Example 2.1.3. *Recall the $\Sigma \rightharpoonup \Sigma$ poset we introduced in the previous chapter. This poset is directed-complete with the join defined as taking the union of the graph[3] of the functions. It has a bottom element being the function that is undefined on all inputs. In fact, for any sets A, B, the poset $A \rightharpoonup B$ is directed-complete.*

[3] i.e. the input-output pairs

Example 2.1.4. *Given any set A, we can construct the free DCPO $A_\bot = A \cup \{\bot\}$, which is just an order where $a \sqsubseteq b \iff a = \bot \vee a = b$, i.e. it is a flat order where any two distinct elements of A are incomparable and $\bot$ is below everything else.*

$\mathbb{N}_\bot$



Example 2.1.5. *(**Tapes** [Plo83]) Suppose we have a turing machine (TM) with an input tape and an output tape. The tape squares can be blank, or, after printing, contain a* 0 *or a* 1*. The observable events that can occur are the printing of a* 0 *or a* 1 *on an output tape square. Hence, the DCPO* **Tapes** *consists of the space of possible sets of events that can occur, ordered by subset inclusion. Equivalently, this is the set of all finite or infinite binary sequences with the subsequence ordering.*

Example 2.1.6. *($\mathcal{P}\omega$ [Plo83]) Taking a different view of what constitutes an output event, we can obtain a different DCPO from the same TM situation. If we agree that outputting a* 0 *followed by n* 1*s and then another* 0 *constitutes outputting the natural number n, then the appropriate DCPO is $\mathcal{P}\omega$.*

As previously mentioned, the appropriate notion of mapping between DCPOs consists of functions that preserve directed joins, i.e. continuous. This allows us to form the category **DCPO** of DCPOs with continuous functions as morphisms.

Definition 2.1.7. A function $f : D \to E$ between DCPOs is

1. *monotone*, if for all $x, y \in D$, if $x \sqsubseteq_D y$ then $f(x) \sqsubseteq_E f(y)$.
2. *continuous*, if it is monotone and for all directed subsets $A \subseteq D$, we have $f(\bigsqcup A) = \bigsqcup f[A]$.

Proposition 2.1.8. *1. The identity function $id_D : D \to D$ is continuous.*

2. The composition of continuous functions is continuous.

If we think of such a mapping f as a program transformation, then monotonicity is saying that if x approximates y, then the transformation must act the same on x as it acts on the part of y approximated by x. The continuity condition means that the program transformation does not behave erratically in the limit: if d is built up from a sequence of approximations $\bigsqcup A$, then $f(d) = f(\bigsqcup A)$ is built up from how it acts on the approximations, i.e. $\bigsqcup f[A]$. Both of these are behaviors one would expect from a computable function, so the slogan is therefore:

Computable functions are continuous!

However, note that we do not ask for these mappings to preserve the bottom element. Indeed, requiring $f(\bot) = \bot$ means that the limit $\bigsqcup_{n\in\mathbb{N}} f^n(\bot)$ is always going to be $\bot$, which is not what we want. From a more intuitive perspective, we want f to act as an information-adder, in the sense that every application of f contributes additional information. We are then seeking the minimal element which is f-saturated, i.e. which already contains all information f can contribute to it. Forcing $f(\bot) = \bot$ means that $\bot$ is f-saturated, meaning f isn't actually adding any information at all.

We are now in position to prove the fact that $\bigsqcup_{n\in\mathbb{N}} f^n(\bot)$ is in fact the least fixed point.

Theorem 2.1.9. *Let $f : D \to D$ be a continuous function. Then*

1. *$\bigsqcup_{n\in\mathbb{N}} f^n(\bot)$ is a fixed point of f.*
2. *For any fixed point d of f, we have $\bigsqcup_{n\in\mathbb{N}} f^n(\bot) \sqsubseteq d$.*

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

2. It suffices to show that each $f^n(\bot) \sqsubseteq d$. But $\bot \sqsubseteq d$ is always true, so by monotonicity of f, we have $f^n(\bot) \sqsubseteq f^n(d) = d$.

□

2.2 Algebraic DCPOs

In the poset $\Sigma \rightharpoonup \Sigma$ of the previous chapter, every element can be constructed as the least upper bound of a sequence of finite partial functions[4]. From a computational point of view, this is desirable because it means the DCPO is finitely generated, i.e. we only have infinite elements arising in the limit of computations on finite elements. Moreover, as we will see later from the topological perspective, this property[5] is what allows us to apply Stone duality.

[4] i.e. those that are defined on only finitely many inputs.

[5] There are weaker properties that also allow this, such as with continuous domains. See [AJ95] for details.

To express this property on an arbitrary domain, we have to figure out a way of expressing "finiteness" in a purely order-theoretic way. The key tool here is the least upper bounds. If we have a finite partial function f, then given any sequence of finite partial functions $(g_n)_{n\in\mathbb{N}}$ containing f in the limit, at some point g_n in the sequence we must already contain all of f, since f only contains finitely many entries. This is not true if f is infinite, for example consider a partial function $f : \mathbb{N} \to \mathbb{N}$ that is defined on exactly the even numbers.

Definition 2.2.1. Let D be a DCPO, and $x \in D$. The element x is *compact* if for any directed subset $A \subseteq D$, whenever $x \sqsubseteq \bigsqcup A$, there is some $a \in A$ s.t. $x \sqsubseteq a$ already.

Note that we called it "compact" and not "finite". The reason for this is that it doesn't quite fully capture our intuitive notion of finiteness. For example, the ordinal $2\omega + 1$ is directed complete[6], but its compact elements consist of the successor ordinals, including the infinite ones such as $\omega + 1$. The successor ordinals are "compact"[7] in the topological sense: using a sequence, one cannot infinitely "procrastinate" approaching the successor ordinal. This is quite a weird example that one doesn't typically encounter in denotational semantics. For more conventional DCPOs, this adequately captures finiteness.

[6] It is easy to check that any successor ordinal is directed complete, while no limit ordinal is directed complete.

[7] In fact, this definition is a generalization of the topological notion of compactness. If we apply this definition to the lattice of open subsets of a space instead of a DCPO, this yields the standard notion of topological compactness.

Example 2.2.2. *1. The bottom element is compact in any DCPO.*

2. *Every element is compact in a flat or finite DCPO.*
3. *The compact elements in the powerset lattice $\mathcal{P}A$ are exactly the finite ones.*

We can now express what it means for a DCPO to be "finitely generated".

Definition 2.2.3. Let D be a DCPO.

1. Let $\mathbf{K}D$ denote the subposet of compact elements in D.
2. Let $\downarrow_{\mathbf{K}} x = \downarrow x \cap \mathbf{K}D$ denote the subset of compact elements below $x \in D$, also called its set of approximations[8].
3. D is *algebraic* iff for every element $x \in D$, $\downarrow_{\mathbf{K}} x$ is directed, and $x = \bigsqcup \downarrow_{\mathbf{K}} x$.
4. Let **AlgDCPO** denote the full subcategory of **DCPO** containing the algebraic DCPOs.

[8] This is where the use of the word "approximation" starts to become more specific than the use in the introductory chapter.

We will now build up some results showing that the behavior of any element x in an algebraic DCPO D is determined by its approximations. This will culminate in a representation theorem for algebraic DCPOs in terms of their subposet of compact elements.

Proposition 2.2.4. *Let D be an algebraic DCPO. Then*

1. *$x \sqsubseteq_D y$ iff $\downarrow_{\mathbf{K}} x \subseteq \downarrow_{\mathbf{K}} y$; and*
2. *if $A \subseteq D$ is directed, $\downarrow_{\mathbf{K}} \bigsqcup A = \bigcup_{a \in A} \downarrow_{\mathbf{K}} a$.*

A particular corollary of the above is that $x = y$ iff $\downarrow_{\mathbf{K}} x = \downarrow_{\mathbf{K}} y$. If every element is determined by some directed subset of compact elements, then given just the compact elements $\mathbf{K}D$ we can recover D as the poset of directed subsets of $\mathbf{K}D$. Well, this is almost true except that an element can arise as the join of many different directed subsets. Therefore, we have to choose a canonical representative directed subset. For this, we can take the maximal one, AKA the downwards closed one.

Definition 2.2.5. Let P be a poset. An *ideal* is a downwards closed directed subset of P. We denote the poset of ideals in P by $\mathbf{Idl}(P)$.

Proposition 2.2.6. *Let $A, B \subseteq \mathbf{K}D$ be directed subsets. Then*

1. *$\downarrow_{\mathbf{K}} A$ and $\downarrow_{\mathbf{K}} B$ are ideals in $\mathbf{K}D$; and*
2. *$\bigsqcup A = \bigsqcup B$ iff $\downarrow_{\mathbf{K}} A = \downarrow_{\mathbf{K}} B$.*

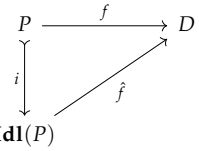
Using this, we can develop the representation theorem of algebraic DCPOs by their compact elements.

Theorem 2.2.7. *Let P be a poset.*

1. *The map $i : p \mapsto \downarrow p$ is an order embedding $P \to \mathbf{Idl}(P)$, with the image of i being $\mathbf{K}(\mathbf{Idl}(P))$. In other words, the compact elements of $\mathbf{Idl}(P)$ is exactly P.*
2. *$\mathbf{Idl}(P)$ is an algebraic DCPO.*
3. *Let $f : P \to D$ be a monotone map. Then it extends uniquely[9] to a continuous map $\mathbf{Idl}(P) \to D$.*

Corollary 2.2.8 (representation theorem for algebraic DCPOs)**.** *Every algebraic DCPO D is isomorphic to the ideal completion of a poset, specifically $D \cong \mathbf{Idl}(\mathbf{K}D)$.*

[9] i.e. there is a unique map $\hat{f} : \mathbf{Idl}(P) \to D$ making the following diagram commute:



By category theory, this makes $\mathbf{Idl}(-)$ into a functor left adjoint to the forgetful functor $U : \mathbf{AlgDCPO} \to \mathbf{Pos}$.

2.3 Constructions on DCPOs

In this section, we explore the basic constructions in the category of (algebraic) DCPOs. Many of the structures we need in denotational semantics can be obtained by performing these constructions on some basic DCPOs. Moreover, we want the constructions to be functorial, because we also want to solve for the least fixpoints of functors defined as composites of these constructions, as motivated in the previous chapter.

For brevity, we omit many of the proofs in this section, since they amount to a lot of routine verification of categorical properties. We refer to Chapter 2 of [SLG94] for the proofs, which this section mostly draws from.

2.3.1 Lifting DCPOs

Given a set S, how can we "freely" generate a DCPO? We can do so by adding a bottom element to S, and ordering it such that any two elements of S are incomparable, and the bottom element is below everything else. Of course, we can do this not just for a set S, but for any poset P, although in that case it will not be directed complete unless P itself already has directed joins.

Definition 2.3.1. Let $(P, \leq)$ be a poset. The *lift* $P_\perp$ is the set $P \cup \{\perp\}$ along with the ordering

$$x \sqsubseteq_{P_\perp} y \iff \begin{cases} x \leq y & \text{if } x, y \in P \\ \text{true} & \text{if } x = \perp \\ \text{false} & \text{if } y = \perp \end{cases}$$

Proposition 2.3.2. *If P is a directed complete poset, then*

1. *$P_\perp$ is a DCPO;*
2. *$\mathbf{K}P_\perp = \mathbf{K}P \cup \{\perp\}$;*
3. *if P is an algebraic DCPO, then so is $P_\perp$.*

As usual, we functorialize our construction.

Definition 2.3.3. Let $(-)_\perp : \mathbf{DCPO} \to \mathbf{DCPO}$ be the functor which assigns a DCPO D to $D_\perp$, and any continuous function $f : D \to E$ to the function $f_\perp : D_\perp \to E_\perp$ defined by

$$f_\perp(d) = \begin{cases} f(d) & \text{if } d \in D \\ \perp & \text{if } d = \perp \end{cases}$$

2.3.2 Product of DCPOs

Definition 2.3.4. Let D, E be posets. The product $D \times E$ is given by the cartesian product of D and E as sets, endowed by the ordering

$$(d, e) \sqsubseteq_{D \times E} (d', e') \iff d \sqsubseteq_D d' \text{ and } e \sqsubseteq_E e'.$$

Moreover, we have monotone projection maps $\pi_1 : D \times E \to D$ and $\pi_2 : D \times E \to E$.

Proposition 2.3.5. *Let D, E be DCPOs. Then*

1. *$D \times E$ is a DCPO and the projection maps are continuous;*
2. *$\mathbf{K}(D \times E) = \mathbf{K}D \times \mathbf{K}E$;*
3. *if D and E are algebraic then so is $D \times E$.*

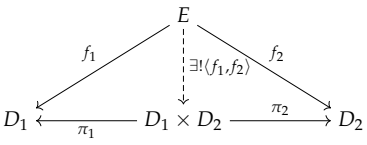
Proof. The proof of 1. is a routine verification after observing that $\bigsqcup S = (\bigsqcup \pi_1(S), \bigsqcup \pi_2(S))$.

For 2, it is clear that a pair of compact elements (d, e) is compact in the product. On the other hand, if we have a pair (d, e) that is compact, then we can see that d is compact as follows. Given a directed subset $S \subseteq D$, define the directed subset $S' = \{(s, e) \mid s \in S\}$. Then since $(d, e) \sqsubseteq \bigsqcup S$, there is some (s, e) such that $(d, e) \sqsubseteq (s, e)$, i.e. $d \sqsubseteq s$. The proof that e is compact is completely analogous. 3. then follows fairly quickly from 2. by considering that $\downarrow_{\mathbf{K}}(d, e) = \downarrow_{\mathbf{K}} d \times \downarrow_{\mathbf{K}} e$. □

Next, we characterize the continuous functions into and out of the product. The latter is the universal property which shows that the product of DCPOs is indeed the categorical product in the category of DCPOs.

Proposition 2.3.6. *Let D_1, D_2, E be DCPOs.*

1. *A function $f : D_1 \times D_2 \to E$ is continuous iff f is continuous in each argument, i.e. $f(d, -)$ is continuous for each $d \in D_1$ and $f(-, d)$ is continuous for each $d \in D_2$.*
2. *For any two continuous functions $f_1 : E \to D_1$ and $f_2 : E \to D_2$ there is a unique continuous function $\langle f_1, f_2 \rangle : E \to D_1 \times D_2$ such that $\pi_1 \circ \langle f_1, f_2 \rangle = f_1$ and $\pi_2 \circ \langle f_1, f_2 \rangle = \pi_2$.*

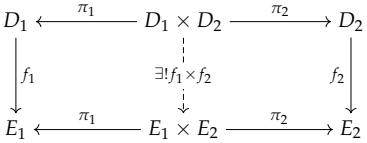


Note that while the universal property of the product is stated in the category **DCPO**, since the full subcategory of algebraic DCPOs is closed under products, this is also the correct notion of categorical product in **AlgDCPO**. Moreover, as is standard in category theory, the universal property of the categorical product allows us to lift it into a functor.

Definition 2.3.7. Let $- \times - : \mathbf{DCPO} \times \mathbf{DCPO} \to \mathbf{DCPO}$ be the functor which assigns to any pair (D_1, D_2) the product $D_1 \times D_2$, and to any map of pairs $(f_1, f_2) : (D_1, D_2) \to (E_1, E_2)$ the map $f_1 \times f_2 := \langle f_1 \circ \pi_1, f_2 \circ \pi_2 \rangle$ induced by the universal property of the product, as in the diagram on the right.

Moreover, this functor restricts to

$$- \times - : \mathbf{AlgDCPO} \times \mathbf{AlgDCPO} \to \mathbf{AlgDCPO}.$$



The product construction means that in $D \times E$ there are elements of the form $\langle \bot, e \rangle$ or $\langle d, \bot \rangle$. If we consider the product to be the type of tuples in a programming language, and interpret $\bot$ as a non-terminating computation, this does not always make sense since for example in Python we would expect the operation of forming a pair to not terminate if one of its argument doesn't terminate[10], i.e. $\langle \bot, e \rangle = \bot$.

We can define a product construction in which this is indeed the case, although of course it loses the universal property in **DCPO**.

[10] A programming language that tries to evaluate the arguments to a function call before evaluating the function is called an *eager* or *strict* programming language. Your typical imperative programming languages such as Python and Java are eager. A standard example of a language that does not behave like this is Haskell, which is usually called a *lazy* language in contrast.

Definition 2.3.8. Let D and E be DCPOs. We define the *strict product* $D \otimes E$ to be the poset $(D - \{\bot\}) \times (E - \{\bot\})_\bot$ where the ordering on the poset $D - \{\bot\}$ and $E - \{\bot\}$ are inherited from D and E.

2.3.3 Coalesced & Separated Sum

We have discussed the product of DCPOs, but what about the coproduct? Between posets, we have the following notion of coproduct:

Definition 2.3.9. Let P and Q be posets. We define an order on the disjoint sum $P \uplus Q$ defined by

$$x \leq_{P \uplus Q} y \iff x \leq_P y \in P \text{ or } x \leq_Q y \in Q$$

Of course, given DCPOs D, E, the poset $D \uplus E$ is no longer a DCPO - it does not have a bottom element, even if it is directed complete. We can rectify this by using lifting, but unfortunately the notion we obtain will not be the categorical coproduct[11].

[11] and in fact, **DCPO** cannot have coproducts, since the combination of domain equations, cartesian closure and coproducts do not play nice [HP90].

Definition 2.3.10. Let D and E be DCPOs. Let the *separated sum* be

$$D + E := (D \uplus E)_\bot$$

Note that the separated sum has three "bottom" elements, which is the one inherited from D, the one from E, and the real bottom element. As before, we can also define a strict version which collapses all these bottom elements together.

Definition 2.3.11. Let D and E be DCPOs. The *strict sum* $D \oplus E$ is the DCPO $((D - \{\bot\}) \uplus (E - \{\bot\}))_\bot$.

Note that the separated sum can be recovered from the strict sum by taking $D + E = D_\bot \oplus E_\bot$.

2.3.4 Function Spaces

Definition 2.3.12. Let D and E be DCPOs. Then the *function space* $[D \to E]$ is the set $\{f : D \to E \mid f \text{ continuous}\}$ equipped with the ordering

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

There is an associated *evaluation* map $\mathsf{ev} : [D \to E] \times D \to E$ defined by $\mathsf{ev}(f, d) = f(d)$.

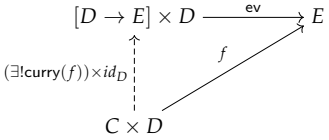
Theorem 2.3.13. *Let C, D, E be DCPOs.*

1. *Then $[D \to E]$ is a DCPO, with least element $\lambda d.\ \bot_E$, and if $S \subseteq [D \to E]$ is a directed set, then*

$$(\bigsqcup S)(d) = \bigsqcup \{f(d) \mid f \in S\}$$

2. *The evaluation map* $\mathsf{ev} : [D \to E] \times D \to E$ *is continuous.*

3. *Given a continuous function $f : C \times D \to E$, there is a unique continuous function* $\mathsf{curry}(f) : C \to [D \to E]$ *such that it makes the diagram on the right commute, i.e.* $\mathsf{ev}(\mathsf{curry}(f)(c), d) = f(c, d)$ *for all $c \in C$ and $d \in D$.*



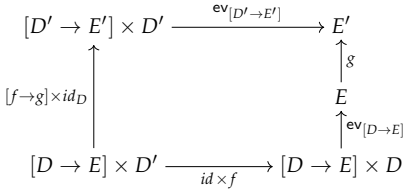
The last item in the theorem above yields the universal property of $[D \to E]$ as the exponential E^D in **DCPO**. Once again, as is familiar from category theory we can functorize this construction using the universal property.

Definition 2.3.14. Let $[- \to -] : \mathbf{DCPO}^{op} \times \mathbf{DCPO} \to \mathbf{DCPO}$ be the functor which sends pairs of DCPOs (D, E) to $[D \to E]$, and pairs of maps $(f : D' \to D, g : E \to E')$ to the map

$$[f \to g] := \mathsf{curry}(g \circ \mathsf{ev}_{[D \to E]} \circ id_{[D \to E]} \times f) : [D \to E] \to [D' \to E']$$

Explicitly, it is given by

$$h \mapsto g \circ h \circ f$$



Recall that the key point of moving to DCPOs was our ability to find least fixpoints of arbitrary continuous endofunctions $f : D \to D$. The method of finding the least fixed point by considering chains of $f^n(\bot)$ is considered to be a computation of some sort. We now make this precise by showing that the mapping $\mathsf{fix} : [D \to D] \to D$ taking endofunctions to their least fixed points is continuous[12].

[12] Recall our slogan: every computable function is continuous. We do not necessarily mean the inverse: "continuous functions are computable". However, using only the tools of domain theory we introduced so far, the continuous functions are our best approximation of which functions are computable.

Theorem 2.3.15. *The function fix $: [D \to D] \to D$ defined by*

$$f \mapsto \bigsqcup_{n \in \mathbb{N}} f^n(\bot)$$

is continous.

