Domain Theory

Part 4: PCF, then Scott Domains

Dr. Liam O'Connor

based on material from

Graham Hutton, John Longley, Robert Muller, Dana Scott, Joseph E. Stoy, Carl Gunter, Glynn Winskel

March 18, 2024

1 Introduction

Previously we saw a formalisation of constructions on the category **Cpo**, whose objects are cpos and morphisms are continuous functions between them. These constructions allow us to give a denotational semantics to a more interesting language, called PCF, a variant of typed λ-calculus.

Syntax

The syntax of PCF consists of types (τ) and expressions (e), given below. A context, written Γ, consists of a sequence of typing judgements for variables:

$$\begin{array}{lcl} e & ::= & n \mid x \mid \lambda x : \tau.\ e \mid e_1\ e_2 \mid \mathsf{succ} \mid \mathsf{pred} \mid \mathsf{ifz}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3 \mid \mathsf{fix}\ x : \tau.\ e \\ \tau & ::= & \mathsf{nat} \mid \tau_1 \to \tau_2 \end{array}$$

The typing rules are given below.

$$\frac{n \in \mathbb{N}}{\Gamma \vdash n : \mathsf{nat}} \quad \frac{x : \tau \in \Gamma}{\Gamma \vdash x : \tau} \quad \frac{\Gamma, x : \tau_1 \vdash e : \tau_2}{\Gamma \vdash \lambda x : \tau_1.\ e : \tau_1 \to \tau_2} \quad \frac{\Gamma \vdash e_1 : \tau_1 \to \tau_2 \qquad \Gamma \vdash e_2 : \tau_1}{\Gamma \vdash e_1\ e_2 : \tau_2}$$

$$\frac{}{\Gamma \vdash \mathsf{succ} : \mathsf{nat} \to \mathsf{nat}} \quad \frac{}{\Gamma \vdash \mathsf{pred} : \mathsf{nat} \to \mathsf{nat}} \quad \frac{\Gamma \vdash e_1 : \mathsf{nat} \qquad \Gamma \vdash e_2 : \tau \qquad \Gamma \vdash e_3 : \tau}{\Gamma \vdash \mathsf{ifz}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3 : \tau}$$

$$\frac{\Gamma, x : \tau \vdash e : \tau}{\Gamma \vdash \mathsf{fix}\ x : \tau.\ e : \tau}$$

Because PCF is typed, we shall assign denotations only to well-typed expressions. The range of our denotation function for expressions is determined by the type of its input expression. That is, the denotation of a type τ *is* the cpo whose elements are the denotations of expressions of type τ.

The denotation of the type nat, then, is merely the flat domain (i.e. lifted cpo) from the natural numbers $\mathbb{N}_\perp$:

$$[\![\mathsf{nat}]\!] = \mathbb{N}_\perp$$

And, the denotation of the function type is the domain of continuous functions on cpos:

$$[\![\tau_1 \to \tau_2]\!] = [\![\tau_1]\!] \rightarrow [\![\tau_2]\!]$$

A closed term $e : \tau$ with no free variables simply denotes an element of $[\![\tau]\!]$, so we might be tempted to define our denotation function for expressions $e : \tau$ like so:

$$[\![e]\!] : [\![\tau]\!]$$

However, if $e : \tau$ involves free variables from our context Γ, the valuation of e will clearly depend on the values assigned to all the variables. Thus, the denotation of a typed expression $\Gamma \vdash e : \tau$ is defined instead as a continuous function:

$$[\![e]\!]_\Gamma : [\![\Gamma]\!] \to [\![\tau]\!]$$

where the meaning of a context $\Gamma = (x_1 : \tau_1, x_2 : \tau_2, \dots, x_n : \tau_n)$ will be a a big product of the values assigned to each variable:

$$[\![\Gamma]\!] = [\![\tau_1]\!] \times [\![\tau_2]\!] \times \cdots \times [\![\tau_n]\!]$$