Finally, we have also a strict version of this construction. Recall that we said in most programming languages, a function call evaluates its arguments before evaluating the function. Therefore, it does not terminate unless the arguments terminate, i.e. $f(\bot) = \bot$. We call such a function strict, and consider the space of strict functions.

Definition 2.3.16. Let D and E be DCPOs. The strict function space $[D \to_\bot E]$ is the set $\{f : D \to E \mid f \text{ continuous and strict}\}$ equipped with the pointwise ordering.

2.4 Scott Domains

In our discussion of the function space of the previous section, there is an ominous omission of algebraicity. That is, we were missing the following theorem:

Theorem ?????. *If D and E are algebraic DCPOs, then $[D \to E]$ is algebraic.*

Indeed, this is not true, but on somewhat of a technicality. The following explanation is due to [AJ95]. First of all, let us consider what the compact elements in $[D \to E]$ should be. At the very least they should include the following elements.

Definition 2.4.1. Let D and E be DCPOs with $d \in D$ and $e \in E$. We define the *pair function* $\langle d; e \rangle : D \to E$ as

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

Note that this function is not necessarily continuous for arbitrary d and e. However, it is continuous if d is compact, and is moreover a compact element of $[D \to E]$ if e is compact.

Lemma 2.4.2. *Let D and E be DCPOs with $d \in \mathbf{K}D$ and $e \in \mathbf{K}E$. Then $\langle d; e \rangle \in \mathbf{K}[D \to E]$.*

Proof. First to see that $\langle d; e \rangle$ is continuous, consider a directed set $S \subseteq D$. If $\bigsqcup S \not\sqsupseteq d$ then for all $s \in S$, $s \not\sqsupseteq d$ so $\langle d; e \rangle (\bigsqcup S) = \bot$ and $\bigsqcup_{s \in S} \langle d; e \rangle (s) = \bigsqcup_{s \in S} \bot = \bot$. If $\bigsqcup S \sqsupseteq d$ then $\langle d; e \rangle (\bigsqcup S) = e$ and by compactness of d there is $s \in S$ s.t. $d \sqsubseteq s$. Therefore, $\langle d; e \rangle (s) = e$ so $\bigsqcup_{s \in S} \langle d; e \rangle (s) = \bigsqcup \{\bot, e\} = e$.

To see that $\langle d; e \rangle$ is compact, consider a directed subset $S \subseteq [D \to E]$, such that $\langle d; e \rangle \sqsubseteq \bigsqcup S$. Since least upper bounds in $[D \to E]$ are defined pointwise, we have that $\langle d; e \rangle (d) = e \sqsubseteq \bigsqcup \{g(d) \mid g \in S\}$. By the compactness of e, there is some g such that $e \sqsubseteq g(d)$. Now, we can show that this extends to $\langle d; e \rangle \sqsubseteq g$ as follows. Take an arbitrary $x \in D$. If $x \sqsupseteq d$, then $\langle d; e \rangle (x) = e \sqsubseteq g(d) \sqsubseteq g(x)$. Otherwise if $x \not\sqsupseteq d$, then $\langle d; e \rangle (x) = \bot \sqsubseteq g(x)$. □

Lemma 2.4.3. *Let D and E be DCPOs with $d \in \mathbf{K}D - \{\bot\}$ and $e \in \mathbf{K}E$. Then $\langle d; e \rangle \in \mathbf{K}[D \to_\bot E]$.*

Proof. The compact elements in $[D \to_\bot E]$ are exactly those that are also compact in $[D \to E]$, so this follows immediately from the previous lemma. □

Now, we can show that any continuous function $f : D \to E$ between algebraic DCPOs is the least upper bound of a bunch of compact pair functions. The role of the pair functions here is similar to how a set function is set-theoretically built up by a union of its input-output pairs.

Proposition 2.4.4. *Let D and E be algebraic DCPOs and $f : D \to E$ be a continuous function. Then f is the least upper bound of pair functions in the form $\langle d; e \rangle$ with d, e compact.*

If moreover f is strict, we can ensure $d \neq \bot$ in all the compact pair functions.

Proof. We show that for each $x \in D$ and $e \in \downarrow_{\mathbf{K}} f(x)$ there is a pair function mapping x to e that is below f. Indeed, since $x = \bigsqcup \downarrow_{\mathbf{K}} x$, by continuity of f we have that $f(x) = \bigsqcup_{d \in \downarrow_{\mathbf{K}} x} f(d)$. Therefore, by compactness of e there is some $d \in \downarrow_{\mathbf{K}} x$ such that $e \sqsubseteq f(d)$. With this, the pair function we are looking for can be given as $\langle d; e \rangle$. By the algebraicity of E, each $f(x)$ is equal to $\bigsqcup_{e \downarrow_{\mathbf{K}} f(x)} \langle d; e \rangle\ (x)$, and since joins are defined pointwise this suffices to show that we can approximate $f(x)$ by a set of pair functions.

If f is strict, but is above some $\langle d; e \rangle$ with $d \neq \bot$ then this almost immediately leads to a contradiction since it would mean $f(d) \sqsupseteq e \neq \bot$, unless e is also $\bot$. If e is also $\bot$, then $\langle d; e \rangle = \bot$ so can be safely omitted. □

However, the set of compact approximations of f is in general NOT directed, and this seems to be a fundamental limitation of our definition of algebraic DCPO. Even trying to find a compact upper bound of two pair functions is difficult, as we will now demonstrate.

Suppose $\langle d_1; e_1 \rangle$ and $\langle d_2; e_2 \rangle$ are compact pair functions as defined above, approximating a continuous function $f : D \to E$. We want to find a compact upper bound g of the two pair functions, such that $g \sqsubseteq f$ still. An obvious candidate then is to define

$$g(x) = \begin{cases} \bot & \text{if } x \not\sqsupseteq d_1 \text{ and } x \not\sqsupseteq d_2 \\ e_1 & \text{if } x \sqsupseteq d_1 \text{ and } x \not\sqsupseteq d_2 \\ e_2 & \text{if } x \not\sqsupseteq d_1 \text{ and } x \sqsupseteq d_2 \\ ??? & \text{if } x \sqsupseteq d_1 \text{ and } x \sqsupseteq d_2 \end{cases}$$

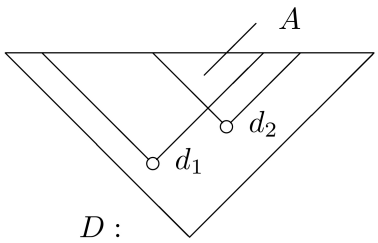


Image taken from [AJ95]

but then it is not clear how to define the case when $x \sqsupseteq d_1$ and $x \sqsupseteq d_2$ (i.e. when $x \in A$ in the figure to the right). It has to be an element above both e_1 and e_2, but it has to still approximate $f(x)$. Moreover, the requirement that g be compact means that this element itself has to be compact. An element that satisfies all of the above is $e_1 \sqcup e_2$, but of course this is not a directed join and so may not necessarily exist. However, its not such a conceptual difficulty to accept that two elements e_1 and e_2 with an upper bound (i.e. the information they carry are "consistent" with each other) should have a *least* upper bound (carrying just the combination of the information in e_1 and e_2)[13]. Therefore, we require our domains to satisfy this additional requirement.

[13] This is not so much a justification as it is a "why not?"

Definition 2.4.5. 1. A *Scott domain* or *bounded-complete (bc) domain* is an algebraic DCPO E such that for any two elements $e_1, e_2 \in E$ bounded above, the least upper bound $e_1 \sqcup e_2$ exists.

2. Let **Scott** denote the full subcategory of **DCPO** containing the Scott domains.

Theorem 2.4.6. *1. If D is an algebraic DCPO and E is a Scott domain then $[D \to E]$ and $[D \to_{\bot} E]$ are Scott domains.*

2. **Scott** *is a cartesian closed category.*

Proof. 2. follows immediately from 1, so we just prove 1.

(Bounded-completeness) Let $f, g : D \to E$ be continuous maps bounded above by $h : D \to E$. Then we have that $(f \sqcup g)(x) = f(x) \sqcup g(x)$ which exists since $f(x), g(x)$ are bounded above by $h(x)$. Additionally, note that $f \sqcup g$ is strict if f, g are strict.

(Algebraicity) Let g_1, g_2 be compact approximations of a continous map $f : D \to E$. We find a compact upper bound g of g_1 and g_2 that still approximates f, by defining $g = g_1 \sqcup g_2$. This shows $\downarrow_{\mathbf{K}} f$ is directed. To see that $f = \bigsqcup \downarrow_{\mathbf{K}} f$, recall that in Proposition 2.4.4 we already showed f to be the least upper bound of the compact pair functions approximating f, which are a subset of $\downarrow_{\mathbf{K}} f$. Again, we remark that the argument carries through if f is strict, since then g_1 and g_2 are strict so g is also strict. □

Finally, we can characterize the compact elements of the function space as those composed of a finite consistent set of compact pair functions, provided we are in the category of Scott domains.

Proposition 2.4.7. *Let D and E be Scott domains. Then*

$$\mathbf{K}[D \to E] = \{\bigsqcup_{i \in I} \langle d_i; e_i \rangle \mid I \text{ finite}, \forall i \in I.(d_i \in \mathbf{K}D \text{ and } e_i \in \mathbf{K}E), \{\langle d_i; e_i \rangle\}_{i \in I} \text{ is bounded above}\}$$

and

$$\mathbf{K}[D \to_\perp E] = \{\bigsqcup_{i \in I} \langle d_i; e_i \rangle \mid I \text{ finite}, \forall i \in I.(d_i \in \mathbf{K}D - \{\perp\} \text{ and } e_i \in \mathbf{K}E), \{\langle d_i; e_i \rangle\}_{i \in I} \text{ is bounded above}\}$$

Proof. (For $[D \to E]$) The right-to-left inclusion is clear, since any finite join of compact elements is still compact. For the left-to-right inclusion, take an arbitrary $f \in \mathbf{K}[D \to E]$. By Proposition 2.4.4 we can express f as a join of compact pair functions. Since we are now working with Scott domains, this join can be made directed by bounded-completeness, so the compactness of f implies f is a finite join of compact pair functions. Moreover, these compact pair functions are obviously bounded above by f.

(For $[D \to_\perp E]$) This follows immediately on account that these are exactly the strict functions that are also compact in $[D \to E]$. □

2.5 *Denotational Semantics of* **Imp**

With the domain theory we have developed so far, we are now in the position to define the semantics of **Imp**, the basic imperative language introduced in section 1.1. We define $\Sigma_{\mathbf{Imp}}$, or just Σ for short in this section, to be the domain $[\mathbb{N}_\perp \to_\perp \mathbb{Z}_\perp]$. Note that this isomorphic to $\mathbb{N} \rightharpoonup \mathbb{Z}$. Here, we are making the choice to use natural numbers to represent variable names. We will denote a variable as n, and an integer[14] as k.

[14] actual integers, not arithmetic expressions.

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

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

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

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

where the operations are defined as one would expect on non-$\perp$ elements, but as soon as any argument is $\perp$ it also returns $\perp$. The rationale here is that we want **Imp** to be a strict language, so as long as an argument doesn't terminate, the operation also shouldn't. For example, $[\![\texttt{and}]\!]$ is defined as:

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

For the commands, there is nothing surprising, except for the fact that we now have to account for the proliferation of $\perp$ elements everywhere, just as for the arithmetic and boolean expressions. Once again, we interpret the language strictly, so for anything involving boolean or arithmetic evaluation we output $\perp$ as long as the corresponding boolean or arithmetic expression is $\perp$.

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

$$
\begin{aligned}
[\![\texttt{skip}]\!] &= id_\Sigma \\
[\![C_1\,;C_2]\!] &= [\![C_2]\!] \circ [\![C_1]\!] \\
[\![\texttt{def } n \texttt{ := } A]\!](s) &= \begin{cases} \bot_\Sigma & \text{if } [\![A]\!]_s = \bot_{\mathbb{N}_\bot} \\ s[n := [\![A]\!]_s] & \text{otherwise} \end{cases} \\
[\![\texttt{if } B \texttt{ then } C_1 \texttt{ else } C_2]\!](s) &= \begin{cases} \bot & \text{if } [\![B]\!]_s = \bot_{2_\bot} \\ C_1(s) & \text{if } [\![B]\!]_s = tt \\ C_2(s) & \text{if } [\![B]\!]_s = ff \end{cases} \\
[\![\texttt{while } B \texttt{ do } C]\!] &= \bigsqcup_{n\in\omega} F^n_{B,C}(\bot)
\end{aligned}
$$

where $s[n := k] \in \Sigma$ denotes the state

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

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

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

2.6 *Solving Domain Equations*

Our treatment of domains so far has allowed us to model **Imp**. However, we still have not developed a way of solving domain equations in order to model the definition and reuse procedures. As suggested before, the equations can be computed in the same way that we computed the solution to an endofunction. To do this, we have to consider our categories of domains as a "big[15] DCPO"[16], consider what it means for an endofunctor to be "continuous", and then prove a fixpoint theorem for such continuous endofunctors. The developments in this section will follow that of [Plo83, Chapter 4 & 5].

As a first attempt, we can simply consider the usual category **DCPO** itself. However, there are some issues with this. The first issue is pragmatic. Recall that in the introduction we sought to find the least fixedpoint of a functor like so:

$$
F(D) := (\langle Var \rangle \to \mathbb{Z}) \times (\langle ProcName \rangle \to (D \rightharpoonup D))
$$

To boil it down and focus on the problem, let us try to find a fixpoint for

$$
F(D) := [D \to D]
$$

We would like to think of F as an endofunctor $F : \mathbf{DCPO} \to \mathbf{DCPO}$, but this is not possible since the functor $[- \to -]$ is contravariant on one argument and covariant in the other.

The second issue is conceptual. Simply considering the existence of any map to consist an information ordering is very weak. For example, every DCPO has a map to the unique one-element DCPO $\emptyset_\bot$,

[15] in the sense that the DCPO will not have an underlying set, but rather an underlying class.

[16] Actually, since we are only concerned with solving fixpoints here, we can consider the category of domains as an ω-CPO instead, i.e. posets with upper bounds of any ω-chains.

but we certainly should not consider them as containing less information than $\emptyset_\perp$. Instead, let us consider from first principles what it means for one DCPO to approximate another.

Suppose we have DCPOs D and E that models a particular space of programs, in such a way that the modelling is not superfluous, i.e. each element of D models some program p, and similarly for E. Then we would consider E to be better at modelling this space than D if for every $e \in E$ modelling p, the corresponding $d \in D$ that also models p satisfies $d \sqsubseteq e$. Of course, such a comparison is ill-defined because d and e live in different DCPOs. Therefore, we have to consider a comparison map $i : D \to E$ that embeds D into E. Moreover, for every $e \in E$ modelling p, we should also be able to find the d modelling p such that $i(d) \sqsubseteq e$. Therefore, we ask for a map $j : E \to D$ backwards as well, such that $ji(d) = d$.

Definition 2.6.1. 1. Let D and E be DCPOs. An *embedding-projection pair* is a pair of continuous maps $\langle i : D \to E, j : E \to D \rangle$ i, satisfying

$$ji = id_D \text{ and } ij \sqsubseteq id_E.$$

2. We say that i is an *embedding* if there is a corresponding j such that $\langle i, j \rangle$ is an embedding-projection pair. In similar fashion we say that j is a *projection* if there is a corresponding i making $\langle i, j \rangle$ into an embedding-projection pair. Notationally, we will use $i : D \lhd E$ to denote that i is an embedding from D into E, and $j : E \rhd D$ to denote that j is a projection.

3. Let $\mathbf{DCPO}^E$ denote the category whose objects are DCPOs, but the maps are embedding-projection pairs[17].

[17] the direction being the direction of the embedding, not the projection

The following lemma shows that in fact $\mathbf{DCPO}^E$ can equivalently be considered as the subcategory consisting of all DCPOs except the maps are just the embeddings, and that $\mathbf{DCPO}^{E^{op}}$ can be considered the category where the maps are projections.

Lemma 2.6.2. *Given an embedding $i : D \lhd E$, there is a unique continuous map $i^R : E \to D$ such that $\langle i, i^R \rangle$ forms an embedding-projection pair. Similarly, any projection $j : E \rhd D$ is associated with a unique embedding j^L.*

Proof. Let j and j' be two maps that form an embedding projection with i. We have $ij \sqsubseteq id_E$ which implies $j = j'ij \sqsubseteq j'$. Symmetrically we have that $j' \sqsubseteq j$, so $j = j'$. The proof for unique embedding is analogous by considering $i = iji' \sqsubseteq i'$. □

This also solves our pragmatic problem on the matter of mixed variance, since to turn a contravariant functor on **DCPO** into a covariant one in $\mathbf{DCPO}^E$, we just make it act on the projection instead!

Definition 2.6.3. 1. A functor $F : \mathbf{DCPO}^{opm} \times \mathbf{DCPO}^n \to \mathbf{DCPO}$ is *locally monotonic* if for any family of morphisms $f_i \sqsubseteq f_i' : D_i' \to D_i$ with $i < m$ and $g_i \sqsubseteq g_i' : E_i \to E_i'$ with $i < n$ we have that

$$F(f_0 \dots f_{m-1}, g_0 \dots g_{n-1}) \sqsubseteq F(f_0' \dots f_{m-1}', g_0' \dots g_{n-1}')$$

Lemma 2.6.4. *Let $F : \mathbf{DCPO}^{op^m} \times \mathbf{DCPO}^n \to \mathbf{DCPO}$ be a locally monotonic functor. Then there is a monotone functor $F^E : \mathbf{DCPO}^{E^m} \times \mathbf{DCPO}^{E^n} \to \mathbf{DCPO}^E$ defined on objects in the same way as F, and on morphisms by*

$$F^E(i_0, \ldots, i_{m-1}, j_0, \ldots, j_{n-1}) = F(i_0^R, \ldots, i_{m-1}^R, j_0, \ldots, j_{n-1})$$

Proof. The functoriality of F^E follows from functoriality of F, and the fact that $(ii')^R = i'^R i^R$ and $id^R = id$. Moreover, we need to make sure that $F^E(i_0, \ldots, i_{m-1}, j_0, \ldots, j_{n-1})$ is also an embedding. We claim that the projection is $F(i_0, \ldots, i_{m-1}, j_0^R, \ldots, j_{n-1}^R)$.

$$\begin{aligned} F(\overline{i^R}, \bar{j}) \circ F(\bar{i}, \overline{j^R}) &= F(\overline{i \circ i^R}, \overline{j \circ j^R}) && \text{By functoriality} \\ &\sqsubseteq F(\overline{id}, \overline{id}) && \text{By local mononicity} \\ &= id && \text{By functoriality} \end{aligned}$$

$$\begin{aligned} F(\bar{i}, \overline{j^R}) \circ F(\overline{i^R}, \bar{j}) &= F(\overline{i^R \circ i}, \overline{j^R \circ j}) && \text{By functoriality} \\ &= F(\overline{id}, \overline{id}) \\ &= id && \text{By functoriality} \end{aligned}$$

□

Example 2.6.5. *From the functor $[- \to -]$ we obtain a functor $[- \to -]^E : \mathbf{DCPO}^E \times \mathbf{DCPO}^E \to \mathbf{DCPO}^E$ defined on morphisms by*

$$[(i : D \lhd D') \to (j : E \lhd E')]^E(h : D \lhd E) = [i^R \to j](h) = j \circ h \circ i^R.$$

Next, we have to consider what a limit of a sequence in $\mathbf{DCPO}^E$ is. The least upper bound in a poset is categorically a colimit. Following this, we define the limit as the colimit of the sequence. Let us see what this means explicitly.

Lemma 2.6.6. *Let $I : \omega \to \mathbf{DCPO}^E$ be a diagram. Then the colimiting cocone of the induced diagram $I^* : \omega \xrightarrow{I} \mathbf{DCPO}^E \hookrightarrow \mathbf{DCPO}$ always exists, and is also a colimiting cocone of I.*

Proof. Let I_n denote the n-th DCPO in the diagram, and for $n \leq m$ let $I_{n,m} : I_n \lhd I_m$ denote the embedding from the n-th DCPO to the m-th.

We define the cocone vertex of I^* as the sub-DCPO D of $\prod_{n \in \omega} I_n$ containing

$$\left\{ d \in \prod_{n \in \omega} I_n \mid \forall i \in \omega . d_i = I_{i,i+1}^R(d_{i+1}) \right\}$$

The cocone maps are defined by

$$\rho_n : I_n \to D$$

$$d_n \mapsto (I_{0,n}^R(d_n), I_{1,n}^R(d_n), \ldots, d_n, I_{n,n+1}(d_n), I_{n,n+2}(d_n), \ldots)$$

It is routine to verify that this is an embedding with projection $d \mapsto d_n$, and commutes with the $I_{n,m}$ maps making it a cocone.

To see that this is indeed the colimit of I^*, consider an arbitrary cocone $(E, (\kappa_n)_{n\in\omega})$. Then we can define a map $f : D \to E$ given by

$$d \mapsto \bigsqcup_{n\in\omega} \kappa_n(d_n)$$

Indeed, this definition is forced since we can show that $d = \bigsqcup_{n\in\omega} \rho_n(d_n)$. Since f has to preserve directed joins and behave well with respect to κ, we must have that $f(d) = \bigsqcup_{n\in\omega} f(\rho_n(d_n)) = \bigsqcup_{n\in\omega} \kappa_n(d_n)$.

Moreover, it is clear that this cocone is also the colimit of I, since the cocone maps are embeddings and therefore exist inside the subcategory $\mathbf{DCPO}^E$ where it inherits the universal property. □

It is then clear what a continuous endofunctor is in this context.

Definition 2.6.7. A functor $F : (\mathbf{DCPO}^E)^n \to \mathbf{DCPO}^E$ is *continuous* if F preserves ω-colimits.

Analogous to the definition of locally monotone, we can formulate a definition of "locally continuous" for functors on **DCPO** that ensures they can be lifted to a continuous functor on $\mathbf{DCPO}^E$.

Definition 2.6.8. A functor $F : (\mathbf{DCPO}^{op})^m \times \mathbf{DCPO}^n \to \mathbf{DCPO}$ is *locally continuous* if given families of morphisms

- $(f_i^k) : D_i' \to D_i$ with $k \in \omega$ and $i < m$ such that $f_i^k \sqsubseteq f_i^{k+1}$
- $(g_j^k) : E_j \to E_j'$ with $k \in \omega$ and $j < n$ such that $g_j^k \sqsubseteq g_j^{k+1}$

we have that $F(f_0^k, \ldots, f_{m-1}^k, g_0^k, \ldots, g_{n-1}^k) \sqsubseteq F(f_0^{k+1}, \ldots, f_{m-1}^{k+1}, g_0^{k+1}, \ldots, g_{n-1}^{k+1})$ and

$$F(\bigsqcup_{k\in\omega} f_0^k, \ldots, \bigsqcup_{k\in\omega} f_{m-1}^k, \bigsqcup_{k\in\omega} g_0^k, \ldots, \bigsqcup_{k\in\omega} f_{n-1}^k) = \bigsqcup_{k\in\omega} F(f_0^k, \ldots, f_{m-1}^k, g_0^k, \ldots, g_{n-1}^k)$$

Lemma 2.6.9. *If $F : (\mathbf{DCPO}^{op})^m \times \mathbf{DCPO}^n \to \mathbf{DCPO}$ is a locally continuous functor then $F^E : (\mathbf{DCPO}^E)^{m+n} \to \mathbf{DCPO}$ is a continuous functor.*

Proof. Notice that local continuity of f implies local monotonicity, so F^E is a well-defined functor. Consider an ω-diagram $D : \omega \to (\mathbf{DCPO}^E)^{m+n}$, and the standard colimit D' of this diagram, as defined in Lemma 2.6.6. This colimit has cocone maps $\rho_k : D(k) \to D'$.

We need to show the canonical embedding h from the standard colimit E' of $F^E \circ D$ to $F^E(D')$ is an isomorphism. Unfolding the definition of E' and h, for every $d \in F(D')$ we need to find a unique $e \in E'$ such that

$$d = h(e) := \bigsqcup_{k\in\omega} F(\rho_k)(e_k)$$

We take $e_k := F(\rho_k)^R(d)$, and have that

$$\begin{aligned} h(e) &= \bigsqcup_{k\in\omega} F(\rho_k)F(\rho_k)^R(d) \\ &= \bigsqcup_{k\in\omega} F(\rho_k\rho_k^R)(d) \\ &= F(\bigsqcup_{k\in\omega} \rho_k\rho_k^R)(d) && \text{By local continuity} \\ &= F(id)(d) = d && \text{since } \bigsqcup_{k\in\omega} \rho_k\rho_k^R = id \end{aligned}$$

□

Example 2.6.10. *$[- \to -]$ is locally continuous so $[- \to -]$ is a continuous functor.*

Theorem 2.6.11. *Let $F : \mathbf{DCPO}^E \to \mathbf{DCPO}^E$ be a continuous functor. Then there is a DCPO D such that $D \cong F(D)$.*

Proof. D can be obtained as the colimit of the ω-diagram

$$\emptyset_\perp \overset{!}{\lhd} F(\emptyset_\perp) \overset{F(!)}{\lhd} F^2(\emptyset_\perp) \dots$$

□

In fact, we can generalize this theorem further to the solution of a system of simultaneous equations:

Theorem 2.6.12. *Let $F_{i<n} : (\mathbf{DCPO}^E)^n \to \mathbf{DCPO}^E$ be an n-family of continuous functors. Then there is an n-family of DCPOs such that*

Proof. (sketch) We can consider the F_i as a single functor $F : (\mathbf{DCPO}^E)^n \to (\mathbf{DCPO}^E)^n$ and compute the least fixed point of F. This requires appropriate generalizations of the previous concepts, but there is nothing new conceptually. □

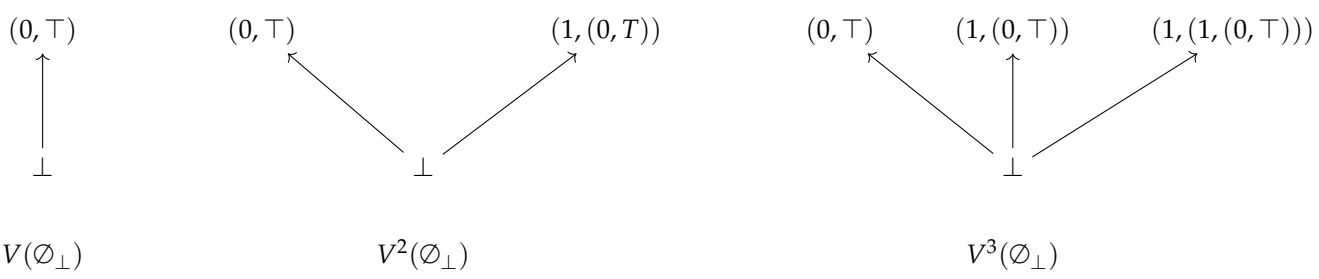
2.7 *Denotational Semantics of* **ImpProc**

With all the constructions we have defined so far, we can finally give a proper denotational semantics of the **ImpProc** language we defined in the introduction. In fact we can construct the whole semantics from basic finite domains $\emptyset_\perp$ and $1_\perp$ and using domain equations over $[- \to -]$ and $- \times -$. This will come in handy in the final chapter when we construct the logic for reasoning about **ImpProc**, since we will describe the logic of $[D \to E]$ inductively in terms of the logic for D and E, and so on.

To begin with, we can represent the set of variables and and procedure names as the flat domain $\mathbb{N}_\perp$. This can be constructed as the fixed point of the functor

$$V(X) := 1_\perp \oplus X$$

To illustrate why this is, we see that $V^k(\_\perp)$ is the flat domain with k elements:



$V(\emptyset_\perp)$ $V^2(\emptyset_\perp)$ $V^3(\emptyset_\perp)$

In the limit, we obtain the flat domain $\mathbb{N}_\perp$. We can also obtain $\mathbb{Z}_\perp$ as $\mathbb{N}_\perp \oplus \mathbb{N}_\perp$.

Next, we have to construct the state space $\Sigma_{\mathbf{ImpProc}}$. For brevity, we refer to $\Sigma_{\mathbf{ImpProc}}$ as Σ in this section. First, we separate Σ into the part that keeps track of variables ($\mathsf{VEnv} := \Sigma_{\mathbf{Imp}}$) and the part that keeps track of procedures (PEnv).

$$\Sigma = \mathsf{VEnv} \times \mathsf{PEnv}$$

The variable structure does not need a domain equation to define, since it is just $\Sigma_{\mathbf{Imp}}$:

$$\mathsf{VEnv} := [\mathbb{N}_\perp \to_\perp \mathbb{Z}_\perp]$$

To define Σ, we solve for the single domain equation

$$\Sigma = \mathsf{VEnv} \times [\mathbb{N}_\perp \to_\perp [\Sigma \to \Sigma]]$$

and define

$$\mathsf{Proc} := [\Sigma \to \Sigma]$$

$$\mathsf{PEnv} := [\mathbb{N}_\perp \to_\perp \mathsf{Proc}]$$

Therefore, we have an isomorphism

$$\alpha : \Sigma \xrightarrow{\cong} \mathsf{VEnv} \times \mathsf{PEnv}$$

With this, we can finally define the semantics.

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

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

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

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

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

$$
\begin{aligned}
[\![\texttt{skip}]\!] &= id_\Sigma \\
[\![C_1\texttt{;}C_2]\!] &= [\![C_2]\!] \circ [\![C_1]\!] \\
[\![\texttt{def } n \texttt{ := } A]\!](s) &= \begin{cases} \bot_\Sigma & \text{if } [\![A]\!]_s = \bot_{\mathbb{N}_\bot} \\ s[n := [\![A]\!]_s] & \text{otherwise} \end{cases} \\
[\![\texttt{if } B \texttt{ then } C_1 \texttt{ else } C_2]\!](s) &= \begin{cases} \bot & \text{if } [\![B]\!]_s = \bot_{2_\bot} \\ C_1(s) & \text{if } [\![B]\!]_s = tt \\ C_2(s) & \text{if } [\![B]\!]_s = ff \end{cases} \\
[\![\texttt{while } B \texttt{ do } C]\!] &= \bigsqcup_{n\in\omega} F^n_{B,C}(\bot) \\
[\![\texttt{def } n \texttt{ := } C]\!](s) &= s[n := [\![C]\!]] \\
[\![\texttt{run } n]\!] &= \gamma(\pi_{\mathsf{PEnv}}(\alpha(s)))(n)
\end{aligned}
$$

where $s[n := k] \in \Sigma$ denotes the state

$$
\left(x \mapsto \begin{cases} k & \text{if } x = n \\ \pi_{\mathsf{VEnv}}(\alpha(s))(x) & \text{if } x \neq n \end{cases}, \pi_{\mathsf{PEnv}}(\alpha(s)) \right)
$$

and similarly $s[n := [\![C]\!]]$ denotes the state

$$
\alpha^{-1}\left(\pi_{\mathsf{VEnv}}(\alpha(s)), x \mapsto \begin{cases} [\![C]\!] & \text{if } x = n \\ \pi_{\mathsf{PEnv}}(\alpha(s))(x) & \text{if } x \neq n \end{cases} \right)
$$

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

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

Notice the difference between the semantics of defining an integer variable and defining a process variable. The former models how we eagerly compute the assigned integer upon assignment, while the latter models how we lazily only compute the assigned command when we run it.

3
Duality Theory

In this chapter, we introduce Stone duality, which provides a correspondence between logics (represented as categories of algebraic structures) on one hand, and their spaces of models on the other. Normally, you would have a good idea of what your logic is, and therefore apply Stone duality to find a space of models. With domains, we work in reverse: we have a good intuitive understanding of domains as a space, and so we apply Stone duality to find logics whose models are elements of the domain. In other words, we use Stone duality to find a logic for reasoning about programs.

In section 3.1, we begin by introducing the intuitive understanding of domains as topological spaces. We develop some basic aspects of the spatial structure, which will be useful later. Then, section 3.2 and section 3.3 introduces the basic form of Stone duality.

The remaining sections then sharpen this duality. section 3.5 makes the observation that while DCPOs in general do not fit under the framework of Stone duality, algebraic DCPOs do fit. We can therefore sharpen the duality to discover the algebras/logic corresponding to algebraic DCPOs.

In section 3.6, we impose a restriction on the logic - it has to be representable using only finite logical operators[1]. We sharpen the Stone duality to discover what conditions this corresponds to on the spatial side, and observe that both Scott domains and bifinite domains fit these conditions, so it is still applicable.

In the next chapter, we put together these dualities to construct a logic for reasoning about programs.

[1] as we will see, this is requirement corresponds to algebraicity as well, but this time on the algebra side instead of the spatial/DCPO side

3.1 Topologizing Domains

The order relation in a DCPO is understood as being an information ordering: $x \sqsubseteq y$ means that any property true of x is also true of y. Now, mathematically we can identify a property with the set of elements satisfying that property[2]. But that means a property is not just any set: by our interpretation of $\sqsubseteq$, a proposition has to be an upwards-closed set with respect to this ordering.

[2] That is, we take an extensional view of properties.

Moreover, for programs, we care only about the finitely observable properties: if a computation $\bigsqcup A$ satisfies a finitely observable property P, then it should be satisfied at some finite stage $a \in A$ of the

computation. Mathematically, we say such a property is *inaccessible by directed joins*.

The finitely observable properties should be closed under arbitrary disjunction: if we can finitely observe one of them then we can finitely observe all of them. Moreover, they should be closed under finite conjunction: we can finitely observe finitely many such properties, but not if there are infinitely many. Finally, the property that is always true and never true should be trivially observable. We capture this idea with the notion of a topology.

Definition 3.1.1. A *topological space* is a set X equipped with a set of subsets ΩX, called the *topology* of X. The elements of ΩX are called the *open* subsets of the space. A subset $S \subseteq X$ is *closed* iff $X - S$ is open. S is *clopen* iff it is both closed and open.

A function $f : X \to Y$ between two topological spaces are *continuous* if for any open set $U \subseteq Y$, $f^{-1}[U]$ is also an open set[3].

[3] The intuition here being that if I observe a property U of $y \in Y$, I want to observe a corresponding property on the elements of $f^{-1}(y)$.

Let **Top** denote the category of topological spaces and continuous functions.

The collection of upwards-closed sets in a DCPO that are inaccessible by directed joins are closed under finite intersection and arbitrary union, so they form a topology called the Scott topology.

Definition 3.1.2. Let $(D, \sqsubseteq)$ be a DCPO. Then a set U is *Scott-open* if it satisfies the following two properties:

1. upwards closed: if $x \sqsubseteq y$ and $x \in U$, then $y \in U$.
2. inaccessible by directed joins: for all directed sets A, if $\bigsqcup A \in U$ then $A \cap U$ is non-empty.

The Scott-open sets form the *Scott topology* ΣD, although we will also often refer to this topology by ΩD when there is no ambiguity.

The condition of inacessibility by directed joins presents in a simpler way in terms of Scott-closed sets.

Proposition 3.1.3. *Given a DCPO D, a subset $S \subseteq D$ is closed iff it is*

1. *downwards closed: if $x \sqsubseteq y$ and $y \in S$, then $x \in S$.*
2. *closed under directed joins: for all directed sets $A \subseteq S$, $\bigsqcup A \in S$.*

As a first exercise, we observe that the intuitive meaning of $\sqsubseteq$ holds: the specialization ordering induced by ΣD corresponds to the order on D.

Definition 3.1.4. Given a topological space X, define the *specialization ordering* to be $x \leq_X y \iff (\forall U \text{ open}. x \in U \implies y \in U)$

Proposition 3.1.5. *Let $(D, \sqsubseteq)$ be a DCPO. Then $x \leq_{\Sigma D} y \iff x \sqsubseteq y$.*

Proof. ($\Longleftarrow$) Obvious, since every Scott-open set is upwards closed w.r.t. $\sqsubseteq$.

($\Longrightarrow$) Suppose $x \leq_{\Sigma D} y$. Consider the set of elements that are **not**[4] below y:

[4] using the set of elements **not** below some element y is generally a good tactic when proving things about the Scott topology, since the usual candidate $\uparrow y$ is not necessarily Scott-open.

$$U_y = \{z \in D \mid z \not\sqsubseteq y\}.$$

Observe that it is Scott-open and that $y \notin U_y$, so by applying $x \leq_{\Sigma D} y$, we have also that $x \notin U_y$. Therefore, $x \sqsubseteq y$. □

The choice of the Scott topology is also justified in terms of morphisms, for the topologically continuous functions between Scott topologies are exactly those that preserve directed joins.

Proposition 3.1.6. *Let $f : D \to E$ be a function between DCPOs D, E. Then f is continuous in the domain-theoretic sense (Definition 2.1.7) iff it is a topologically continuous function between the corresponding Scott topologies.*

Proof. ($\Longrightarrow$) Let S be a closed set in E. Then $f^{-1}[S]$ is downwards closed in D, for whenever $x \sqsubseteq y \in f^{-1}[S]$, then $f(x) \sqsubseteq f(y) \in S$, so $f(x) \in S$ which means $x \in f^{-1}[S]$. By a similar reasoning using f's preservation of joins, we can show S is closed under directed joins.

($\Longleftarrow$) For monotonicity, suppose $x \sqsubseteq y$ in D. We need to show $f(x) \sqsubseteq f(y)$ in E, so consider the closed set $\downarrow f(y)$ in E. Then by continuity of f, $f^{-1}[\downarrow f(y)]$ is also closed and therefore downward closed. Since $y \in f^{-1}[\downarrow f(y)]$, so is x, and so $f(x) \in \downarrow f(y)$ which means $f(x) \sqsubseteq f(y)$.

For preservation of joins, let $A \subseteq D$ be a directed set. Clearly, since f is monotone and $\bigsqcup A \sqsubseteq a$ for all $a \in A$, we have $f(\bigsqcup A) \sqsubseteq f(a)$ for all $a \in A$. Hence, by the universal property of the join we have $\bigsqcup f[A] \sqsubseteq f(\bigsqcup A)$. For the other direction, Consider the closed set $S = \downarrow \bigsqcup f[A]$ in E, whose inverse image will also be closed by continuity of f. Then $f[A]$ is a subset of S, so A is a subset of the inverse image. Since the inverse image is closed, it is closed under directed joins, so $\bigsqcup A$ is in the inverse image, meaning $f(\bigsqcup A) \in S$. Hence, $f(\bigsqcup A) \sqsubseteq \bigsqcup f[A]$. □

For algebraic DCPOs, the Scott topology has an alternate definition, which even more closely resembles our intuition.

Proposition 3.1.7. *Let $(D, \sqsubseteq)$ be an algebraic DCPO. Then U is Scott-open iff it is upwards-closed and whenever $x \in U$, there is a compact element $x' \in \downarrow_{\mathbf{K}} x$ such that $x' \in U$.*

Proof. Clearly, if U is inaccesible by directed joins this will imply the above property for an algebraic DCPO, since $x = \bigsqcup(\downarrow_{\mathbf{K}} x)$. On the other hand, if U has the above property, and $x = \bigsqcup S \in U$ for some directed set S. Then, there is some $x' \in \downarrow_{\mathbf{K}} x$ such that $x' \in U$. Since $x' \sqsubseteq x = \bigsqcup S$, by compactness, there is $y \in S$ such that $x' \sqsubseteq y$. Then by upwards closure of U, we have $y \in U$. □

3.2 *Topological Spaces & Locales*

The core idea of Stone duality is that there is a correspondence between certain spaces and certain logical algebraic structures. Indeed, from a topological space there is one such algebraic structure we can

extract, which consist of the open sets of the space. The operations of this structure is that of infinite joins and finite meets, and we axiomatise this structure by the notion of frame. While we are defining frames, we take a moment to also introduce related structures.

Definition 3.2.1. A poset $(P, \leq)$ is a *lattice* iff any two elements has both a meet and a join. A function between two lattices is a *lattice homomorphism* iff it preserves all finite meets and joins.

We can also define a lattice $(L, \wedge, \vee)$ by the following axioms.

$(Commutativity)$	$a \vee b = b \vee a$	$a \wedge b = b \wedge a$
$(Associativity)$	$a \vee (b \vee c) = (a \vee b) \vee c$	$a \wedge (b \wedge c) = (a \wedge b) \wedge c$
$(Idempotence)$	$a \vee a = a$	$a \wedge a = a$
$(Absorption)$	$a = a \vee (a \wedge b)$	$a = a \wedge (a \vee b)$

A lattice $(L, \wedge, \vee)$ is *distributive* if the following additional axioms are satisfied:

$$a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c)$$
$$a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$$

$(L, \wedge, \vee)$ is *bounded* iff there are elements $0, 1 \in L$ such that $0 \leq a \leq 1$ for all $a \in L$. These can be thought of respectively as the join and meet of the empty set.

Definition 3.2.2. Let P be a poset. For $a \in P$ and $S \in P$, we write

$$\uparrow S = \{x \in P \mid \exists y \in S . x \geq y\}$$
$$\uparrow a = \uparrow \{a\}$$
$$\downarrow S = \{x \in P \mid \exists y \in S . x \geq y\}$$
$$\downarrow a = \downarrow \{a\}$$

Definition 3.2.3. A bounded lattice is a *frame* iff additionally every subset has a join and binary meets distribute over arbitrary joins:

$$x \wedge (\bigvee Y) = \bigvee \{x \wedge y \mid y \in Y\}$$

A function between two frames is a *frame homomorphism* iff it preserves arbitrary joins and finite meets. We denote the category of frames and frame homomorphisms as **Frm**. A *subframe* is a subset Y of a frame X s.t. the inclusion map $Y \hookrightarrow X$ is a frame homomorphism. More explicitly, Y is a subframe of X if it satisfies the conditions:

- If $S \subseteq Y$ then $\bigvee S \in Y$.
- If $S \subseteq Y$ finite then $\bigwedge S \in Y$.