The actual definition of our expression semantics is then:

$$\begin{array}{lcl}
[\![n]\!]_\Gamma(\vec{z}) & = & n \\
[\![x]\!]_\Gamma(\vec{z}) & = & z_j \text{ where } j \text{ is largest } j \text{ s.t. } x = x_j \\
[\![\lambda x : \tau_1.\ e]\!]_\Gamma(\vec{z}) & = & (\Lambda v \in [\![\tau_1]\!].\ [\![e]\!]_{\Gamma, x:\tau_1}(\vec{z}, v)) \\
[\![e_1\ e_2]\!]_\Gamma(\vec{z}) & = & [\![e_1]\!]_\Gamma(\vec{z})([\![e_2]\!]_\Gamma(\vec{z})) \\
[\![\mathsf{succ}]\!]_\Gamma(\vec{z}) & = & (\Lambda v \in \mathbb{N}_\perp.\ v + 1) \\
[\![\mathsf{pred}]\!]_\Gamma(\vec{z}) & = & (\Lambda v \in \mathbb{N}_\perp.\ v - 1) \\
[\![\mathsf{ifz}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3]\!]_\Gamma(\vec{z}) & = & \begin{cases} [\![e_2]\!]_\Gamma(\vec{z}) & \text{if } [\![e_1]\!]_\Gamma(\vec{z}) > 0 \\ [\![e_3]\!]_\Gamma(\vec{z}) & \text{if } [\![e_1]\!]_\Gamma(\vec{z}) = 0 \\ \perp & \text{if } [\![e_1]\!]_\Gamma(\vec{z}) = \perp \end{cases} \\
[\![\mathsf{fix}\ x : \tau.\ e]\!]_\Gamma(\vec{z}) & = & \mathbf{fix}(\Lambda v \in [\![\tau]\!].\ [\![e]\!]_{\Gamma, x:\tau}(\vec{z}, v))
\end{array}$$

Here $\Lambda x \in C.\ t$ is notation for an anonymous continuous function $C \to D$ between cpos C and D. Verifying that these functions are indeed continuous is straightforward, but is necessary to justify the use of **fix**.

Are we done?

For most purposes in semantics, describing semantics in terms of (continuous functions on) cpos is enough. The above semantics of PCF is evidence of this: PCF is a Turing-complete, higher order functional programming language. This semantic approach generalises to many languages with richer type systems. Thus, just the content we have covered so far is sufficient to give denotational semantics to many kinds of languages!

If we *discard* the type system, **fix**, natural number primitives and any other superfluous features, and just boil our language down to a minimal, Turing-complete subset, we end up with the untyped λ-calculus — a language consisting only of untyped functions:

$$e \ ::= \ x \mid \lambda x.e \mid e_1\ e_2$$

Trying to give a semantics to the untyped λ-calculus poses an issue: we can no longer rely on the type of an expression to select an appropriate semantic domain. Instead, we we must pick a *single* domain D which, since functions can be applied to themselves, must apparently include the set of functions $D \to D$. By a simple cardinality argument, such an inclusion cannot hold. Instead, we shall insist on an *isomorphism*[1]. between our domain D and our space of functions $(D \to D)$. Even if we restrict D to be a cpo and our functions to be continuous, D may not admit such an isomorphism. We must further strengthen our requirement on D, instead requiring that it is a particular type of cpo called a Scott domain. This lecture will introduce the key concepts behind Scott domains.

[1]Technically this is a slight weakening called a retraction pair, but we will reach this in the next part of the course.

Later Connections

We will see how to construct such recursively defined domains work in general in the next part of the course. Scott domains are also necessary to give semantics to non-deterministic programs through a construction called the *power domain*, which we will discuss towards the end of the course.