We may view frames as an abstraction of topological spaces, considering only the order-structure of the observations/properties (opens) and forgetting the things (points) being observed. There is a slight subtlety involved in this view though. The natural notion of morphism between topological spaces is a continuous map $f : (X, \Omega X) \to$

$(Y, \Omega Y)$, which maps an open set U in Y to an open set $f^{-1}[U]$ in X. In view of this, it is the inverse map $f^{-1} : \Omega Y \to \Omega X$ that acts as a frame homomorphism[5] between ΩX and ΩY. Hence, when we consider frames as abstractions of spaces, we call them locales[6], and a frame homomorphism $A \to B$ is instead considered as a "continuous map" $B \to A$ between locales.

[5] as opposed to the direct image map $f[-] : \Omega X \to \Omega Y$.

[6] locale is a fancy word for space, so frames/locales are fancy spaces.

Definition 3.2.4. The category of locales **Loc** is the opposite category of **Frm**. Thus a locale morphism $A \to B$ is in fact a frame homomorphism $A \to B$.

With this definition, we can properly express this view of locales as "pointless" spaces by the localification functor $\Omega(-) : (X, \Omega X) \mapsto \Omega X$ forgetting the points of a topological space X and giving back just the frame.

Definition 3.2.5. The functor $\Omega(-) : \mathbf{Top} \to \mathbf{Loc}$ maps topological spaces $(X, \Omega X)$ to its frame of opens ΩX, and maps continuous function $f : X \to Y$ to its inverse image function $f^{-1}[-] : \Omega Y \to \Omega X$, which is a locale morphism $\Omega X \to \Omega Y$.

[7] in the sense that each property is determined by the set of points they contain.

For topological spaces, we started by specifying what the points are, and essentially enforced the frame structure to be extensional[7] over these sets of points. For locales, the frame structure comes first and we do not know what points we were originally observing - is there some way of "reconstructing" the points of the space? We can make a best guess by saying that there is exactly one point corresponding to each "complete" set F of observations[8]. Here, "complete" means that it completely describes the space of observations on a single point[9].

[8] after all, if there was no such point, how did we make the observation in the first place? So this is a kind of conservative estimate: we put in only the points which have to exist.

[9] A non-example would be the set $\{1\}$ consisting of the empty observation, which is observable on any point so it does not adequately describe a single point.

What exactly does it mean for a set of properties to "completely" describe a single point? Observe that the set must respect the frame ordering: if $a \leq b$ and the point observes a then it must also observe b. Hence, the "complete" sets have to be upwards closed. Moreover, if the point observes both a and b, then it must also observe $a \wedge b$. Finally, if the point observes $\bigvee_{i \in I} a_i$ then this can only be because the point observes some a_i.

Definition 3.2.6. Let A be a frame. A *filter* $\emptyset \subsetneq F \subseteq A$ is an upwards closed set satisfying the following property: if $a, b \in F$ then $a \wedge b \in F$.

The filter F is *proper* if $F \subsetneq A$[10], and it is *completely prime* if, in addition to being proper, it has the following property: $\bigvee_{i \in I} a_i \in F$ implies $a_i \in F$ for some a_i.

[10] or equivalently if $0 \notin F$.

The set of completely prime filters of A is denoted by $\mathbf{pt}(A)$, the *set of points of* A.

A more succinct way of describing $\mathbf{pt}(A)$ is as the set of frame homomorphisms $A \to 2$[11]. A completely prime filter F induces a homomorphism $x_F : A \to 2$, defined as the characteristic map of F. On the other hand, the kernel $x^{-1}(1)$ of any frame homomorphism $x : A \to 2$ is a completely prime filter.

[11] Note that this is a locale morphism $2 \to A$, and that $\Omega 1 \cong 2$ where 1 refers to the topological space with a single element. Points of a topological space X correspond to maps $1 \to X$, so it makes sense to do something similar with locales.

Proposition 3.2.7. *Given any frame A, we have the isomorphism $\mathbf{pt}(A) \cong A \to 2$.*

For any such $x : A \to 2$, we can equivalently look at the complement of $x^{-1}(1)$, namely $x^{-1}(0)$. That is, we look at the set of properties that are *not* satisfied by the point. If $a \leq b$ and the point does not observe b then it also cannot observe a. If does not satisfy $a \wedge b$, then either it doesn't satisfy a or it doesn't satisfy b. Finally, if the point does not satisfy any of a set of properties S, then it also cannot satisfy $\bigvee S$. But note that if we take $S = x^{-1}(0)$, then the single element $\bigvee x^{-1}(0)$ in fact encapsulates all the information about $x^{-1}(0)$.

Definition 3.2.8. Let A be a frame. An *ideal* $\emptyset \subsetneq I \subseteq A$ is a downwards closed set satisfying the following property: if $a, b \in I$ then $a \vee b \in I$. A *principal ideal* is one that is generated by some $a \in A$ as $\downarrow a$, and if the resulting ideal is also prime then we say a is *prime*.

So $x^{-1}(0)$ is a prime ideal, and the above analysis shows that it is principally generated by $\bigvee x^{-1}(0)$. We therefore have also a correspondence between the prime elements of A and $\mathbf{pt}(A)$. We will switch around between these three different perspectives on points depending on the situation.

Having "reconstructed" the set of points, we now seek to re-associate them to the properties and construct a topology. The natural thing to do is to associate each observable property $a \in A$ with the set of points observing a[12]:

$$\varepsilon_A(a) := \{x \in \mathbf{pt}(A) \mid a \in x\}$$

[12] One can verify that this mapping forms a frame homomorphism.

Proposition 3.2.9. *Let A be a locale. Then $(\mathbf{pt}(A), \Omega\mathbf{pt}(A))$ forms a topology, where $\Omega\mathbf{pt}(A)$ is the image of ε_A. Moreover, this mapping can be lifted to a functor $\mathbf{pt}(-) : \mathbf{Loc} \to \mathbf{Top}$ mapping each frame homomorphism $f : B \to A$ to $f^{-1}[-] : \Omega\mathbf{pt}(A) \to \Omega\mathbf{pt}(B)$.*

We may also start with a topological space X, forget its points leaving us with only ΩX, and then reconstruct the points as $\mathbf{pt}(\Omega X)$. Analogous to ε, we may now ask how to associate the original points to the reconstructed points. The natural thing to do is to associate each original point $x \in X$ with the point corresponding to the completely prime filter of properties observed by x:

$$\eta_X(x) := \{a \in \Omega X \mid x \in a\}$$

Even though these spatialisation/localification operations are not perfect (as we will soon see), they may be regarded as computing the best space/locale approximating the given locale/space, i.e. spatialisation and localification forms an adjunction.

Theorem 3.2.10 ([Joh82, subsection II.1.4])**.** *The functor $\mathbf{pt}(-)$ is right adjoint to $\Omega(-)$.*

Proof. We directly exhibit the isomorphism

$$\mathrm{Hom}_{\mathbf{Loc}}(\Omega X, A) \cong \mathrm{Hom}_{\mathbf{Frm}}(A, \Omega X) \cong \mathrm{Hom}_{\mathbf{Top}}(X, \mathbf{pt}(A)).$$

Given a continuous map $f : X \to \mathbf{pt}(A)$, define its transpose $\lceil f \rceil : A \to \Omega X$ to be $\lceil f \rceil (a) := f^{-1}[\varepsilon_A(a)]$. On the other hand, given a frame homomorphism $f : A \to \Omega X$, define its transpose $\lfloor h \rfloor : X \to \mathbf{pt}(A)$ to be $\lfloor h \rfloor (x) := h^{-1}[\eta_X(x)]$.

□

$\lceil - \rceil$ and $\lfloor - \rfloor$ are inverses:

$$\begin{aligned}\lfloor \lceil f \rceil \rfloor (x) &= \lceil f \rceil^{-1}[\eta_X(x)] \\ &= \{a \in A \mid x \in \lceil f \rceil (a)\} \\ &= \{a \in A \mid x \in f^{-1}[\varepsilon_A(a)]\} \\ &= \{a \in A \mid a \in f(x)\} \\ &= f(x)\end{aligned}$$

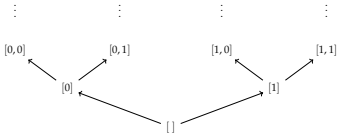
$$\begin{aligned}\lceil \lfloor h \rfloor \rceil (a) &= \lfloor h \rfloor^{-1}[\varepsilon_A(a)] \\ &= \{x \in X \mid a \in \lfloor h \rfloor (x)\} \\ &= \{x \in X \mid a \in h^{-1}[\eta_X(x)]\} \\ &= \{x \in X \mid x \in h(a)\} \\ &= h(a)\end{aligned}$$

3.3 Sober Spaces & Spatial Locales

Going back and forth by spatialising and then localifying (and vice versa), one does not always recover the same structure, i.e. these operations do not preserve all information associated with the input structure. The following example shows two topological spaces X for which $\mathbf{pt}(\Omega X)$ is not the homeomorphic to X.

Example 3.3.1. *Consider the space $X = \{1, 2, 3\}$ equipped with the topology $\Omega X = \{\emptyset, X\}$. Then $\eta(1) = \eta(2) = \eta(3) = \{X\}$, and in fact this is the only point in $\mathbf{pt}(\Omega X)$, so we have collapsed three points to the same point because we did not have enough open sets to distinguish them.*

On the other hand, consider the so-called Alexandroff topology on the set of finite binary lists, which is a binary tree when understood as a partial order.



The open sets of the Alexandroff topology are precisely the upward closed sets, generated by a union of sets of the form $\uparrow xs$ where xs is a finite binary list. We have the completely join prime filter consisting of $\{\uparrow [\,], \uparrow [0], \uparrow [0,0], \uparrow [0,0,0] \ldots\}$, but observe that this filter is not of the form $\eta(xs)$ for any xs, since we can always find an open set $\uparrow [0 : |xs| + 1$ many times$]$ which does not contain xs. Hence, we have generated a new point, not corresponding to any original point in the topology.

The first example demonstrates how we start with three distinct points, but they satisfy exactly the same properties, so the spatialisation operation on ΩX cannot distinguish the points. On the other hand, the second example shows that not all "complete" sets of properties are embodied by a point, so the spatialisation operation ends up hallucinating a new point entirely. If we restrict attention to those spaces that do not hallucinate or confuse different points, then what we get are sober topological spaces.

Definition 3.3.2. A topological space $(X, \Omega X)$ is *sober* if the unit map $\eta_X : X \to \mathbf{pt}(\Omega X)$ of the adjunction is a homeomorphism.

We can obtain a more useful description of this condition by considering the reconstructed points as prime elements of ΩX. The unit map η_X can be expressed instead as the following prime element:

$$\eta_X(x) := \bigcup \{a \in \Omega X \mid x \notin a\}$$

It's complement is $\bigcap \{b \mid X - b \in \Omega X \,\&\, x \in b\}$, which is the closure of $\{x\}$ in X.

More generally, the complement b of a prime element is a closed subset of X that is *irreducible* in the sense that they cannot be expressed as a union $b_1 \cup b_2$ of proper closed subsets of b.

Proposition 3.3.3. *A space X is sober iff every irreducible closed subset is expressible as the closure of $\{x\}$ for some unique $x \in X$.*

Localification is also not perfect, since there may not be enough points to distinguish two properties - it is possible in general that $a \neq b$ but $\varepsilon(a) = \varepsilon(b)$. The following is an extreme example where we start with a locale with no points, so the spatialising the locale and the localifying it back yields the trivial locale.

Example 3.3.4. *Complete Boolean algebras are locales since they have infinite meets and joins, and the infinitary de-Morgan laws hold. In a complete Boolean algebra, the abstract points AKA prime elements correspond to co-atoms. There exist (co-)atom-less complete boolean algebras B, so if we spatialize B, then it has no elements. Therefore the corresponding localification must be the one-element locale 1, and not B. The counit map is then the unique map $B \to 1$, which identifies any two opens in B.*

As before, we can also restrict attention to those locales with sufficiently many points.

Definition 3.3.5. A locale A is *spatial* if the counit map $\varepsilon_A : A \to \Omega\mathbf{pt}(A)$ is a frame isomorphism.

Proposition 3.3.6. *A locale A is spatial iff for any $a, b \in A$, whenever $a \not\leq b$, then there exists $x \in \mathbf{pt}(A)$ s.t. $a \in x$ but $b \notin x$.*

3.4 Duality for Sober Spaces & Spatial Locales

We have previously identified the sober spaces which remain unchanged under the localification-spatialization process. Similarly, we identified the spatial locales which remain unchanged under the spatialization-localification process. Given a sober space X, its localification is spatial, and similarly a spatial locale A has a sober spatialization. This means that the adjunction between spaces and locales restricts to a correspondence between the sober spaces and the spatial locales.

Theorem 3.4.1 (Stone Duality)**.** *The adjunction of Theorem 3.2.10 restricts to an equivalence of categories* $\mathbf{Sob} \simeq \mathbf{SLoc}$.

This means that whenever we have a category of spaces, and we show each space in this category to be sober, then we can correspondingly express that category as a category of spatial locales. We just have to study how the defining conditions for the spaces correspond to the locales. In the following sections, we will do this for the various notions of domains considered as spaces.

3.5 Duality for Algebraic DCPOs

The first observation to make is that the Scott-open sets for an algebraic DCPO D are generated by the upsets $\uparrow c$ of compact elements $c \in \mathbf{K}D$. This observation captures the essence of algebraicity in a topological manner.

This section a sharpening of the developments in [Vic89, Section 9.3]. We motivate a property called *super-coherence* as proposed in [SVng], which is weaker than spectral algebraicity and corresponds exactly to the Scott topology on algebraic DCPOs.

Definition 3.5.1. Let X be a topological space. A basis for X is a set $\mathcal{B} \subseteq \omega X$ such that $\mathcal{B}$ is closed under finite intersection, and every open set is equal to a union of some elements in $\mathcal{B}$. A subbasis for X is a set $\mathcal{B}$ such that the closure of $\mathcal{B}$ under intersections is a basis.

Proposition 3.5.2. *Let $(D, \sqsubseteq)$ be an algebraic DCPO. Then the set of opens*

$$\uparrow \mathbf{K}D = \{ \uparrow c \mid c \in \mathbf{K}D \}$$

is a subbasis for ΣD.

Proof. Let U be a Scott-open set. Then we have that $U = \bigcup \{ \uparrow c \mid c \in U \cap \mathbf{K}D \}$, since for any $x \in U$, by the algebraicity of D, there is a compact element $c \sqsubseteq x$ in U, so $x \in \uparrow c \subseteq U$. □

This characterisation allows us to prove that the Scott topology is sober, placing algebraic DCPOs inside the Stone duality of the previous section.

Proposition 3.5.3. *Let $(D, \sqsubseteq)$ be an algebraic DCPO. Then ΣD is sober.*

Proof. We show that the following unit map is a homeomorphism.

$$\eta_D : D \to \mathbf{pt}(\Sigma D)$$

$$d \mapsto \{U \in \Sigma D \mid d \in U\}$$

(Injectivity) Let $d, e \in D$, and suppose $d \neq e$. Then either $d \not\sqsubseteq e$ or $e \not\sqsubseteq d$. By Proposition 3.1.5, the Scott topology induces $\sqsubseteq$ as its specialisation ordering, so $d \not\sqsubseteq e$ is equivalent to saying that there exists $U \in \Sigma D$ such that $d \in U$ but $e \notin U$, and similarly for $e \not\sqsubseteq d$. In either case, this shows that $\eta_D(d) \neq \eta_D(e)$.

(Surjectivity) Let $x \in \mathbf{pt}(\Sigma D)$. Intuitively, by Proposition 3.5.2, the contents of x (treated as a completely prime filter) are entirely determined by what basic opens x contains. So the idea here is that the set

$$K_x := \{c \in \mathbf{K}D \mid \ \uparrow c \in x\}$$

determines x, and since we are in an algebraic DCPO, a set of compact points determines a unique point in D. Therefore, we claim that K_x is directed and that $\eta_D(\bigsqcup^{\uparrow} K_x) = x$.

To see that K_x is directed, take $c, d \in K_x$. Then $\uparrow c$, $\uparrow d \in x$, so by x being a filter, $(\uparrow c \cap \uparrow d) \in x$. By Proposition 3.5.2,

$$\uparrow c \cap \uparrow d = \bigcup \{ \uparrow e \mid e \in \mathbf{K}D \ \& \ \uparrow e \subseteq (\uparrow c \cap \uparrow d) \}$$

Since x is completely prime, we must therefore have some $e \in \mathbf{K}D$ such that $\uparrow e \in x$ and $\uparrow e \subseteq (\uparrow c \cap \uparrow d)$, which equivalently means $e \in K_x$ and $e \geq c, d$. This shows K_x is directed.

Next, to see that $\{U \in \Sigma D \mid \bigsqcup^{\uparrow} K_x \in U\} = x$ we can reason in a

fairly formal fashion:

$$
\begin{aligned}
& \bigsqcup\nolimits^{\uparrow} K_x \in U \\
\iff & \bigsqcup\nolimits^{\uparrow} K_x \in \bigcup \{ \uparrow e \mid e \in \mathbf{K}D \ \& \ \uparrow e \subseteq U \} && \text{by Proposition 3.5.2} \\
\iff & \exists e \in \mathbf{K}D.\ \uparrow e \subseteq U \ \& \ \bigsqcup\nolimits^{\uparrow} K_x \in \uparrow e \\
\iff & \exists e \in \mathbf{K}D.\ \uparrow e \subseteq U \ \& \ e \leq \bigsqcup\nolimits^{\uparrow} K_x \\
\iff & \exists e \in \mathbf{K}D. \exists c \in K_x.\ \uparrow e \subseteq U \ \& \ e \leq c \\
\iff & \exists e, c \in \mathbf{K}D.\ \uparrow e \subseteq U \ \& \ \uparrow c \subseteq \uparrow e \ \& \ \uparrow c \in x \\
\iff & \exists e \in \mathbf{K}D.\ \uparrow e \subseteq U \ \& \ \uparrow e \in x && \text{by } x \text{ upwards closed} \\
\iff & U \in x && \text{by } x \text{ completely prime}
\end{aligned}
$$

(Open) Let U be a Scott-open set. Then by definition of the topology on $\mathbf{pt}(\Sigma D)$, the set $\varepsilon_{\Sigma D}(U) := \{x \in \mathbf{pt}(\Sigma D) \mid U \in x\}$ is open. We can then show that $\eta_D[U] = \varepsilon_{\Sigma D}(U)$, and is therefore also open. □

So algebraic DCPOs with their Scott topologies are sober, but can we identify the sober spaces which are the Scott topology on some algebraic DCPO? Clearly, this must have something to do with the topological characterisation of algebraicity we gave above. We can take it further by understanding $\uparrow \mathbf{K}D$ in purely topological terms. c is compact iff $\uparrow c$ is Scott open. Given an open set of the form $a = \uparrow x$ in a sober space, we have that the completely prime filter $\{b \mid x \in b\}$ corresponding to x is equal to $\uparrow a$ since $a \subseteq b$ iff $\uparrow x \subseteq b$ iff $x \in b$. We may therefore equivalently characterise such open sets a as those for which $\uparrow a$ is a completely prime filter, i.e. $\uparrow a$ is a point. Generalizing, we call such opens a completely prime. As we will see, every completely prime open a is of the form $\uparrow x$, except for the trivial case when $a = 0$.

Definition 3.5.4. In a locale A, an open $a \in A$ is *completely prime*[13] if, for any $S \subseteq A$, $a \leq \bigvee S$ implies $a \leq s$ for some $s \in S$. We denote the set of completely prime elements of A as $\mathbf{CP}(A)$, and if X is a topological space we take $\mathbf{CP}(X)$ as shorthand for $\mathbf{CP}(\Omega X)$.

[13] Because $\uparrow a$ becomes a completely prime filter in this case.

With this characterisation, we can say that the Scott topology of an algebraic DCPO has a basis of completely prime opens, which characterizes algebraicity in purely topological terms.

Definition 3.5.5. A topological space X is *super-coherent*[14] if it is sober and has a basis of completely prime opens. Let **SCohSp** denote the full subcategory of **Sob** containing the super-coherent spaces.

Similarly, a locale A is super-coherent if it is spatial and has a basis of completely prime opens, with **SCohLoc** the corresponding full subcategory of **SLoc**.

[14] Super-coherent is actually a terrible name because this condition does not imply X is coherent.

This topological characterisation of algebraicity in a DCPO suggests that we can restrict the duality between sober spaces and spatial locales to algebraic DCPOs, with super-coherent topologies arising on the spatial/localic side.

Theorem 3.5.6. *Let X be a super-coherent space. Then*

1. *the compact points of X are in 1-to-1 order-reversing correspondence with the (non-trivial) completely prime opens;*
2. *the induced specialisation ordering[15] $(X, \leq_X)$ is an algebraic DCPO; and*
3. *ΩX is the Scott topology generated by this ordering.*

[15] Note that for sober spaces, the induced specialisation ordering $\leq_X$ and the subset ordering $\subseteq$ obtained by considering points as completely prime filters coincide.

Proof. 1. Any point $x \in X$ can instead be considered as an abstract point $x \in \mathbf{pt}(\Omega X)$ since X is sober. We now show x can be expressed as a directed join

$$x = \bigcup \{ \uparrow a \mid a \in \mathbf{CP}(X) \ \& \ a \in x \}.$$

Note that by our previous analysis this morally says x is a directed join of the compact elements below it - we will soon prove that this is indeed the case formally.

It suffices to show:

$$b \in x \iff \exists a \in \mathbf{CP}(X). a \in x \ \& \ a \leq b$$

Since X has a basis of completely prime opens, we know that b is the join of the set $\downarrow_{\mathbf{CP}} b$ of **CP**-opens below it. Hence, by x being completely prime, we have

$$\begin{aligned} b \in x &\iff (\bigvee \downarrow_{\mathbf{CP}} b) \in x \\ &\iff \exists a \in \downarrow_{\mathbf{CP}} b. a \in x \\ &\iff \exists a \in \mathbf{CP}(X). a \in x \ \& \ a \leq b \end{aligned}$$

To see that the set $\{ \uparrow a \mid a \in \mathbf{CP}(X) \ \& \ a \in x \}$ is directed, take $a, b \in x$ such that $a, b \in \mathbf{CP}(X)$. Then we have the open $a \wedge b \in x$ since x is a filter, and moreover $a \wedge b = \bigvee \downarrow_{\mathbf{CP}} (a \wedge b)$. Hence, we have $\bigvee \downarrow_{\mathbf{CP}} (a \wedge b) \in x$, so by x being completely prime we have some completely prime $c \in x$ such that $c \leq a \wedge b$. Therefore, $\uparrow c \in \{ \uparrow a \mid a \in \mathbf{CP}(X) \ \& \ a \in x \}$, and $\uparrow a \subseteq \uparrow c$ and similarly $\uparrow b \subseteq \uparrow c$ so the set is indeed directed.

Now, we prove that the map

$$\mathbf{CP}(X) - \{0\} \to \mathbf{K}X$$

$$a \mapsto \uparrow a$$

is an order-reversing isomorphism. Clearly, $a \leq b$ iff $\uparrow a \supseteq \uparrow b$, which also gives us injectivity[16]. For surjectivity, let $x \in \mathbf{K}X$ be a compact point. Then we have:

$$\begin{aligned} & x = \bigcup \{ \uparrow a \mid a \in \mathbf{CP}(X) \ \& \ a \in x \} \\ \implies & x \subseteq \bigcup \{ \uparrow a \mid a \in \mathbf{CP}(X) \ \& \ a \in x \} \\ \implies & \exists a \in \mathbf{CP}(X) \cap x. x \leq \uparrow a \qquad \text{By compactness} \\ \implies & \exists a \in \mathbf{CP}(X) \cap x. x = \uparrow a \end{aligned}$$

[16] Consider:

$$\begin{aligned} & a \neq b \\ \implies & a \not\leq b \text{ or } b \not\leq a \\ \implies & \uparrow a \not\supseteq \uparrow b \text{ or } \uparrow b \not\supseteq \uparrow a \\ \implies & \uparrow a \neq \uparrow b \end{aligned}$$

which shows that any compact point x lies in the image of the given map.

2. Now that we have shown the correspondence between compact points and completely prime opens, it is easy to see that $\{ \uparrow a \mid a \in \mathbf{CP}(X) \ \& \ a \in x \}$ corresponds to $\downarrow_{\mathbf{K}} x$, so $\downarrow_{\mathbf{K}} x$ is directed and $x = \bigsqcup \downarrow_{\mathbf{K}} x$.

3. By [Joh82, Lemma II.1.9], every open set in a sober space is Scott-open, so it suffices to show the converse that every Scott-open set is open in X. This follows immediately by considering a Scott-open U, which by Proposition 3.5.2 is a union of **CP** opens, so must itself be open.

□

Since every algebraic DCPO induces a super-coherent space and vice versa, we have a correspondence between the two objects. In Proposition 3.1.6 we showed that the continuous functions correspond to monotone, join-preserving functions, so we can lift this to an equivalence of categories.

Corollary 3.5.7 (Stone Duality for Algebraic DCPOs)**. SCohLoc** $\simeq$ **SCohSp** $\simeq$ **AlgDCPO**.

3.6 *Spectral Spaces*

The point of constructing a duality theory for domains is to find a dual logic that we can use to reason about these domains. With this goal in mind, the super-coherent locales of the previous section can be seen as a logic for algebraic DCPOs. However, it is a logic with infinitary operations, in particular the infinitary join operation. Can we find a subcategory of super-coherent locales consisting of those locales whose structure is determined only by its finitary operations? This question can be stated more broadly for all locales, and the resulting subclass of locales we shall call *spectral locales*[17].

[17] This name comes from the fact that the spectral locales arise from the spectrum of a commutative ring, but this fact will not be relevant to our development. A synonym for "spectral" is "coherent", but we will mostly avoid the latter to avoid confusion with coherence spaces which are an entirely different thing.

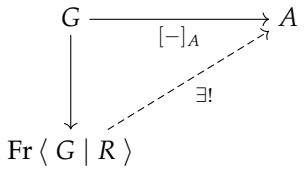
To reason about the finitary operations, we need to discuss presentations of frames, which are syntactic descriptions in terms of generators and relations:

$$\mathrm{Fr} \langle \text{ generators} \mid \text{relations} \rangle$$

The idea is that a presentation is a formula for formally constructing a frame. Consider the set of formal joins of finite meets of the generators, 0 and 1. These formal expressions are then quotiented based on the expressions given by the frame axioms and the relations given in the above specification, so that it is a frame by construction.

A *model* of a presentation $\mathrm{Fr} \langle G \mid R \rangle$ is a locale A as well as an assignment of generators $[-]_A : G \to A$, in such a way that the relations R are also satisfied. Another way to understand the frame presented by a presentation is that it is the unique frame satisfying the following universal property. For any model $(A, [-]_A : G \to A)$, there is a

unique frame homomorphism $\mathrm{Fr}\langle G \mid R \rangle \to A$ making the diagram commute:



Definition 3.6.1 ([Vic89, p. 41])**.** A presentation without relations is *free*. If a frame can be presented without relations, we also say the frame is free.

Example 3.6.2 ([Vic89, p. 40])**.** *The only elements we can generate with* $Fr\langle \mid \rangle$ *is* 0 *and* 1 *so this is the two-element frame* **2***. The frame* $Fr\langle a, b \mid \rangle$ *on the other hand yields the 6-element frame shown to the right.*

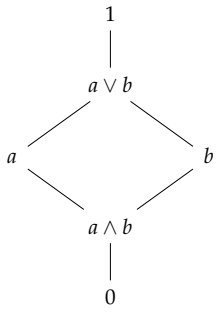


Figure 3.1: The frame generated by $\mathrm{Fr}\langle a, b \mid \rangle$

If the relations make no mention of infinitary operations, then essentially the infinitary structure of the frame is freely generated from a finitary part. This is what we mean by the structure of the locale being generated by the finitary operations.

Definition 3.6.3. A *finitary presentation* of a frame is a description

$$\mathrm{Fr}\langle G \mid R \rangle$$

such that R only makes use of 0, 1 as well as finitary joins and meets.

Definition 3.6.4. A frame is *coherent* if it has a finitary presentation. A locale A is *spectral* if it is coherent as a frame.

We can now make formal[18] this intuition that the infinitary structure of A is freely generated, using the notion of algebraicity[19].

Lemma 3.6.5. *Let* K *be a distributive lattice. Then* $\mathbf{Idl}(K)$ *is a coherent frame.*

Proof. First, observe that if K is a distributive lattice then $\mathbf{Idl}(K)$ is not only an algebraic DCPO, but also a locale. Its binary meet $I \wedge J$ is given by $I \cap J$. Since we have already shown before that $\mathbf{Idl}(K)$ has directed joins, it suffices to say that binary joins is given by

$$I \vee J = \downarrow \{x \vee y \mid x \in I, y \in J\}.$$

Then the join for any set of ideals S is the closure of S under binary joins, and then the directed join of this closure. Moreover, one can verify that any map induced by the universal property of the ideal completion on K will be a frame homomorphism.

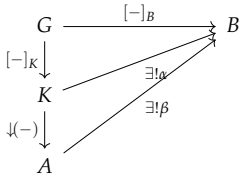
One can show that K can always be presented by $\mathbf{DLat}\langle G \mid R \rangle$. R is therefore a set of relations in the language of distributive lattices, which is also exactly the language of a coherent frame presentation. We may therefore also consider $\mathrm{Fr}\langle G \mid R \rangle$, which we claim is isomorphic to $A = \mathbf{Idl}(K)$.

This isomorphism follows easily: for any model B of $\mathrm{Fr}\langle G \mid R \rangle$, we can apply the universal property of K to obtain a unique map α, which then allows us to apply the universal property of A as the

[18] The core idea of the proof is that a coherent presentation of A can also be used to present a distributive lattice corresponding to $\mathbf{K}A$ such that A is the ideal completion of $\mathbf{K}A$.

[19] Note that in the previous section we developed the duality theory for algebraicity on the points of a frame. This time, we are developing duality theory for algebraicity on the frame itself. In the end, we will merge the two developments together.

ideal completion to obtain a unique map β making the diagram to the right commute. This shows A satisfies the universal property of $\mathrm{Fr}\langle G \mid R \rangle$, so they must be isomorphic.



□

Theorem 3.6.6 ([Vic89, Theorem 9.2.2])**.** *Let A be a frame. Then the following are equivalent:*

1. *A is coherent.*
2. *A is algebraic and $\mathbf{K}A$ is a sublattice of A.*

Proof. By the representation theorem for algebraic DCPOs, we can equivalently restate 2. as saying that A is isomorphic to $\mathbf{Idl}(K)$ for some distributive lattice K. The theorem then follows easily from the previous lemma:

(1. $\implies$ 2.) If A is coherent, then it has a presentation $\mathrm{Fr}\langle G \mid R \rangle$ for which R is in the language of a distributive lattice. Applying the lemma, this shows that $A \cong \mathbf{Idl}(K)$ where $K = \mathbf{DLat}\langle G \mid R \rangle$ is a distributive lattice.

(2. $\implies$ 1.) The distributive lattice K must have a presentation $\mathbf{DLat}\langle G \mid R \rangle$, and so $\mathrm{Fr}\langle G \mid R \rangle \cong \mathbf{Idl}(K) \cong A$, which is a coherent presentation of A. □

In section 2.2, we had an analogous situation but for the more general scenario between posets and DCPOs. There, we could have gone further and established an equivalence between the category of posets and the subcategory of algebraic DCPOs with Scott-continuous functions that preserve compact elements. We can now do the same between distributive lattices and coherent frames, which becomes a duality between distributive lattices and spectral locales.

Definition 3.6.7. Let A, B be spectral locales. A locale morphism $f : A \to B$ is *spectral* if the corresponding frame homomorphism $f : B \to A$ preserves compact elements. The category **SpecLoc** is the subcategory of spectral locales with morphisms being the spectral maps[20].

[20] In the literature, these are also called perfect maps.

Corollary 3.6.8. $\mathbf{Idl}(-) : \mathbf{DLat}^{op} \to \mathbf{SpecLoc}$ *is an equivalence of categories.*

We can extend the definition of spectral to topological spaces.

Definition 3.6.9. A topological space X is *spectral* if ΩX is spectral. Similarly, a continuous map $f : X \to Y$ is *spectral* if Ωf is spectral.

Let **SpecSp** denote the category of spectral spaces with spectral maps as morphisms.

Theorem 3.6.10 ([Vic89, Theorem 9.2.4])**.** *Spectral locales are spatial.*

Combining Theorem 3.6.6 and Theorem 3.6.10, we find the following characterisation of spectral spaces and spectral maps.

Corollary 3.6.11. *1. Let X be a topological space. Then X is spectral iff*

(a) *X is sober,*

(b) *the compact open sets form a basis for X, and*

(c) *the compact open sets are closed under finite intersection*[21].

[21] in particular, the whole space is compact since it is the empty intersection.

2. *Let $f : X \to Y$ be a continuous map between spectral spaces. Then f is spectral iff for any compact open set U, $f^{-1}[U]$ is also compact.*

3. *There is an equivalence of categories* $\mathbf{SpecSp} \simeq \mathbf{SpecLoc} \simeq \mathbf{DLat}^{op}$.

Now, we can combine the two conditions of spectrality and super-coherence, and give it a more economic definition. Since spectral spaces already have compact opens as a basis, and noting that **CP**-opens are also compact, it suffices to say that the **CP**-opens generate the compact opens.

Definition 3.6.12. A topological space X is *spectral algebraic* if X is spectral and super-coherent. Let **SpecAlgSp** denote the category of spectral algebraic spaces with the maps being arbitrary continuous maps[22].

[22] This is a really important detail: we do not restrict the maps to the spectral ones, while the category **SpecSp** does restrict to spectral maps.

Proposition 3.6.13. *Let X be a spectral space. Then X is spectral algebraic iff every compact open set in X can be expressed as a finite union of* **CP***-open sets.*

Proof. ($\Longleftarrow$) Since X is spectral, every open set can be expressed as a union of compact open sets, which in turn can be expressed as a union of **CP**-opens. Hence, every open set can be expressed as a union of **CP**-opens making the **CP**-opens a basis.

($\Longrightarrow$) By super-coherence, every compact open set U can be expressed as a union of **CP**-opens $\bigcup_{i \in I} U_i$. However, by the compactness of U we can in fact express it as a finite union.

□

Just as with super-coherence, we can characterize the above property purely in terms of the points, giving us a purely order-theoretic characterization of the spectral algebraic DCPOs.

Definition 3.6.14. A poset P satisfies the "2/3 bifinite" property if for each finite set $S \subseteq P$, the set M of minimal upper bounds of S is finite, and moreover for any other upper bound m' of S, there is some $m \in M$ s.t. $m \sqsubseteq m'$.

The "2/3 bifinite" property can be understood as a weaker version of the bounded completeness condition in Scott domains. The name is associated with another category of domains called the bifinite domains. They admit a wider class of domain constructions than Scott domains that come in handy for modelling concurrent and non-deterministic processes.

Let **2/3BifAlgDCPO** denote the full subcategory of **AlgDCPO** containing the 2/3 bifinite algebraic DCPOs.

Theorem 3.6.15 ([Vic89, Lemma 9.3.6 & 9.3.7])**.** *1. Let X be spectral algebraic space. Then* $\mathbf{pt}(\Omega X)$ *is an algebraic DCPO whose poset of compact points satisfy the "2/3 bifinite" property.*

2. *Let P be a poset satisfying the "2/3 bifinite" property. Then* $\mathbf{Idl}(P)$ *with the Scott topology is a spectral algebraic space.*

This allows us to sharpen the duality between algebraic DCPOs and super-coherent spaces even further, to include those DPCOs whose logic is finitary and therefore amenable to a proper logical treatment[23].

[23] with actual flesh and blood (i.e. a finite set of axioms and inference rules)

Corollary 3.6.16. $\mathbf{2/3BifAlgDCPO} \simeq \mathbf{SpecAlgSp}$.

In the next section, we will see that Scott Domains are 2/3 bifinite and sharpen the duality even further to obtain a topological characterization of Scott domains.

3.7 Duality for Scott Domains

Previously, we described the localic correspondent of algebraic DCPOs, and now we would like to sharpen it to Scott domains. One aid in this goal is the observation that Scott domains are also spectral, allowing us to use the duality we have just established for spectral algebraic locales and 2/3 bifinite algebraic DCPOs.

Proposition 3.7.1. *Let D be a Scott domain. Then D is 2/3 bifinite, so D is spectral algebraic, when considered as a space.*

Proof. Consider a finite set $S \subseteq D$. If S has no upper bound, then trivially the 2/3 bifinite condition holds for S. If S has an upper bound, then by definition of Scott domain, S has a least upper bound $\bigsqcup S$. Hence, the set of minimal upper bounds of S is the singleton $\{\bigsqcup S\}$, and every upper bound is above $\bigsqcup S$ by construction. □

What does the Scott domain condition on D mean in terms of its corresponding locale? well, compact elements $x \in \mathbf{K}D$ correspond to **CP**-opens $\uparrow x$. The join $x \sqcup y$ of two compact elements then corresponds to the **CP**-open $\uparrow (x \sqcup y) = \uparrow x \cap \uparrow y$. This join exists whenever x and y are consistent, i.e. have an upper bound. However, if no such upper bound exists, then that means $\uparrow x \cap \uparrow y = \emptyset$, which is also **CP**. So in any case, this means the **CP**-opens are closed under intersection. It remains to check that this is sufficient.

Theorem 3.7.2. *1. If D is a Scott domain, then equipped with the Scott topology it is a spectral algebraic space with the* **CP***-opens closed under intersection.*

2. Let X be a spectral algebraic space such that the **CP***-opens are closed under intersection. Then $\mathbf{pt}(\Omega X)$ is a Scott domain.*

Proof. 1. The preceding argument establishes this.

2. We have already shown that $\mathbf{pt}(\Omega X)$ will be an algebraic DCPO, so it remains to show $\mathbf{pt}(\Omega X)$ has joins of consistent elements. So, let $x, y \in \mathbf{pt}(\Omega X)$ such that x and y have an upper bound. If x and y are compact, then we have the join since their correspondents as **CP**-opens have meet, so this meet corresponds to the join of x and y.

   If x or y are not compact, we can obtain their join as follows. Consider the set $\downarrow_{\mathbf{K}} x \cup \downarrow_{\mathbf{K}} y$. This set is directed, by the preceding argument on the existence of joins for compact points. Hence, we can take its directed join, and this gives us the join of x and y.

□

3.8 Approximable Mappings

With Corollary 3.6.16, it seems that to obtain a logic for programs all we have to do is tidy up and connect our dualities:

1. Observe that **SpecAlgSp** is a subcategory of **SpecSp**.
2. Run **SpecAlgSp** under the duality $\mathbf{SpecSp} \simeq \mathbf{DLat}^{op}$, obtaining a dual equivalence to the category **AlgDLat** of distributive lattices where every element is a finite join of prime elements[24].
3. Chain the dualities up to obtain $\mathbf{2/3BifAlgDCPO} \simeq \mathbf{AlgDLat}^{op}$.
4. Construct a logical language for **AlgDLat**, and develop a semantic relation[25] $x \models \varphi$ where φ is a formula denoting some element of the distributive lattice and x is an element of the corresponding DCPO. The soundness and completeness should be follow from the duality. This we will do in the next chapter.

[24] This is just a translation of the condition in Proposition 3.6.13.

[25] Okay, maybe this step is not so easy.

Unfortunately its not that easy, for step 1 is an "obvious" but already incorrect observation. While the objects of **SpecAlgSp** are included in **SpecSp**, the morphisms are not. As established in chapter 2, we are interested generally in continuous maps between algebraic DCPOs, so the morphisms of **SpecAlgSp** should consist of continuous maps in general. However, the morphisms of **SpecSp** are spectral maps, which is a more restrictive class of maps. We therefore cannot run step 3, unless we expand the duality $\mathbf{SpecSp} \simeq \mathbf{DLat}^{op}$ to cover arbitrary continuous maps. This means that we have to represent continuous maps between spectral spaces purely in terms of their compact opens. Since a continuous map $f : X \to Y$ doesn't reflect compactness, it may associate a compact open in Y to an arbitrary open in X, which by algebraicity is the family of compact opens in X that map into Y. In other words, we have to describe a continuous map $X \to Y$ by a map $\mathbf{K}\Omega Y \to \mathcal{P}(\mathbf{K}\Omega X)$, or equivalently as a relation between the compact opens of X and Y.

Of course, not just any such relation will do, there will be certain properties that must hold if it were to simulate a continuous function. Once again, the relation aRb is supposed to mean $a \subseteq f^{-1}[b]$, so we note some basic properties of how R interacts with the operations of the lattice of compact open sets:

1. Given a family of compact opens a_i, if each a_i is a subset of $f^{-1}[b]$, then $\bigcup_{i\in I} a_i \subseteq f^{-1}[b]$. In lattice terms, $\forall i \in I.a_i R b \implies \bigvee_{i\in I} a_i R b$.
2. Given a family of compact opens b_i, if each $f^{-1}[b_i]$ contains a, then b_i contains $f[a]$, $f[a] \subseteq \bigcap_{i\in I} b_i$. Therefore, $a \subseteq \bigcap_{i\in I} f^{-1}[b_i] = f^{-1}[\bigcap_{i\in I} b_i]$. In lattice terms, $\forall i \in I.aRb_i \implies aR\bigwedge_{i\in I} b_i$.
3. Suppose we have compact opens a and b such that $a \subseteq f^{-1}[b]$. Then $f[a] \subseteq b$, so for any $a' \subseteq a$ and $b' \supseteq b$, we have $f[a'] \subseteq b'$, which means $a' \subseteq f^{-1}[b']$. In lattice terms, $a' \leq aRb \leq b' \implies a'Rb'$.