2 Compactness

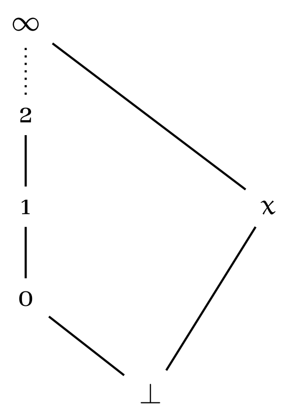
We begin by formalising the notion of an element in a cpo representing a finite amount of information.

Intuition

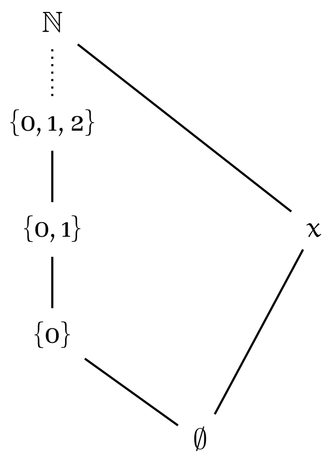
There are two kinds of directed sets:

- **Boring** sets contain their lub.
  Example: Finite directed sets are boring (but boring sets aren't all finite!).
- **Interesting** sets don't contain their lub.
  Example: $\mathbb{N}$ in the chain $\mathbb{N} \cup \{\infty\}$ is interesting.

Following the above intuition, we might be tempted to say that the *infinite elements* of a cpo are those which are the lub of an interesting set, but this notion is too weak. Consider this cpo X:



By the above definition, the only infinite element would be ∞, but if we consider the following isomorphic cpo, ordered by subset inclusion:



Then the set x cannot be a finite set, as any finite set would be a subset of one of the finite sets in the chain $\emptyset \sqsubseteq \{0\} \sqsubseteq \{0,1\} \sqsubseteq \cdots$. Thus, it makes more sense for us to call x an *infinite element* as well.

Definition

Let A be a cpo. Then $x \in A$ is compact (a.k.a. *finite*) iff for all directed $X \subseteq A$:

$$x \sqsubseteq \bigsqcup X \implies \exists y \in X.\, x \sqsubseteq y$$

In English: A compact element will approximate some element of a directed set if it approximates the lub. We write $K(A)$ for the set $\{x \in A \mid x \text{ is compact}\}$.

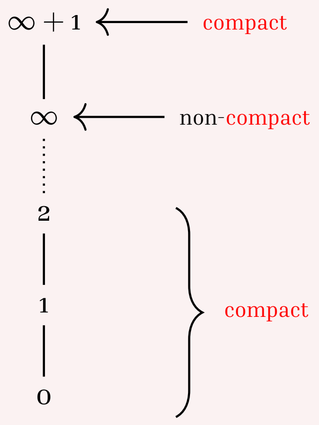
In the example cpo X above, all the elements except ∞ and x would be compact. Thus, compactness better captures our notion of an element representing a finite amount of information. This understanding of compact elements is a generalisation of the notion of a finite element from the theory of algebraic lattices.

Example

- Every element in a finite cpo is compact. More generally, every element of a cpo of finite *height* (e.g. $\mathbb{Z}_\perp$) is compact. This is because finite directed sets always contain their lub.
- The cpo $\mathcal{P}(X)$ of subsets of X ordered by inclusion $\subseteq$. The compact elements of $\mathcal{P}(X)$ are those of finite cardinality.
- The cpo $\mathbb{N} \rightharpoonup \mathbb{N}$ of partial functions on the natural numbers, ordered by inclusion of graphs. The compact elements of $\mathbb{N} \rightharpoonup \mathbb{N}$ are the functions which are defined only for finite domains.

Examples Contrary to Intuition (Infinite height)

Compact elements may still have an infinite number of approximations. Consider the cpo $\mathbb{N} \cup \{\infty, \infty + 1\}$, where we have tacked on an additional top element $\infty + 1$ to our normal cpo of natural numbers extended with infinity. Then, while ∞ is not compact, $\infty + 1$ is compact — it is not the lub of an interesting directed set. If $\infty + 1$ is the lub of a set X then $\infty + 1$ must be in the set X. Nonetheless, there are an infinite number of *approximations* to $\infty + 1$, i.e., elements x such that $x \sqsubseteq \infty + 1$.