Now, given such a relation $R \subseteq \mathbf{K}\Omega X \times \mathbf{K}\Omega Y$, let us try to reconstruct a frame homomorphism[26] ${f_R}^{-1} : \Omega Y \to \Omega X$. Now, by algebraicity of ΩY, every open b is the join of the compact sets below it, i.e. $b = \bigvee \downarrow_{\mathbf{K}} b$. Since ${f_R}^{-1}$ has to preserve joins, ${f_R}^{-1}$ is therefore determined by how it behaves on compact opens in Y:

$$ {f_R}^{-1}(b) = {f_R}^{-1}(\bigvee \downarrow_{\mathbf{K}} b) = \bigvee {f_R}^{-1}[\downarrow_{\mathbf{K}} b]. $$

Moreover, as explained initially, the relation R relates each compact open c in Y to all the compact opens in X that map into Y, whose join should form the original inverse image of c. Therefore, we have for any compact open c in Y,

$$ {f_R}^{-1}(c) = \bigvee \{a \in \mathbf{K}\Omega X \mid aRc\}. $$

Combining the two requirements we therefore obtain the following definition:

$$ {f_R}^{-1}(b) = \bigvee \{a \in \mathbf{K}\Omega X \mid \exists c \in \mathbf{K}\Omega Y. c \subseteq b \text{ and } aRc\}. $$

Let us now check that this indeed gives us a frame homomorphism.

(Preservation of 1) Taking $I = \emptyset$ in condition 2. we have that $aR1$ for every compact open a. We therefore have ${f_R}^{-1}(1_{\Omega Y}) = \bigvee \{a \mid aR1\} = \bigvee(\mathbf{K}\Omega X) = 1_{\Omega X}$.

(Preservation of binary meets) We have the following chain of equalities:

$$ \begin{aligned} & {f_R}^{-1}(b_1 \wedge b_2) \\ = & \bigvee \{a \in \mathbf{K}\Omega X \mid \exists c \in \mathbf{K}\Omega Y. c \leq b_1 \wedge b_2 \text{ and } aRc\} \\ =^{*} & \bigvee (\{a \in \mathbf{K}\Omega X \mid \exists c_1 \in \mathbf{K}\Omega Y. c_1 \leq b_1 \text{ and } aRc_1\} \\ & \qquad \cap \{a \in \mathbf{K}\Omega X \mid \exists c_2 \in \mathbf{K}\Omega Y. c_2 \leq b_2 \text{ and } aRc_2\}) \\ =^{**} & (\bigvee \{a \in \mathbf{K}\Omega X \mid \exists c_1 \in \mathbf{K}\Omega Y. c_1 \leq b_1 \text{ and } aRc_1\}) \\ & \qquad \wedge (\bigvee \{a \in \mathbf{K}\Omega X \mid \exists c_2 \in \mathbf{K}\Omega Y. c_2 \leq b_2 \text{ and } aRc_2\}) \\ = & {f_R}^{-1}(b_1) \wedge {f_R}^{-1}(b_2) \end{aligned} $$

(Preservation of arbitrary joins) Unfolding the definition of ${f_R}^{-1}(\bigvee B)$ we obtain:

$$ \begin{aligned} {f_R}^{-1}(\bigvee B) &= \bigvee \{a \in \mathbf{K}\Omega X \mid \exists c \in \mathbf{K}\Omega Y. c \leq \bigvee B \text{ and } aRc\} \\ &= \bigvee \{a \in \mathbf{K}\Omega X \mid \exists c \in \mathbf{K}\Omega Y. \exists B' \subseteq_{fin} B. c \leq \bigvee B' \text{ and } aRc\} \end{aligned} $$

On the other hand, unfolding $\bigvee {f_R}^{-1}[B]$ gives us:

$$ \begin{aligned} \bigvee {f_R}^{-1}[B] &= \bigvee \{{f_R}^{-1}(b) \mid b \in B\} \\ &= \bigvee \{\bigvee \{a \in \mathbf{K}\Omega X \mid \exists c \in \mathbf{K}\Omega Y. c \leq b \text{ and } aRc\} \mid b \in B\} \\ &= \bigvee \{a \in \mathbf{K}\Omega X \mid \exists c \in \mathbf{K}\Omega Y. \exists b \in B. c \leq b \text{ and } aRc\} \end{aligned} $$

[26] Remember that by duality this corresponds to a continuous map $f_R : X \to Y$.

*We show that the two sets inside the $\bigvee$ are equal. The $(\subseteq)$ direction is obvious, so we focus on showing $(\supseteq)$. Take $a \in \mathbf{K}\Omega X$ with aRc_1 and aRc_2 for some $c_1 \leq b_1$ and $c_2 \leq b_2$. Then $c_1 \wedge c_2 \leq b_1 \wedge b_2$, and moreover by condition 2. we have that $aR(c_1 \wedge c_2)$. Therefore there is some c s.t. $c \leq b_1 \wedge b_2$ and aRc.

**Denote

$$ S_1 = \{a \mid \exists c_1 \in \mathbf{K}\Omega Y. c_1 \leq b_1 \;\&\; aRc_1\} $$

$$ S_2 = \{a \mid \exists c_2 \in \mathbf{K}\Omega Y. c_2 \leq b_2 \;\&\; aRc_2\}. $$

We show $\bigvee(S_1 \cap S_2) = (\bigvee S_1) \wedge (\bigvee S_2)$ by reasoning element-wise.

For the left-to-right direction, take $x \in \bigvee(S_1 \cap S_2)$. Then there is some a s.t. $x \in a \in S_1 \cap S_2$. Therefore, $x \in a \in S_1$ and $x \in a \in S_2$, so $x \in \bigvee S_1$ and $x \in \bigvee S_2$, which means $x \in (\bigvee S_1) \cap (\bigvee S_2)$.

For the right-to-left direction, take $x \in (\bigvee S_1) \cap (\bigvee S_2)$. Therefore, there are a_1 and a_2 s.t. $x \in a_1 \in S_1$ and $x \in a_2 \in S_2$. Since $a_1 \in S_1$, we have $a_1 R c_1$ for some compact $c_1 \leq b_1$ so by condition 3. of R, and the fact that $a_1 \wedge a_2 \leq a_1$, we have also $a_1 \wedge a_2 R c_1$. Therefore, $a_1 \wedge a_2 \in S_1$. By similar reasoning, we can show $a_1 \wedge a_2 \in S_2$. In conclusion, we have $x \in a_1 \wedge a_2 \in S_1 \cap S_2$, so $x \in \bigvee(S_1 \cap S_2)$.

Let us now try to prove the two to be equal. We clearly have that $\bigvee {f_R}^{-1}[B] \leq {f_R}^{-1}(\bigvee B)$, since the condition on each element a in the expression of $\bigvee {f_R}^{-1}[B]$ satisfies the condition in that of ${f_R}^{-1}(\bigvee B)$: just take $B' := \{b\}$. For the other direction, we take an arbitrary $a \in \mathbf{K}\Omega X$ with $c \in \mathbf{K}\Omega Y$ s.t. $c \leq \bigvee B'$ and aRc, and we try to show that $a \leq \bigvee {f_R}^{-1}[B]$.

It is not immediately clear that we can show $c \leq b$ for some $b \in B$. So to gather intuition let us first look at the case when R is the real thing, i.e. is obtained from a continuous function f. Then we can see a and c as compact open sets, and aRc means $a \subseteq f^{-1}(c)$. Since $c \subseteq \bigcup B'$, we can decompose c as the union of all the sets $c \cap b$ for $b \in B'$. The idea then is that each $c \cap b$ induces a corresponding subset $a_b \subseteq a$ s.t. $a_b \subseteq f^{-1}(c \cap b)$, by taking $a_b := f^{-1}(c \cap b) \cap a$. Then we have $a_b R c \cap b$, and $c \cap b \leq b$, so $a_b \in \{a \in \mathbf{K}\Omega X \mid \exists c \in \mathbf{K}\Omega Y. \exists b \in B. c \leq b \text{ and } aRc\}$. We can then conclude $a = \bigvee_{b \in B'} a_b \leq \bigvee \{a \in \mathbf{K}\Omega X \mid \exists c \in \mathbf{K}\Omega Y. \exists b \in B. c \leq b \text{ and } aRc\} = f^{-1}[B]$.

There are a couple hitches in this plan. First, $c \cap b$ is not necessarily compact, but to fix this we just have to further decompose $c \cap b$ into its unique decomposition of compact opens, as follows from the algebraicity of ΩY. The bigger issue is that we implicitly used a property of R that we have not imposed, even though it is true for relations of the form $- \subseteq f^{-1}(-)$. This property can be summarized as follows:

4. If we decompose on the right, i.e. $aR \bigvee C$, then we can similarly decompose on the left to obtain $a = \bigvee A$ s.t. for each $a' \in A$, $a' R c'$ for some $c' \in C$.

Once we impose this extra condition on R, we can show ${f_R}^{-1}$ to be a frame homomorphism - the proof for join-preservation follows as we laid out. Now we have characterized the continuous maps between spectral spaces in terms of certain relations between their compact opens. The compact opens are approximations of arbitrary opens, so we can think of these relations as *approximations of continuous maps*.

Definition 3.8.1. Let A, B be distributive lattices. Then an *approximable mapping* $R : A \to B$ is a relation $R \subseteq A \times B$ satisfying the following conditions, where I and J are finite indexing sets:

1. $\forall i \in I. a_i R b \implies \bigvee_{i \in I} a_i R b$.
2. $\forall i \in I. a R b_i \implies a R \bigwedge_{i \in I} b_i$.
3. $a' \leq aRb \leq b' \implies a' R b'$.
4. If $aR \bigvee_{i \in I} b_i$ Then there is a family $\{a_j\}_{j \in J}$ such that $a = \bigvee_{j \in J} a_j$ and $\forall j \in J. \exists i \in I. a_j R b_i$.

Consider now the composition of two continuous maps $f : X \to Y$ and $g : Y \to Z$. We find that

$$a \subseteq f^{-1}(g^{-1}(c)) \iff \exists b \in \mathbf{K}\Omega Y. a \subseteq f^{-1}(b) \text{ and } b \subseteq g^{-1}(c).$$

In terms of relations, this is just the standard relational composition.

Proposition 3.8.2. *Approximable mappings are closed under relational composition: if $R_1 : A \to B$ and $R_2 : B \to C$ are approximable then so is*

$$R_2 \circ R_1 = R_1; R_2 = \{(a, c) \mid \exists b \in B. a R_1 b \text{ and } b R_2 c\}$$

Moreover, the identity for this composition operation is the order relation, i.e. $id_A = \leq_A$.

Proof. The proof that approximable mappings are closed under relational composition is fairly routine:

1. Suppose for each $i \in I$, we have $a_i R_1; R_2 c$. Then that means we have a family b_i s.t. $a_i R_1 b_i R_2 c$. Therefore, by condition 2 for R_2, we have $\bigvee_{i\in I} b_i R_2 c$. Moreover, by condition 3 for R_1, each a_i satisfies $a_i R_1 \bigvee_{i\in I} b_i$, so by condition 1 for R_1 we have $\bigvee_{i\in I} a_i R_1 \bigvee_{i\in I} b_i$. Combining the observations above, we have $\bigvee_{i\in I} a_i R_1; R_2 c$.

2. Analogous to the proof for condition 1.

3. Suppose $a' \leq a R_1; R_2 c \leq c'$. Then we have b s.t. $a R_1 b R_2$. By condition 3 for R_1, we have $a' R_1 b$, and similarly by condition 3 for R_2, we have $b R_2 c'$. Therefore, $a' R_1; R_2 c'$.

4. Suppose $a R_1; R_2 \bigvee_{i\in I} c_i$, so then we have some b with $a R_1 b R_2 \bigvee_{i\in I} c_i$. By condition 4 for R_2, we have a family $\{b_j\}_{j\in J}$ such that $b = \bigvee_{j\in J} b_j$ and moreover for each j we have some c_i s.t. $b_j R_2 c_i$. Then, by condition 4 for R_1, from the fact that $a R_1 \bigvee_{j\in J} b_j$, we can similarly decompose a as $\bigvee_{k\in K} a_k$ s.t. for each $k \in K$, there is some b_j s.t. $a_k R_1 b_j$. As mentioned earlier, from this b_j we have some c_i s.t. $b_j R_2 c_i$, so $a_k R_1; R_2 c_i$.

To see that $\leq$ is the identity, suppose $a R_1; \leq b$. Then there is some b' s.t. $a R_1 b' \leq b$, so clearly by condition 3, $a R_1 b$. On the other hand, if $a R_1 b$ then $a R_1 b \leq b$ so $a R_1; \leq b$. By a similar fashion we can show that $\leq$ is the left-identity. □

Definition 3.8.3. Let **SpecSp**+ be the category of spectral spaces with morphisms being arbitrary continuous maps. Let **DLat**+ be the category of distributive lattices with the morphisms being approximable mappings.

Theorem 3.8.4. *There is an equivalence of categories* **SpecSp**+ $\simeq$ **DLat**+[27].

Using this expanded duality, we can actually carry out the program outlined in the beginning of this section. First of all, we observe that when we have an approximable mapping between two prime-generated distributive lattices[28], the fourth axiom may be simplified.

Proposition 3.8.5. *Let A, B be prime-generated distributive lattices. Then any relation $R : A \times B$ satisfying conditions 1 - 3 of Definition 3.8.1 and*

4′. If a is prime and $a R \bigvee_{i\in I} b_i$ then there is b_i such that $a R b_i$.

is an approximable mapping.

Proof. To prove condition 4., we can always take the prime decomposition of a, and use 4′. to show that each of the prime component maps to some b_i. □

[27] Note that with how we defined it, the morphisms of **DLat**+ are reversed w.r.t. to that of **DLat**, so technically we don't get a dual equivalence anymore. This makes it less confusing when thinking about the direction of approximable mappings, but it is something to keep in mind.

[28] i.e. exactly the lattices that correspond to spectral algebraic spaces.

4
Logics for Program Reasoning

In this chapter, we use the duality theory developed in the last section to construct a logic for reasoning about our imperative language **Imp**, following closely the technique of [Abr91, Chapters 3 & 4]. Here, we make the choice that our propositions/properties are to be compact opens. This has the advantage that the logic can be described using finitary operations, and in some sense there is no loss of information since our domains are spectral spaces which are generated by their compact opens. In another sense however, we do lose expressivity as there are certain program properties that are not expressible as a compact open. For a very simple example, consider the flat domain $\mathbb{N}_\perp$ of natural numbers. The property of being greater than or equal to 10 is not a compact open, for clearly we can find an open cover consisting of each singleton $\{n\}$ for $n \geq 10$ - this has no finite subcover. Nevertheless, we stick with compact opens for now to keep things simple. The work of Bonsangue [Bon98] extends the logic to consider open sets, and even non-open sets in order to incorporate infinitary conjunctions.

The technique in [Abr91] proceeds as follows:

1. Develop a propositional logic of properties, whose Lindenbaum algebra[1] corresponds to the compact open sets of a spectral algebraic space. In other words, we develop a propositional logic of compact opens.

2. Prove that the category of such propositional logics are equivalent to the category of Scott domains.

3. Understand the domain constructions introduced in chapter 2, including that of recursively constructed domains, in terms of these propositional logics.

4. Develop an axiomatic theory of when a program exhibits a certain property in this propositional logic. In other words, axiomatize the relation $\in$: $D \times \mathbf{K}\Omega D$.

5. Prove (soundness &) completeness for this theory.

[1] i.e. the quotient of the propositions under logical equivalence.

Points (1.) and (2.) are independent of the programming language, so for these we will mention results from [Abr91] without proof, and

refer there instead. For point (3.), since [Abr91] is interested in developing a logic for a generic meta-language for reasoning about arbitrary domains, they have to understand all the domain constructions in full generality. For our case of modelling **Imp**, we only need to worry about the domains $2_\perp$, $\mathbb{N}_\perp$, $\mathbb{Z}_\perp$ and Σ. As we will see, the propositions in the logic corresponding to these domains are very intuitive. The main work then lies in the axiomatization of point (4.) where the difficulty lies in designing our axioms such that the completeness proof of point (5.) goes through. We are not just going through this blindly however, for we are guided by the axioms of the meta-logic already developed in [Abr91].

4.1 Domain Prelocales

The essential difference between a classical propositional language and a Boolean algebra is that the former is a preorder, since if two formulas logically entail each other it does not mean that they are literally the same formula. For our logic of domain properties, we already know what our algebras are - they are the compact open sets of spectral algebraic spaces, or equivalently distributive lattices where every element is a finite join of prime elements. We now give the pre-ordered version of these lattices, called *domain prelocales*.

In this section, we use I and J to denote *finite* indexing sets.

Definition 4.1.1. A *spectral algebraic prelocale* is a structure

$$A = (|A|, \leq_A, \approx_A, 0_A, 1_A, \vee_A, \wedge_A, \mathsf{P}_A, \mathsf{T}_A)$$

where

- $|A|$ is a carrier set,
- $\leq_A$ and $\approx_A$ are binary relations over $|A|$,
- $0_A, 1_A$ are constants in $|A|$,
- $\vee_A, \wedge_A$ are binary operations on $|A|$,
- $\mathsf{T}_A, \mathsf{P}_A$ are unary predicates on $|A|$. Write $\mathsf{T}(A) = \{a \in |A| \mid \mathsf{T}(a)\}$ and similarly for $\mathsf{P}(A)$.

satisfying the distributive (pre-)lattice axioms[2] and some axioms characterizing the predicates[3].

If A additionally satisfies

$$\frac{\mathsf{P}(a) \quad \mathsf{P}(b)}{\mathsf{P}(a \wedge b)}$$

then it is a *(Scott) domain prelocale*.

The sets $\mathsf{P}(A)$ and $\mathsf{T}(A)$ are supposed to represent the non-zero prime compact opens and the compact opens that do not contain $\perp$, respectively. We say that $a \in A$ is *terminating* if $\mathsf{T}(a)$ holds[4].

Note that even though we did not add it as an axiom, we can show that $\mathsf{P}(A)$ indeed contains all the primes.

[2]

$$\frac{}{a \leq a} \quad \frac{a \leq b \quad b \leq c}{a \leq c} \quad \frac{a \leq b \quad b \leq a}{a \approx b} \quad \frac{a \approx b}{a \leq b \quad b \leq a}$$

$$\frac{}{0 \leq a} \quad \frac{a \leq c \quad b \leq c}{a \vee b \leq c} \quad \frac{}{a \leq a \vee b} \quad \frac{}{b \leq a \vee b}$$

$$\frac{}{a \leq 1} \quad \frac{a \leq b \quad a \leq c}{a \leq b \wedge c} \quad \frac{}{a \wedge b \leq a} \quad \frac{}{a \wedge b \leq b}$$

$$\frac{}{a \wedge (b \vee c) \leq (a \wedge b) \vee (a \wedge c)}$$

[3]

$$\frac{\mathsf{P}(a) \quad a \approx b}{\mathsf{P}(b)} \qquad \frac{\mathsf{P}(a) \quad a \leq \bigvee_{i \in I} b_i}{\exists i \in I. a \leq b_i}$$

$$\frac{}{\exists b_1 \ldots b_n \in \mathsf{P}(A). a \approx \bigvee_{i=1}^n b_i}$$

$$\frac{\mathsf{T}(a) \quad b \leq a}{\mathsf{T}(b)} \qquad \frac{\forall i \in I. \mathsf{T}(a_i)}{\mathsf{T}(\bigwedge_{i \in I} a_i)} \qquad \frac{\forall i \in I. \mathsf{T}(a_i)}{\mathsf{T}(\bigvee_{i \in I} a_i)}$$

$$\frac{\mathsf{T}(a)}{a \neq 1} \qquad \frac{a \neq 1}{\mathsf{T}(a)}$$

[4] The termination predicate is necessary because our imperative language is a strict language, so certain reasoning principles in our program logic will only hold assuming certain termination conditions. This will become clearer in the following sections.

Proposition 4.1.2. *Let A be a spectral algebraic prelocale. If an element a is prime in the sense that for any finite set S with $a \leq \bigvee S$, we have $a \leq s$ for some $s \in S$, then $\mathsf{P}(a)$.*

Proof. Consider a prime decomposition of a. By the assumption it has to be equal to one of its prime components, so must itself be prime. See the formal proof below.

$$\dfrac{P(b_i) \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{\exists b_1 ... b_n \in \mathsf{P}(A).a = \bigvee_{i=1}^n b_i}}{a \leq \bigvee_{i=1}^n b_i}}{a \leq b_i}\,\text{Ass} \qquad \dfrac{\dfrac{}{\exists b_1 ... b_n \in \mathsf{P}(A).a = \bigvee_{i=1}^n b_i}}{b_i \leq a}}{b_i \approx a}}{P(a)}$$

□

The notion of morphism between prelocales are the approximable mappings that we discussed in the previous chapter, since they are the ones that correspond to continuous maps.

Definition 4.1.3. Let A, B be domain prelocales. Then an *approximable mapping* $R : A \to B$ is a relation $R \subseteq A \times B$ satisfying the following conditions:

1. $\forall i \in I. a_i R b \implies \bigvee_{i \in I} a_i R b$.
2. $\forall i \in I. a R b_i \implies a R \bigwedge_{i \in I} b_i$.
3. $a' \leq aRb \leq b' \implies a' R b'$.
4. If $\mathsf{P}(a)$ and $aR \bigvee_{i \in I} b_i$ then $\exists i \in I. aRb_i$.

This allows us to prove that the two categories are equivalent, giving us a logical view of domains.

Definition 4.1.4. Let A be a spectral algebraic prelocale. The *Lindenbaum algebra* $A/_\approx$ is the distributive lattice obtained by quotienting[5] $|A|$ by the relation $\approx_A$.

[5] We can easily show that $\approx_A$ is a congruence on the distributive lattice operations based on the prelocale axioms.

Theorem 4.1.5. *The two functors*

$$F(D) = (\mathbf{K\Omega}D, \subseteq, =, \emptyset, D, \cup, \cap, \mathbf{CP}(D), \{U \mid \ \perp \notin U\})$$

and

$$G(A) = \{F \subseteq |A| \mid F \text{ is a prime filter}\}$$

form an equivalence of categories **Scott** $\simeq$ **DPL**.

4.2 Domain Prelocales for **Imp**

In this section, we define some domain prelocales that correspond to the domains we care about for modelling **Imp**. Ideally, we want to present the prelocales in a syntactic way by freely generating them from some generator elements and axioms[6], so that our program logic

[6] This free generation of algebraic structures is a common construction in algebra, and we refer to **??** for a primer on this technique.

is a fully syntactic construction[7]. This also leads to a better understanding of the compact open sets in our domains. However, this would add many induction cases to the completeness proof of our logic. To strike a fair balance, we avoid doing this for the flat domains $\mathbb{N}_\perp, \mathbb{Z}_\perp$ and $2_\perp$. We already understand the compact opens in flat domains really well anyway.

[7] After all, to some people a formal system is only a logic if it is given by a finite set of axiom schemas and inference rules.

Definition 4.2.1. Let $L(2_\perp)$ be the domain pre-locale obtained by

$$(\mathbf{K\Omega}2_\perp, \subseteq, =, \emptyset, 2_\perp, \cup, \cap, \mathbf{CP}(D), \{U \mid \perp \notin U\}),$$

and similarly for $L(\mathbb{Z}_\perp)$. We define denotation functions $[\![-]\!] : L(\mathbb{Z}_\perp) \to \mathbf{K\Omega}\mathbb{Z}_\perp$ and $[\![-]\!] : L(2_\perp) \to \mathbf{K\Omega}2_\perp$ as the identity function.

In a flat domain, the compact opens are just the finite open subsets plus the whole set itself. We have an even more explicit description of the prime elements: in a flat domain they just correspond to the singleton open subsets, plus the whole set itself.

This leaves only $L(\Sigma)$ for which we do not yet have a good understanding of the compact opens. We shall analyze $\mathbf{K\Omega}\Sigma$ and find some simple elements that can be used to generate all the compact opens under $\wedge$ and $\vee$, as well as some axioms that the elements have to obey. Based on this analysis, we define a freely generated prelocale that is equivalent to $\mathbf{K\Omega}\Sigma$.

The starting point of our analysis is the characterization of the compact elements in the Scott domain $[D \to E]$ and $[D \to_\perp E]$ (Proposition 2.4.7). Since we know that the **CP**-opens correspond to the compact elements and that they generate all the compact opens, they are prime[8] candidates for our generating elements.

[8] Pun fully intended.

A compact pair function $\langle d; e\rangle$ where d and e are compact corresponds to the **CP**-open

$$(\uparrow d) \to (\uparrow e) = \{f \in [D \to E] \mid \forall d' \sqsupseteq d. f(d') \sqsupseteq e\}.$$

A compact function in $[D \to E]$ is a join of a finite consistent set of compact pair functions, which topologically corresponds to the intersection of their corresponding **CP**-opens. The consistency requirement corresponds to asking that this intersection be non-empty, i.e. $\bigcap_{i\in I}((\uparrow d_i) \to (\uparrow e_i)) \neq \emptyset$. For $[D \to_\perp E]$, there is an additional requirement that each $d_i \neq \perp$.

Therefore, every **CP**-open can be expressed as as a finite meet

$$\bigwedge_{i\in I}(a_i \to b_i)$$

as long as the a_i and b_i are **CP**-open and $\bigwedge_{i\in I}(a_i \to b_i)$ is terminating. Specializing to our scenario, we want to consider the **CP**-opens in $\Sigma = [\mathbb{N}_\perp \to_\perp \mathbb{Z}_\perp]$. Since the **CP**-opens in $\mathbb{N}_\perp$ and $\mathbb{Z}_\perp$ are either singletons or the whole set, a compact pair function just corresponds to a **CP**-open of the form

$$(n \rightsquigarrow k) = \{s \in \Sigma \mid s(n) = k\} \qquad n \in \mathbb{N}, k \in \mathbb{Z}$$

or the top element 1_Σ which is always **CP**-open, so we will ignore it in our following analysis.

As for consistency, a set of such **CP**-opens are consistent if it does not contain a pair $(n \rightsquigarrow k_1), (n \rightsquigarrow k_2)$ with $k_1 \neq k_2$. Therefore, a **CP**-open is in general of one of the following form:

$$\bigwedge_{i \in I} (n_i \rightsquigarrow k_i) \text{ with } n_i \neq n_j \text{ for all } i \neq j \in I$$

This means that a compact open set is in general a finite join of **CP**-opens of the above form. Based on this analysis, we can define the following prelocale.

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

$$\{(n \rightsquigarrow k) \mid n \in \mathbb{N}, k \in \mathbb{Z}\}.$$

satisfying additional rules:

$$\frac{k_1 \neq k_2 \qquad n \in \mathbb{N}}{(n \rightsquigarrow k_1) \wedge (n \rightsquigarrow k_2) = 0}\ \Sigma\text{-}n \qquad \frac{\forall i, j \in I. n_i = n_j \implies i = j}{\mathsf{P}(\bigwedge_{i \in I} (n_i \rightsquigarrow k_i))}\ \Sigma\text{-P-}\bigwedge$$

Note that an element in this generated prelocale can always be expressed as a join of meet of generators, by distributivity. Since the rules we added imply that any meet of generators is either prime or 0, we automatically have that any element can be expressed as a join of primes (the 0s can be vanished from the join).

We will now define a function that interprets elements of $L(\Sigma)$ as compact opens in Σ. To verify that our construction is correct, we prove that this prelocale is isomorphic to $\mathbf{K}\Omega\Sigma$. Along the way, we have to prove a soundness[9] and completeness theorem.

[9] In fact, the map we define below is only well-defined due to soundness.

Definition 4.2.3. We define the function $[\![-]\!] : L(\Sigma) \to \mathbf{K}\Omega\Sigma$ on its generators:

$$[\![n \rightsquigarrow k]\!] = \{s \in \Sigma \mid s(n) = k\}$$

Which extends to $L(\Sigma)$ by interpreting conjunction by intersection, and disjunction by union.

Theorem 4.2.4 (Soundness)**.** *Let $a, b \in L(\Sigma)$. Then*

$$a \leq b \implies [\![a]\!] \subseteq [\![b]\!].$$

Proof. By induction on the rules for a spectral algebraic prelocale. Note that we have not added any rules producing a $\leq$-statement to $L(\Sigma)$. □

We prove completeness for prime elements first, which we can then easily lift to all elements.

Lemma 4.2.5 (Prime Completeness)**.** *Let $a, b \in \mathsf{P}(L(\Sigma))$. Then*

$$[\![a]\!] \subseteq [\![b]\!] \implies a \leq b.$$

Proof. By induction on the proof of $\mathsf{P}(a)$ and $\mathsf{P}(b)$. We prove only the cases for $L(\Sigma)$ axioms, since the cases for the generic domain prelocale rules are obvious.

If $\mathsf{P}(a)$ and $\mathsf{P}(b)$ are derived by (Σ-P-$\bigwedge$), then $a = \bigwedge_{j \in J} (n_j \rightsquigarrow k_j)$ and $b = \bigwedge_{i \in I} (n_i \rightsquigarrow k_i)$, so we have that for each $i \in I$,

$$[\![a]\!] \subseteq \bigcap_{i \in I} [\![n_i \rightsquigarrow k_i]\!] \subseteq [\![n_i \rightsquigarrow k_i]\!].$$

Therefore, by induction hypothesis, $a \leq n_i \rightsquigarrow k_i$ for each $i \in I$, which means $a \leq \bigwedge_{i\in I}(n_i \rightsquigarrow k_i) = b$. □

In order to prove completeness for all elements, we use the fact that every element can be decomposed as a join of primes, and then apply the prime completeness.

Theorem 4.2.6. *1. Let $a, b \in L(\Sigma)$. Then*

$$[\![a]\!] \subseteq [\![b]\!] \implies a \leq b.$$

2. $[\![-]\!]$ induces an isomorphism from $(L(\Sigma)/_{\approx})$ to $\mathbf{K\Omega\Sigma}$.

Proof. 1. Taking prime decompositions $a = \bigvee_{i\in I} a_i$ and $b = \bigvee_{j\in J} b_j$ with each $[\![a_i]\!] = \uparrow s_i$ and $[\![b_j]\!] = \uparrow t_j$

$$\begin{aligned}
& [\![a]\!] \subseteq [\![b]\!] \\
\iff & \left[\!\!\left[\bigvee_{i\in I} a_i\right]\!\!\right] \subseteq \left[\!\!\left[\bigvee_{j\in J} b_j\right]\!\!\right] \\
\iff & \bigcup_{i\in I} [\![a_i]\!] \subseteq \bigcup_{j\in J} [\![b_j]\!] \\
\implies & \forall i \in I. \exists j \in J. [\![a_i]\!] \subseteq [\![b_j]\!] \\
\implies & \forall i \in I. \exists j \in J. a_i \leq b_j \qquad \text{by completeness for primes} \\
\implies & \forall i \in I. a_i \leq \bigvee_{j\in J} b_j \\
\implies & \bigvee_{i\in I} a_i \leq \bigvee_{j\in J} b_j
\end{aligned}$$

2. Equivalently, we can show that $[\![-]\!]$ is a surjective function satisfying

$$\forall a, b \in L(\Sigma). a \leq b \iff [\![a]\!] \subseteq [\![b]\!]$$

Of course, this condition is just soundness & completeness as we have just proven, and the surjectivity follows from by our analysis of prime opens (i.e. it is surjective by construction).

□

As a consequence of completeness and our previous analysis, we have the following normal form theorem for $L(\Sigma)$, which is baked into the design of $L(\Sigma)$ in the first place.

Corollary 4.2.7. *Let $a \in \mathsf{P}(L(\Sigma))$. Then either $a \approx 1$, or there are $\{n_i\}_{i\in I} \in \mathbb{N}$ and $\{k_i\}_{i\in I} \in \mathbb{Z}$ with $n_i \neq n_j$ for all $i, j \in I$ such that*

$$a \approx \bigwedge_{i\in I} (n_i \to k_i).$$

4.3 *The Program Logic of* **Imp**

Now that we have a language for program properties set up, we are finally in the position to define the full program logic. Our logic turns out to be a variant of Hoare logic: this is not an ad-hoc decision, but

rather its naturally motivated by translating **Imp** into the metalanguage of [Abr91, Chapter 4] and analyzing the form of verifications in their logic. In the next section, we will compare and contrast our logic with more standard forms of Hoare logic found in the "wild".

The basic form in our logic is a Hoare triple $[\Phi]\ M\ [\Psi]$ where M is a program (i.e. a command, boolean expression or arithmetic expression), Φ is a property of states, and Ψ is a property of the program type of M. One should interpret the Hoare triple as saying that the result of running program M at any state exhibiting property Φ exhibits property Ψ.

4.3.1 *The Logic* $\mathcal{L}_{\mathbf{Imp}}$

Let the following symbols denote

- $\alpha, \beta \in L(\mathbb{Z}_\perp)$
- $\chi, \kappa \in L(2_\perp)$
- $\phi, \psi, \theta \in L(\Sigma)$
- $\Phi, \Psi, \Theta, \Phi', \Psi'$ are metavariables standing in for an element of any of the pre-locales.

We also denote singleton sets by the element it contains instead of the set itself, to avoid clutter with braces. For example, the singleton set $\{\mathtt{tt}\} \in L(2_\perp)$ is denoted just `tt`. Furthermore, we lift the denotations of the boolean and arithmetic operations as operations on compact open sets of $\mathbb{N}_\perp$ and $2_\perp$, defined as follows:

- $\alpha \Box \beta := \bigvee\{\uparrow(k_1 [\![\Box]\!] k_2) \mid k_1 \in \alpha, k_2 \in \beta\}$ for $\Box \in \{\mathtt{+}, \mathtt{-}, *, \mathtt{=}, \mathtt{<=}\}$
- $\chi \Box \kappa := \bigvee\{\uparrow(b_1 [\![\Box]\!] b_2) \mid b_1 \in \chi, b_2 \in \kappa\}$ for $\Box \in \{\mathtt{and}, \mathtt{or}\}$
- $\mathtt{not}\ \chi := \bigvee\{\uparrow([\![\mathtt{not}]\!]\, b_1) \mid b_1 \in \chi\}$

Note that we take the upwards closure only to cover the case when one of the operands contains $\perp$, where we have to ensure we take every element above $\perp$ to make the resulting set open. For example, this ensures that if $\alpha = 1_{\mathbb{N}_\perp}$ then $\alpha\ \mathtt{+}\ \beta = 1_{\mathbb{N}_\perp}$.

Generic Rules

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

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

Rules for Arithmetic Expressions

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

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

Rules for Boolean Expressions

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

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

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

Rules for Commands

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

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

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

$$\frac{[\phi]\ A\ [\alpha] \quad \mathsf{T}(\alpha)}{[\phi]\ \texttt{def}\ n\ \texttt{:=}\ A\ [\bigvee_{k\in\alpha}(n \rightsquigarrow k)]}\ \text{H-def} \qquad \frac{n_i \neq n}{[\bigwedge_{i\in I}(n_i \rightsquigarrow k_i)]\ \texttt{def}\ n\ \texttt{:=}\ A\ [n_i \rightsquigarrow k_i]}\ \text{H-def-}\wedge$$

4.3.2 *The Semantics*

We wish to interpret a Hoare triple $[\Phi]\ M\ [\Psi]$ as saying

$$[\![M]\!]\ [\![\Phi]\!] \subseteq [\![\Psi]\!]\,,$$

i.e. whenever a state $s \in \Sigma$ exhibits property Φ, then the program M running at state s exhibits property Ψ. The denotation operations for programs and properties are of course the ones given in section 2.5 and section 4.2, respectively.

To that end, we write $\vdash [\Phi]\ M\ [\Psi]$ to say a Hoare triple is derivable by the rules given above, and $\vDash [\Phi]\ M\ [\Psi]$ to mean $[\![M]\!]\ [\![\Phi]\!] \subseteq [\![\Psi]\!]$.

4.3.3 *Soundness*

As usual, proving the soundness of our logic is a simple matter of induction.

Theorem 4.3.1. *1. Let A be an arithmetic expression in* **Imp**. *Then*

$$\vdash [\phi]\ A\ [\alpha] \implies \vDash [\phi]\ A\ [\alpha].$$

2. Let B be a Boolean expression in **Imp**. *Then*

$$\vdash [\phi]\ B\ [\chi] \implies \vDash [\phi]\ B\ [\chi].$$

3. Let C be a Command in **Imp**. *Then*

$$\vdash [\phi]\ C\ [\psi] \implies \vDash [\phi]\ C\ [\psi]$$

Proof. By induction on the rules. □

4.3.4 Completeness

For the completeness proof, as before we prove the result for prime elements first, since we have a much better grasp on them. It may feel like the rules seem to fit perfectly in the completeness proof, but this is because we chose the rules in such a way that the completeness proof goes through. In this sense, the logic is determined by our denotational semantics, and one should view the following proof as an explanation for why we chose the rules above.

Lemma 4.3.2. *Let A be an arithmetic expression in* **Imp**, $\phi \in \mathsf{P}(L(\Sigma))$ *and* $\alpha \in \mathsf{P}(L(\mathbb{Z}_\perp))$. *Then*

$$\vDash [\phi]\ A\ [\alpha] \implies \vdash [\phi]\ A\ [\alpha]$$

Proof. First, let us analyse α. In $\mathbb{N}_\perp$, either α is a singleton $\{k\}$ or it is $1_{\mathbb{N}_\perp}$. In the latter case, we can always derive the Hoare triple $[\phi]\ A\ [1]$, so we can proceed with the assumption $\alpha = \{k\}$. Moreover, since ϕ is prime, $[\![\phi]\!] = \uparrow s$ for some compact element $s \in \Sigma$. We now carry out the proof by induction on A.

For $\square \in \{+, -, *\}$, we have the inductive cases:

$$\begin{aligned}
&\vDash [\phi]\ A_1 \square A_2\ [k] \\
\implies &[\![A_1 \square A_2]\!]\,[\uparrow s] \subseteq \{k\} \\
\implies &[\![A_1 \square A_2]\!]\,(s) \in \{k\} \\
\implies &[\![A_1 \square A_2]\!]\,(s) = k \\
\implies &[\![A_1]\!]\,(s)\,[\![\square]\!]\,[\![A_2]\!]\,(s) = k \\
\implies &\exists k_1, k_2 \in \mathbb{Z}.k_1\,[\![\square]\!]\,k_2 = k \text{ and } [\![A_1]\!]\,(s) = k_1 \text{ and } [\![A_2]\!]\,(s) = k_2 \\
\implies &[\![A_1]\!]\,(\uparrow s) \subseteq \{k_1\} \text{ and } [\![A_2]\!]\,(\uparrow s) \subseteq \{k_2\} && \text{by monotonicity} \\
\implies &\vDash [\phi]\ A_1\ [k_1] \text{ and } \vDash [\phi]\ A_2\ [k_2] \\
\implies &\vdash [\phi]\ A_1\ [k_1] \text{ and } \vdash [\phi]\ A_2\ [k_2] && \text{by IH} \\
\implies &\vdash [\phi]\ A_1 \square A_2\ [\{k_1\} \square \{k_2\} = k_1\,[\![\square]\!]\,k_2 = k]
\end{aligned}$$

In the case A is a constant $k' \in \mathbb{Z}$, we either have a contradiction if $k' \neq k$, or if $k = k'$ then we can derive the Hoare triple by H-$\mathbb{Z}$.

Finally, if A is a variable $n \in \mathbb{N}$, then

$$\begin{aligned}
&\vDash [\phi]\ n\ [k] \\
\implies &[\![n]\!]\,[\uparrow s] \subseteq \{k\} \\
\implies &[\![n]\!]\,(s) = k \\
\implies &s(n) = k
\end{aligned}$$

If we apply the normal form theorem on ϕ, then we find that either $s = \bigsqcup_{i \in I} \langle n_i; k_i \rangle$ or $s = \perp_\Sigma$. In the first case, we find that $n = n_i$ and $k = k_i$ for some $i \in I$, so we can apply the rule:

$$\frac{}{[\bigwedge_{i \in I}(n_i \rightsquigarrow k_i)]\ n_i\ [k_i]}\ \text{H-var-}\bigwedge$$

In the second case, we have a contradiction, since $\perp_\Sigma\ (n) = \perp_{\mathbb{Z}_\perp} \neq k$. □

Lemma 4.3.3. *Let B be a Boolean expression in* **Imp**, $\phi \in \mathsf{P}(L(\Sigma))$ *and* $\chi \in \mathsf{P}(L(2_\perp))$. *Then*

$$\vDash [\phi]\ B\ [\chi] \implies \vdash [\chi]\ B\ [\chi]$$

Proof. Analogous to the proof of prime completeness for arithmetic expressions. In fact, since Boolean expressions make no use of variables, this should be easier and not require the normal form theorem. □

Lemma 4.3.4. *Let C be a command in* **Imp** *and* $\phi, \psi \in \mathsf{P}(L(\Sigma))$. *Then*

$$\vDash [\phi]\ C\ [\psi] \implies \vdash [\phi]\ C\ [\psi]$$

Proof. Since ϕ and ψ are prime, there are compact elements $s, t \in \mathbf{K}\Sigma$ such that $[\![\phi]\!] = \uparrow s$ and $[\![\psi]\!] = \uparrow t$. We now prove this result by induction on C.