As an aside, requiring that our compact elements have a truly finite number of approximations is the basis for the theory of Berry domains and stable functions (outside the scope of this course).

Examples Contrary to Intuition (Infinite sets can be compact)

A *submonoid* of a monoid $(X, \oplus, \iota)$ is a subset $Y \subseteq X$ such that Y is closed under $\oplus$ and $\iota \in Y$.

Fact: The submonoids of a X form a cpo under $\subseteq$, where union gives the lub.

Contrary to our intuition of compactness meaning finiteness, the *infinite* set E of even natural numbers is nonetheless a compact element in the cpo of submonoids of $(\mathbb{N}, +, 0)$.

Proof: Let $Y \subseteq \mathbb{N}$ be directed and $E \subseteq \bigcup Y$. Since $2 \in E$ there must exist $y \in Y$ such that $2 \in Y$. Since $(y, +, 0)$ is a monoid, $E \subseteq y$.

However, E is *finitely generated* – it is the smallest submonoid of $(\mathbb{N}, +, 0)$ such that the *finite* set $\{2\} \subseteq E$. In fact, the compact submonoids of $(\mathbb{N}, +, 0)$ are precisely the finitely generated ones.

3 Algebraicity

In order to achieve our desired domain $D \simeq D \rightarrow D$, the cpo we choose must be algebraic.

Definition

A cpo is algebraic iff every element is the lub of its compact approximations. That is, a cpo D is algebraic iff for all $x \in D$,

1. $\downarrow(x) = \{y \in K(D) \mid y \sqsubseteq x\}$ is directed;
2. $x = \bigsqcup \downarrow(x)$

Note: It's common in semantics to consider only algebraic cpos with a *countable* set of compact elements. Such cpos are called ω-algebraic.

Aside: Plotkin provides an equivalent definition of ω-algebraic cpos in which directed sets are replaced with ω-chains throughout. Hutton claims this is less appealing, as the definition speaks of *an* ω-chain of compact approximations, rather than *the* directed set of such.

Example

Many of the examples we saw above are ω-algebraic cpos: finite cpos, subsets $\mathcal{P}(X)$ of a set X, partial functions $\mathbb{N} \rightharpoonup \mathbb{N}$ and submonoids of a monoid. In general, any cpo of subalgebras (e.g. subgroups of a group, subrings of a ring etc.) is ω-algebraic. This is the origin of the terminology.

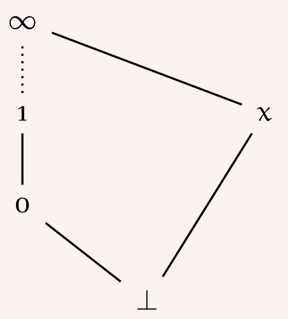
Counterexample

In the cpo on the right (the same as the one we saw previously), the element x is not the lub of its compact approximations. The only compact element that approximates x is $\bot$, and the least upper bound of the set $\{\bot\}$ is just $\bot$, not x. That is:

$$\downarrow(x) = \{\bot\}$$

but:

$$x \neq \bigsqcup\{\bot\}$$



3.1 Continuous functions on Algebraic Cpos

Nothing Suddenly Invented at Infinity

Let D and E be algebraic cpos. Then a function $f : D \to E$ is continuous iff, for all $x \in D$:

$$f(x) = \bigsqcup \{f(a) \mid a \in {\downarrow}(x)\}$$

In other words, in an algebraic cpo, continuous functions are completely defined by their behaviour for compact arguments.

This makes precise our earlier slogan that continuous functions don't suddenly behave differently for infinite (i.e. non-compact) elements.