- ($C = \texttt{skip}$) We have by soundness & completeness for $L(\Sigma)$,

  $$\vDash [\phi]\ \texttt{skip}\ [\psi] \iff [\![\texttt{skip}]\!]([\![\phi]\!]) \subseteq [\![\psi]\!] \iff [\![\phi]\!] \subseteq [\![\psi]\!] \iff \phi \leq \psi.$$

  Therefore, we can derive the Hoare triple by:

  $$\dfrac{\dfrac{}{[\phi]\ \texttt{skip}\ [\phi]}\ \text{H-}\texttt{skip} \qquad \phi \leq \psi}{[\phi]\ \texttt{skip}\ [\psi]}\ \text{H-}\leq$$

- ($C = \texttt{if}\ B\ \texttt{then}\ C_1\ \texttt{else}\ C_2$) We have that

  $$\begin{aligned} &\vDash [\phi]\ \texttt{if}\ B\ \texttt{then}\ C_1\ \texttt{else}\ C_2\ [\psi] \\ \implies &t \sqsubseteq [\![\texttt{if}\ B\ \texttt{then}\ C_1\ \texttt{else}\ C_2]\!](s) \end{aligned}$$

  Then, depending on the value of $[\![B]\!]_s$ we have three possible scenarios. If $[\![B]\!]_s = tt$, we have $[\![\texttt{if}\ B\ \texttt{then}\ C_1\ \texttt{else}\ C_2]\!](s) = [\![C_1]\!](s)$ so:

  $$\begin{aligned} & t \sqsubseteq [\![C_1]\!](s) \text{ and } [\![B]\!]_s = tt \\ \implies & [\![C_1]\!](s) \in [\![\psi]\!] \text{ and } [\![B]\!]_s \in \{tt\} \\ \implies & [\![C_1]\!](\uparrow s) \subseteq [\![\psi]\!] \text{ and } [\![B]\!]_{\uparrow s} \subseteq \{tt\} \text{ by monotonicity} \\ \implies & \vDash [\phi]\ C_1\ [\psi] \text{ and } \vDash [\phi]\ B\ [\texttt{tt}] \\ \implies & \vdash [\phi]\ C_1\ [\psi] \text{ and } \vdash [\phi]\ B\ [\texttt{tt}] \qquad \text{by IH} \\ \implies & \vdash [\phi]\ \texttt{if}\ B\ \texttt{then}\ C_1\ \texttt{else}\ C_2\ [\psi] \quad \text{by H-}\texttt{ite-tt} \end{aligned}$$

  We can prove the case for $[\![B]\!]_s = ff$ analogously using H-`ite-ff`. Finally, if $[\![B]\!]_s = \perp$ then $[\![\texttt{if}\ B\ \texttt{then}\ C_1\ \texttt{else}\ C_2]\!] = \perp$, so $t = \perp$ implying that $\psi \approx 1$. We can then derive the required Hoare triple by H-1.

- ($C = C_1; C_2$) We have that

  $$\begin{aligned} &\vDash [\phi]\ C_1; C_2\ [\psi] \\ \implies &t \sqsubseteq [\![C_2]\!]([\![C_1]\!](s)) \end{aligned}$$

By Proposition 2.4.4, and the fact that Σ is a Scott domain, we can decompose $[\![C_2]\!]$ as a directed join

$$\bigsqcup_{i \in I} \langle r_i; t_i \rangle$$

of compact pair functions. Since t is compact, and $t \sqsubseteq \bigsqcup_{i \in I} \langle r_i; t_i \rangle ([\![C_1]\!](s))$, there is some $i \in I$ such that $t \sqsubseteq \langle r_i; t_i \rangle ([\![C_1]\!](s))$. Now, either $[\![C_1]\!](s) \sqsupseteq r_i$ or not. If it is, then $t \sqsubseteq \langle r_i; t_i \rangle ([\![C_1]\!](s)) = t_i$, so we have

$$t = \langle r_i; t \rangle (r_i) \sqsubseteq \langle r_i; t_i \rangle (r_i) \sqsubseteq [\![C_2]\!](r_i)$$

By the surjectivity of $[\![-]\!] : L(\Sigma) \to \mathbf{K}\Omega\Sigma$, there is some θ such that $[\![\theta]\!] = \uparrow r_i$. Therefore, we now have

$$\vDash [\phi]\ C_1\ [\theta] \text{ and } \vDash [\theta]\ C_2\ [\psi]$$

By the induction hypothesis and applying rule H-;, we can conclude that $\vdash [\phi]\ C_1; C_2\ [\psi]$.

If $[\![C_1]\!](s) \not\sqsupseteq r_i$, then $t = \bot$ meaning $\psi \approx 1$, so we can derive $\vdash [\phi]\ C_1; C_2\ [\psi]$ by H-1.

- ($C =$ `while` B `do` C_1) We have that

$$\begin{aligned}
&\vDash [\phi]\ \texttt{while}\ B\ \texttt{do}\ C_1\ [\psi] \\
\Longrightarrow\ & t \sqsubseteq [\![\texttt{while}\ B\ \texttt{do}\ C_1]\!](s) \\
\Longrightarrow\ & t \sqsubseteq \bigsqcup_{n \in \omega} F^n_{B,C_1}(\bot)(s) \\
\Longrightarrow\ & \exists n \in \omega. t \sqsubseteq F^n_{B,C_1}(\bot)(s) \quad \text{By compactness of } t
\end{aligned}$$

Without loss of generality, we can take the least such n for which this is true. We now prove the result by an inner induction on n.

  - (Base case n = 0) Then $\langle s; t \rangle = \bot$, so in fact $t = \bot$. This means $\psi \approx 1$, so we can just use the rule H-1 to derive the Hoare triple.
  - (Inductive case n + 1) Depending on the value of $[\![B]\!]_s$ we have three possible scenarios. If $[\![B]\!]_s = tt$, then we have

$$F^{n+1}_{B,C_1}(\bot)(s) = F^n_{B,C_1}(\bot)([\![C_1]\!](s)).$$

Therefore, we can carry out the proof:

$$\begin{aligned}
& t \sqsubseteq F^{n+1}_{B,C_1}(\bot)(s) \\
\Longrightarrow\ & t \sqsubseteq F^n_{B,C_1}(\bot)([\![C_1]\!](s)) \\
\Longrightarrow\ & t \sqsubseteq F^n_{B,C_1}(\bot)([\![C_1]\!](s)) \\
\Longrightarrow\ & \exists r \in \mathbf{K}\Omega\Sigma. r \sqsubseteq [\![C_1]\!](s) \text{ and } t \sqsubseteq F^n_{B,C_1}(\bot)(r) \text{ by same technique as in the previous case} \\
\Longrightarrow\ & \vDash [\phi]\ C_1\ [\theta] \text{ and } t \sqsubseteq F^n_{B,C_1}(\bot)(r) \qquad \text{where } [\![\theta]\!] = \uparrow r \\
\Longrightarrow\ & \vdash [\phi]\ C_1\ [\theta] \text{ and } \vdash [\theta]\ \texttt{while}\ B\ \texttt{do}\ C_1\ [\psi] \quad \text{by outer and inner IH respectively} \\
\Longrightarrow\ & [\phi]\ \texttt{while}\ B\ \texttt{do}\ C_1\ [\psi] \qquad \text{by H-}\texttt{while-tt}
\end{aligned}$$

If $[\![B]\!]_s = ff$, then $F^{n+1}_{B,C_1}(\bot)(s) = s$, so we have $t \sqsubseteq s$. This means $\phi \leq \psi$ and we have the derivation

$$\dfrac{\dfrac{\dfrac{\text{by outer IH}}{[\phi]\ B\ [\texttt{ff}]}}{[\phi]\ \texttt{while}\ B\ \texttt{do}\ C_1\ [\phi]}\ \text{H-}\texttt{while-ff} \qquad \phi \leq \psi}{[\phi]\ \texttt{while}\ B\ \texttt{do}\ C_1\ [\psi]}\ \text{H-}\leq$$

If $[\![B]\!]_s = \bot$, then $[\![\texttt{while}\ B\ \texttt{do}\ C_1]\!] = \bigsqcup_{n\in\mathbb{N}} F^n_{B,C_1}(\bot) = \bigsqcup_{n\in\mathbb{N}} \bot = \bot$ so $t = \bot$. This implies that $\psi \approx 1$, so we can derive the required Hoare triple by H-1.

- ($C = \texttt{def}\ n\ \texttt{:=}\ A$) Since ϕ and ψ are prime, we can use the normal form theorem. If $\psi \approx 1$, then we can immediately apply the H-1 rule. If $\phi \approx 1$, then since $s = \bot$ we have

$$t \sqsubseteq [\![\texttt{def}\ n\ \texttt{:=}\ A]\!](\bot) = \bot\ [n := [\![A]\!]_\bot]$$

If $[\![A]\!]_\bot = \bot$, then $t \sqsubseteq \bot$ so $t = \bot$ in which case $\psi \approx 1$. Otherwise, if $[\![A]\!](\bot) = k$, then $t \sqsubseteq \langle n; k\rangle$. This means $t(m) = \bot$ for $m \neq n$, and either $t(n) = k$ or $t(n) = \bot$. The latter case again means $\psi = 1$, so let us focus on when $t(n) = k$, in which case $t = \langle n; k\rangle$. Then $\psi \approx n \rightsquigarrow k$, so we can apply H-`def`.

Finally, if $\phi \approx \bigwedge_{i\in I}(n_i \rightsquigarrow k_i)$ and $\psi \approx \bigwedge_{j\in J}(n_j \rightsquigarrow k_j)$, then unfolding what we have yields

$$t = \bigsqcup_{j\in J} \langle n_j; k_j\rangle \sqsubseteq [\![\texttt{def}\ n\ \texttt{:=}\ A]\!]\left(s = \bigsqcup_{i\in I} \langle n_i; k_i\rangle\right)$$

As before, we can reason by cases on $[\![A]\!]$. If $[\![A]\!]\left(\bigsqcup_{i\in I} \langle n_i; k_i\rangle\right) = \bot$ then we have a contradiction unless $J = \emptyset$, in which case $\psi = 1$. If $[\![A]\!]\left(\bigsqcup_{i\in I} \langle n_i; k_i\rangle\right) = k$ then

$$[\![\texttt{def}\ n\ \texttt{:=}\ A]\!]\left(\bigsqcup_{i\in I} \langle n_i; k_i\rangle\right) = \langle n; k\rangle \sqcup \left(\bigsqcup_{i\in I - \{i \mid n_i = n\}} \langle n_i; k_i\rangle\right)$$

This means that for any $j \in J$, $n_j = n$ with $k_j = k$ or $n_j = n_i \neq n$ and $k_j = k_i$ for some $i \in I$. We can therefore derive the required hoare triple by combining (using H-$\wedge$) multiple applications of the H-`def` and H-`def`-$\wedge$ rule.

□

With these, we can prove the completeness theorem as a corollary.

Theorem 4.3.5. *1. Let A be an arithmetic expression in* **Imp**. *Then*

$$\vDash [\phi]\ A\ [\alpha] \implies \vdash [\phi]\ A\ [\alpha].$$

2. Let B be a Boolean expression in **Imp**. *Then*

$$\vDash [\phi]\ B\ [\chi] \implies \vdash [\phi]\ B\ [\chi].$$

3. Let C be a Command in **Imp**. *Then*

$$\vDash [\phi]\ C\ [\psi] \implies \vdash [\phi]\ C\ [\psi]$$

Proof. The proof is the same for all three results by alluding to prime completeness. Therefore, we only demonstrate it for commands.

$$\begin{aligned}
& \vDash [\phi]\ C\ [\psi] \\
\implies & \vDash [\bigvee_{i\in I} \phi_i]\ C\ [\bigvee_{j\in J} \psi_j] && \text{By prime decomposition} \\
\implies & \forall i \in I. \exists j \in J. \vDash [\phi_i]\ C\ [\psi_j] \\
\implies & \forall i \in I. \exists j \in J. \vdash [\phi_i]\ C\ [\psi_j] && \text{by prime completeness} \\
\implies & \vdash [\bigvee_{i\in I} \phi_i]\ C\ [\bigvee_{j\in J} \psi_j] && \text{by H-}\leq \text{ and H-}\vee \\
\implies & \vdash [\phi]\ C\ [\psi]
\end{aligned}$$

□

4.4 A Comparison with "Wild" Hoare Logic

It is especially pleasing that despite not explicitly setting out with this goal in mind, the resulting logic we obtain is still in the very recognizable and intuitive form of Hoare logic. Of course, there are some differences with standard Hoare logic. We will compare our program logic $\mathcal{L}_{\mathbf{Imp}}$ with the Hoare logic found in the literature. Specifically, the "Wild" Hoare logic we will compare with is from [Win93, Chapter 6].

The first observation is that the Hoare triples in [Win93] are *partial* correctness assertions, i.e. a Hoare triple $\{\phi\}\ C\ \{\psi\}$ states that starting in a state satisfying ϕ, *if C terminates,* then the end state satisfies ψ. On the other hand, our logic is able to express the concept of termination via the $\bot$ element in the domain. If we have that $\mathsf{T}(\psi)$, then our Hoare triple $[\phi]\ C\ [\psi]$ expresses that starting at state ϕ, C terminates with end state satisfying ψ. This is known as *total* correctness. This may seem like a win, but one disadvantage of our approach is that we cannot *also* reason about partial correctness, because if ψ contains $\bot$ then it has to be the trivial assertion.

Another difference is that the logical language of the Hoare logic in [Win93] is much stronger. It is essentially a full first-order logic with the signature being the variables (of the programming language), and Σ being the first-order structures. Of course, since our logic is a propositional logic and can only express compact open sets, it is far more limited. Our logic does not even have a negation.

Finally, we inspect the rules of their logic. The rule for `skip` and sequencing of commands (`;`) remains the same, however the rule for assignments is quite different. They can afford to be more efficient since they can consider arbitrary substitutions of the precondition by the value, while our logic cannot since it does not even have the ability to express arbitrary arithmetic expressions.

$$\frac{}{\{\phi[A/n]\}\ \texttt{def}\ n\ \texttt{:=}\ A\ \{\phi\}}$$

The rule for conditionals also looks quite different, for by law of excluded middle, and by the expressivity of their assertion language, they are able to incorporate the boolean expression and its negation into the precondition of the components.

$$\frac{\{\phi \wedge B\}\ C_1\ \{\psi\} \qquad \{\phi \wedge \neg B\}\ C_2\ \{\psi\}}{\{\phi\}\ \texttt{if}\ B\ \texttt{then}\ C_1\ \texttt{else}\ C_2\ \{\psi\}}$$

This is stronger than our rule for conditionals. The closest we can get is the following rule, by using the completeness theorem[10]:

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

but this is still weaker since in proving $[\phi]\ C_1\ [\psi]$ we cannot use the fact that B holds, and similarly for $[\phi]\ C_2\ [\psi]$. We cannot make the assertion of states that make B true in our logic, since this may not be compact.

The rule for while loops in their logic is

$$\frac{\{\phi \wedge B\}\ C\ \{\phi\}}{\{\phi\}\ \texttt{while}\ B\ \texttt{do}\ C\ \{\phi \wedge \neg B\}}$$

This works in their logic because they are concerned with partial correctness assertions. The closest approximation we have in our logic would be something of the following form:

$$\frac{[\phi]\ B\ [\texttt{tt},\texttt{ff}] \qquad [\phi]\ C\ [\phi]}{[\phi]\ \texttt{while}\ B\ \texttt{do}\ C\ [\phi]}$$

But this is blatantly unsound: take $C = \texttt{def}\ 3\ \texttt{:=}\ 5$, $B = \texttt{tt}$, and $\phi = (3 \rightsquigarrow 5)$.

[10] to prove this proof-theoretically it is not at all obvious. First, we need to use prime decomposition on ϕ. Then, we need to show that for each prime component ϕ_i, $\vdash [\phi_i]\ B\ [\texttt{tt},\texttt{ff}]$ implies either $\vdash [\phi_i]\ B\ [\texttt{tt}]$ or $\vdash [\phi_i]\ B\ [\texttt{ff}]$. This is somewhat of a poor man's excluded middle. Finally, we apply either the H-ite-tt or H-ite-ff rule appropriately to conclude $[\phi_i]\ \texttt{if}\ B\ \texttt{then}\ C_1\ \texttt{else}\ C_2\ [\psi]$ for every ϕ_i, before putting it together using H-$\vee$.

5
What Next?

We conclude with some questions to further investigate and directions to take this research in.

5.1 The Compact-open Restriction

In the last section of the previous chapter, we compared our logic with standard Hoare logic and found some deficiencies. Some of these deficiencies are rectifiable by considering our program properties to be arbitrary open sets, instead of compact opens. However, to maintain completeness for the corresponding notion of $L(\Sigma)$ we would need arbitrary disjunctions. Can we find a middle ground by restricting the logic just to using $\exists$ instead of arbitrary disjunctions?

5.2 Are Elements of Domains Possible Worlds or Possibilities?

The philosophical interpretation[1] of Stone duality establishes the elements of the space corresponding to a logic as possible worlds. There is an alternative duality (called choice-free Stone duality) due to Bezhanishvili & Holliday [BH19; Hol21] where the elements of the corresponding space are to be interpreted as partial possibilities.

[1] or at least *some* interpretation.

By using Stone duality, are we committing to the idea that the elements of a domain are possible worlds? This makes some sense because we should think of programs as completed things-in-and-of-themselves. At the same time, if we only care about complete states (i.e. maximal elements of $\Sigma_{\mathbf{Imp}}$) then many elements of $\Sigma_{\mathbf{Imp}}$ are superfluous from the denotational point of view. However, from the logical point of view they are very useful: the compact pair functions $\langle n; k\rangle$ are elements that precisely express the property of having variable n be value k. In my opinion, this resembles a partial possibility, more than a possible world.

From a technical point of view, it would be interesting to see if we can establish program logics by applying Choice-free Stone duality to domain theory, and comparing them to the ones obtained by classic Stone duality.

5.3 *Accessible Categories as Generalizations of Algebraic DCPOs*

An obvious generalization of the order-theoretic domains in this report is to consider their categorification. Accessible categories are the categorification of algebraic DCPOs, and recently Ivan Di Liberti established the duality theory (named "Scott adjunction") for algebraic DCPOs and their Scott topos (a generalization of Scott domains).

At the same time, in the study of concurrent programs it has been suggested that models of concurrency should be categorified domains[2] [CW05]. It would be interesting to study logics for concurrency using the Scott adjunction and compare them with existing concurrent logics (e.g. separation logic), in analogy with what we did in this project.

[2] specifically, we should consider certain presheaf categories on categories of program execution path as a domain.

Bibliography

[Abr91] S. Abramsky. "Domain theory in logical form*". In: *Annals of Pure and Applied Logic* 51.1 (1991), pp. 1–77. ISSN: 0168-0072. DOI: https://doi.org/10.1016/0168-0072(91)90065-T. URL: https://www.sciencedirect.com/science/article/pii/016800729190065T.

[AJ95] S. Abramsky and A. Jung. "Domain Theory". In: *Handbook of Logic in Computer Science (Vol. 3): Semantic Structures*. USA: Oxford University Press, Inc., 1995, pp. 1–168. ISBN: 019853762X.

[BH19] N. Bezhanishvili and W. H. Holliday. "Choice-free Stone Duality". In: *The Journal of Symbolic Logic* 85.1 (Aug. 2019), pp. 109–148. ISSN: 1943-5886. DOI: 10.1017/jsl.2019.11. URL: http://dx.doi.org/10.1017/jsl.2019.11.

[Bon98] M. M. Bonsangue. *Topological Duality in Semantics*. Vol. 8. Electronic Notes in Theoretical Computer Science. 1998. URL: https://www.sciencedirect.com/science/journal/15710661/8.

[CW05] G. L. Cattani and G. Winskel. "Profunctors, open maps and bisimulation". In: *Mathematical Structures in Computer Science* 15.3 (2005), pp. 553–614. DOI: 10.1017/S0960129505004718.

[Hol21] W. H. Holliday. "Possibility Semantics". In: *Selected Topics from Contemporary Logics*. Ed. by M. Fitting. College Publications, 2021, pp. 363–476.

[HP90] H. Huwig and A. Poigné. "A note on inconsistencies caused by fixpoints in a cartesian closed category". In: *Theoretical Computer Science* 73.1 (1990), pp. 101–112. ISSN: 0304-3975. DOI: https://doi.org/10.1016/0304-3975(90)90165-E. URL: https://www.sciencedirect.com/science/article/pii/030439759090165E.

[Joh82] P. Johnstone. *Stone Spaces*. Cambridge Studies in Advanced Mathematics. Cambridge University Press, 1982. ISBN: 9780521337793. URL: https://books.google.nl/books?id=CiWwoLNbpykC.

[Plo83] G. D. Plotkin. *Domains*. 1983. URL: https://homepages.inf.ed.ac.uk/gdp/publications/Domains_a4.ps.

[SLG94] V. Stoltenberg-Hansen, I. Lindström, and E. R. Griffor. *Mathematical Theory of Domains*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1994. DOI: 10.1017/CBO9781139166386.

[SVng] G. Sambin and S. Valentini. "Topological Characterization of Scott Domains". In: *Archive for Mathematical Logic* (forthcoming). URL: https://www.math.unipd.it/~silvio/papers/WorkInProg/TopologicChar.pdf.

[Vic89] S. Vickers. *Topology via Logic*. USA: Cambridge University Press, 1989. ISBN: 0521360625.

[Win93] G. Winskel. *The Formal Semantics of Programming Languages: An Introduction*. Foundations of Computing. MIT Press, 1993. ISBN: 9780262731034. URL: https://books.google.nl/books?id=oL9NEAAAQBAJ.