Theorem

Let $d : D \rightarrow E$ be a continuous function between algebraic cpos. Define:

$$G_f = \{(a, b) \in K(D) \times K(E) \mid b \sqsubseteq f(a)\}$$

Then for all $x \in D$, we have:

$$f(x) = \bigsqcup \{b \mid (a, b) \in G_f \wedge a \sqsubseteq x\}$$

This is powerful. For example, the continuous function $f : \mathcal{P}(\mathbb{N}) \rightarrow \mathcal{P}(\mathbb{N})$ on an *uncountable* cpo $\mathcal{P}(\mathbb{N})$ is completely determined by the *countable* relation G_f.

Proof:

$$\begin{array}{rcll}
f(x) & = & f(\bigsqcup {\downarrow}(x)) & \text{(D is algebraic)} \\
 & = & \bigsqcup \{f(a) \mid a \in {\downarrow}(x)\} & \text{(f is continuous)} \\
 & = & \bigsqcup \{\bigsqcup {\downarrow}(f(a)) \mid a \in {\downarrow}(x)\} & \text{(E is algebraic)} \\
 & = & \bigsqcup \{b \in K(E) \mid b \sqsubseteq f(a) \wedge a \in {\downarrow}(x)\} & \text{(lubs and defn of } {\downarrow}) \\
 & = & \bigsqcup \{b \mid (a, b) \in G_f \wedge a \sqsubseteq x\} & \text{(defn of } G_f)
\end{array}$$

3.2 A Representation Theorem for Algebraic Cpos

Definition

An ideal is a downwards-closed directed set. The ideal completion of a set A, written sometimes as $\overline{A}$, is the set of all ideal subsets of A, i.e.:

$$\overline{A} = \{X \subseteq A \mid X \text{ is ideal}\}$$

Theorem: Every algebraic cpo D is isomorphic to the ideal completion of its compact elements, i.e.:

$$\overline{K(D)} = \{X \subseteq K(D) \mid X \text{ is ideal}\} \text{ is an algebraic cpo such that } D \simeq \overline{K(D)}$$

Proof omitted.

3.3 Closure Properties

If D and E are algebraic cpos, then so is their product construction $D \times E$. The following definition is useful for the proof:

Definition

A set $X \subseteq K(A)$ is a basis for a cpo A iff for all $x \in A$, $x = \bigsqcup\{a \in X \mid a \sqsubseteq x\}$.

If X is a basis for A, then A is algebraic and $K(A) = X$.
Proof: If $a \in K(A)$, then $\bigsqcup M = a$ where $M = \{x \in X \mid x \sqsubseteq a\}$, as X is a basis. Since a is compact, $a \sqsubseteq b$ for some $b \in M$. But since a is the lub of M, $b \sqsubseteq a$ as well. By antisymmetry $a = b$, hence $a \in X$. Thus $K(A) \subseteq X$, so $K(A) = X$ and A is algebraic.

In the following proof, we show that $K(D) \times K(E)$ is a basis for the product $D \times E$ and thereby show that $D \times E$ is algebraic if D and E are.

Part 1: $K(D) \times K(E) \subseteq K(D \times E)$

Let $(x, y) \in K(D) \times K(E)$. To show that (x, y) is compact, let us assume that $(x, y) \in \bigsqcup X$ where $X \subseteq D \times E$ is directed. We must show that there exists some element e of X such that our $(x, y) \sqsubseteq e$. As $(x, y) \in \bigsqcup X$, by the definition of lub on products we can conclude that[2]:

$$\begin{array}{rcl} x & \sqsubseteq & \bigsqcup \pi_0[X] \\ y & \sqsubseteq & \bigsqcup \pi_1[X] \end{array}$$

Since x and y are both compact, there must exist $x' \in \pi_0[X]$ and $y' \in \pi_1[X]$ such that $x \sqsubseteq x'$ and $y \sqsubseteq y'$. While it does not follow that $(x', y') \in X$, we know that there must exist a pair $(a, b) \in X$ such that $x' \sqsubseteq a$ and $y' \sqsubseteq b$ as X is directed. Hence (a, b) can be our element $e \in X$ that is approximated by (x, y), i.e. $(x, y) \sqsubseteq (a, b)$.

Part 2: $\downarrow(x, y)$ is directed for all $(x, y) \in D \times E$

$$\begin{array}{rcl} \downarrow(x, y) & = & \{(a, b) \in K(D) \times K(E) \mid (a, b) \sqsubseteq (x, y)\} \\ & = & \{a \in K(D) \mid a \sqsubseteq x\} \times \{b \in K(E) \mid b \sqsubseteq y\} \\ & = & \downarrow(x) \times \downarrow(y) \end{array}$$

Because D and E are algebraic, $\downarrow(x)$ and $\downarrow(y)$ are directed. As directedness is closed under product, $\downarrow(x, y)$ is directed too.

Part 3: $(x, y) = \bigsqcup \downarrow(x, y)$

Starting from the right hand side:

$$\begin{array}{rcll} \bigsqcup \downarrow(x, y) & = & \bigsqcup(\downarrow(x) \times \downarrow(y)) & \text{(part 2)} \\ & = & (\bigsqcup(\downarrow(x), \bigsqcup \downarrow(y))) & \text{(lub on products)} \\ & = & (x, y) & \text{(D, E are algebraic)} \end{array}$$

Problem

ω-algebraic cpos are closed under all of our cpo constructions *except* $\rightarrow$ and $\circ\!\!\rightarrow$!

4 Consistent Completeness

The lack of closure under the (continuous) function arrow, strict or non-strict, is not satisfying as it means that our semantic domains are not guaranteed to be algebraic even if they are composed from algebraic cpos. Instead, we must replace algebraic cpos with something stronger still. Multiple solutions exist.

[2]Here we use the notation $f[X]$ to indicate the image of a function on a set, i.e. $\{f(v) \mid v \in X\}$.

Original Solution

Scott's original solution to this lack of closure was to use complete lattices instead of cpos, i.e. requiring lubs for all subsets, not just directed ones. This solves the problem with $\rightarrow$ but introduces new problems:

1. Complete lattices need a top element $\top$, but adding a fictitious top (representing inconsistent information) to cpos like $\mathbb{B}_\bot$ is strange.

2. Extending the functions that capture our primitive semantic operations to complete lattices can spoil nice algebraic properties. Consider these two possible implementations of ite, the function for the semantics of an if expression:

$$\mathsf{ite}(\top, x, y) = x \sqcup y \qquad\qquad \mathsf{ite}(\top, x, y) = \top$$

   Either of these solutions results in the failure of useful and expected laws for if expressions. For example, the left definition above results in the failure of the common equation to eliminate unreachable cases:

$$\mathsf{ite}(b, \mathsf{ite}(b, x, y), z) = \mathsf{ite}(b, x, z)$$

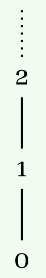
3. The power domain construction, seen later in the course, does not generalise to complete lattices, so semantics for non-deterministic programs are difficult in this setting.

4.1 The Real Solution

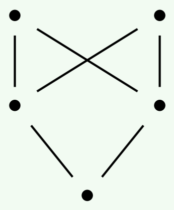
Definition

A poset A is consistent complete (or *bounded complete*) iff $\bigsqcup X$ exists for all consistent $X \subseteq A$. That is, any set with *an* upper bound (a consistent set) has a *least* upper bound.

Example



consistent complete
not directed complete



not consistent complete
directed complete

By adding the requirement that our cpo be consistent complete, we can ensure that our semantic domains are closed under all our cpo constructions, including $\rightarrow$ and $\multimap$

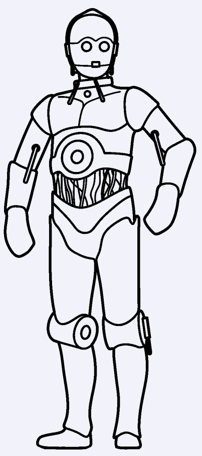
5 Scott Domains

Definition

A cpo D is a Scott domain iff:

1. D is ω-algebraic
2. D is consistent complete

In other words, Scott domains can be summed up by the acronym:



$$\begin{tikzcd}[column sep=small] & \text{ac}^3\text{po} \arrow[dl, no head] \arrow[d, no head] \arrow[dr, no head] & \\ \omega\text{-algebraic} & \text{consistent-complete} & \text{complete partial order} \end{tikzcd}$$

The second requirement in the definition above can, in light of the first, be expressed equivalently as: $x \sqcup y$ exists for all consistent $x, y \in D$. This is useful when needing to show that a given cpo is a Scott domain.

Scott domains are closed under all our cpo constructions, including $\to$ and $\multimap$.

Thesis

Our semantic domains are Scott domains.

Glossary

ω-algebraic An algebraic cpo where the compact elements are countable.. 5, 7, 9, 10

algebraic A cpo is algebraic iff every element is the lub of its compact approximations. That is, a cpo D is algebraic iff for all $x \in D$,

1. $\downarrow(x) = \{y \in K(D) \mid y \sqsubseteq x\}$ is directed;
2. $x = \bigsqcup \downarrow(x)$

. 5–7, 9

basis A set $X \subseteq K(A)$ is a basis for a cpo A iff for all $x \in A$, $x = \bigsqcup\{a \in X \mid a \sqsubseteq x\}$. 7, 9

closed term A term, or expression, with no free variables. 1

compact An element x in a cpo A is *compact* (a.k.a. finite) iff for all directed $X \subseteq A$, $x \sqsubseteq \bigsqcup X \implies \exists y \in X.\ x \sqsubseteq y$. That is, if a compact element approximates the lub of a directed set, it will approximate an element of that set. . 4–7, 9

complete lattice A complete lattice is the same as a cpo, except all sets have a least upper bound, not just the directed ones. . 8

consistent A subset X of a poset A is *consistent* iff it has an upper bound. An *upper bound* of a set Y is some x such that $\forall y \in Y. y \sqsubseteq x$. 8–10

consistent complete A poset A is *consistent complete* (or *bounded complete*) iff $\bigsqcup X$ exists for all consistent $X \subseteq A$. That is, any set with *an* upper bound (a consistent set) has a *least* upper bound. 8–10

context A finite sequence of typing assumptions for each variable in scope $x_0 : \tau_0, x_1 : \tau_1, x_2 : \tau_2, \ldots, x_n : \tau_n$. Often a context is written as Γ. 1, 2, 10

downwards-closed A set X is downwards-closed if $\{y \mid y \sqsubseteq x \wedge x \in X\} = X$. 6, 10

free variables A variable is *free* in an expression e if it is not bound (introduced) within e. For example, in the untyped λ-calculus expression $\lambda x.\ x\ y$, y is free and x is not. . 1, 2, 9

ideal A downwards-closed directed set. 6, 10

ideal completion The set of all ideal subsets of A, often written $\overline{A}$, i.e. $\overline{A} = \{X \subseteq A \mid X \text{ is ideal}\}$. 6

PCF A Turing-complete variant of the typed λ-calculus. 1, 2

Scott domain An ω-algebraic, consistent complete cpo. 2, 3, 9

typed λ-calculus The typed lambda calculus is a programming language developed first by Alonzo Church, which associates types to each variable and expression.. 1, 10

typing rules Inference rules for the judgements $\Gamma \vdash e : \tau$, which states that, under the typing context Γ, the expression e has type τ. . 1

untyped λ-calculus The untyped lambda calculus is a very minimal Turing-complete programming language invented by Alonzo Church, consisting only of function abstractions ($\lambda x.e$), function applications ($e_1\ e_2$) and variables. . 2, 10