Introduction to Program Semantics — Dexter Kozen

Lecture 3 - Domain Theory *June 26, 2026*

1 Motivation and Origin

Denotational semantics was introduced by Dana Scott and Christopher Strachey in the 1960s. The guiding idea is: A program should *denote* a mathematical object, rather than merely being *described* by how it executes.

In operational semantics, we explained a program by the steps it takes. In denotational semantics we instead assign to each program c a meaning $[\![c]\!]$ living in some mathematical space. We already saw a primitive instance of this: an IMP program was interpreted as a binary relation $[\![c]\!] \subseteq \text{Env} \times \text{Env}$.

The central technical problem of this lecture is: how do we assign a meaning to a `while` loop? A loop's meaning is naturally described by a recursive equation (it equals itself unwound once), and solving such equations is exactly what the theory of fixpoints provides.

2 Inductive Definitions as Least Fixpoints

A huge amount of computer science is defined inductively: abstract syntax trees, small- and big-step rules, the reflexive–transitive closure of a relation, and so on. The common pattern is: a set $A \subseteq S$ is built by repeatedly applying a *set operator* $R : 2^S \to 2^S$ that produces new elements from old.

Example 1 (Transitive closure)**.** For a binary relation B, its transitive closure B^* is the least set closed under

$$(x, y) \in B^* \ \wedge \ (y, z) \in B^* \implies (x, z) \in B^*.$$

The corresponding operator is

$$R(C) \ = \ B \cup \{(x, z) \mid (x, y) \in C, \ (y, z) \in C\}.$$

2.1 What "the set defined by R" should mean

Given an operator R, the set A it defines should satisfy three conditions.

(1) **R-closed:** $R(A) \subseteq A$.
Everything the rules say belongs to A really is in A.

(2) **R-consistent:** $A \subseteq R(A)$.
Nothing is in A unless some rule of R justifies it.

(3) **Least:** A is $\subseteq$-minimal among the sets satisfying (1).

Conditions (1) and (2) together say $A = R(A)$, i.e. A is a *fixpoint* of R. But a fixpoint need not be unique, so we add (3): we want the *least* fixpoint.

2.2 Monotone operators have a least fixpoint

Definition 1. $R : 2^S \to 2^S$ is *monotone* if $B \subseteq C \Rightarrow R(B) \subseteq R(C)$.

Theorem 1. *Every monotone $R : 2^S \to 2^S$ has a unique least fixpoint.*

Proof. We define the least fixpoint "from above" as the intersection of all R-closed sets. Let

$$\mathrm{cl}\, R \;=\; \{\, B \subseteq S \mid R(B) \subseteq B \,\}, \qquad A^* \;=\; \bigcap \mathrm{cl}\, R.$$

We claim A^* is the least fixpoint.

Step 1: $R(A^*) \subseteq A^*$. For every $B \in \mathrm{cl}\, R$ we have $A^* \subseteq B$, hence by monotonicity $R(A^*) \subseteq R(B) \subseteq B$. Since B was arbitrary, $R(A^*) \subseteq \bigcap \mathrm{cl}\, R = A^*$. In particular $A^* \in \mathrm{cl}\, R$.

Step 2: $A^* \subseteq R(A^*)$. Applying R to Step 1 gives $R(R(A^*)) \subseteq R(A^*)$, so $R(A^*) \in \mathrm{cl}\, R$. But A^* is contained in every member of $\mathrm{cl}\, R$, hence $A^* \subseteq R(A^*)$.

Therefore $A^* = R(A^*)$. It is the *least* fixpoint because every fixpoint is in particular R-closed, hence a member of $\mathrm{cl}\, R$, and A^* lies below all of them. □

2.3 Building the fixpoint "from below"

The construction above is elegant but non-constructive. There is a complementary, more computational picture: iterate R starting from the empty set,

$$\emptyset \subseteq R(\emptyset) \subseteq R(R(\emptyset)) \subseteq R^3(\emptyset) \subseteq \cdots,$$

and take the union. This works because the iterates form a *chain*: a family of sets linearly ordered by $\subseteq$.

Remark. For a general monotone R this ascending construction may need to run through *transfinite ordinals* before it stabilizes. The operators arising in programming-language semantics, however, typically enjoy a stronger property—*chain-continuity*—which guarantees convergence after at most ω steps.

3 The Knaster–Tarski Theorem

3.1 A minimum of ordinals

We only need a few facts about the ordinals $0, 1, 2, \ldots, \omega, \omega+1, \ldots$:

(1) There are two kinds: *successors* $(\alpha+1)$ and *limits* (everything else, e.g. 0 and ω).

(2) They are *well-ordered*: every nonempty set of ordinals has a least element.

(3) There are arbitrarily many of them (more than the cardinality of any fixed set).

(4) One may perform *induction* over them, since $\leq$ is well-founded.

3.2 The transfinite iteration

Definition 2. For monotone $R : 2^S \to 2^S$ define, by transfinite recursion,

$$A_0 = \emptyset, \qquad A_{\alpha+1} = R(A_\alpha), \qquad A_\beta = \bigcup_{\alpha<\beta} A_\alpha \quad (\beta \text{ a limit ordinal}).$$

Lemma 1. *The A_α form a chain, and $\alpha \leq \beta \Rightarrow A_\alpha \subseteq A_\beta$ (the map $\alpha \mapsto A_\alpha$ is monotone).*

Theorem 2 (Knaster–Tarski)**.** *There is a smallest ordinal κ with $A_{\kappa+1} = A_\kappa$, and A_κ is the least fixpoint of R.*

Why it must terminate: the A_α are increasing subsets of S; if they never stabilized they would form a strictly increasing transfinite sequence, giving more distinct sets than S has subsets—impossible by fact (3) above.

3.3 Continuity: convergence in ω steps

Definition 3. A set operator $R : 2^S \to 2^S$ is

- *monotone* if $B \subseteq C \Rightarrow R(B) \subseteq R(C)$;
- *(chain-)continuous* if for every nonempty chain $\mathcal{C}$,

$$R\Big(\bigcup \mathcal{C}\Big) = \bigcup \{\, R(B) \mid B \in \mathcal{C} \,\};$$

- *finitary* if for every set C,

$$R(C) = \bigcup \{\, R(B) \mid B \subseteq C,\ B \text{ finite} \,\}.$$

Lemma 2.

(i) Every chain-continuous operator is monotone.

(ii) An operator is chain-continuous iff it is finitary.

Proof of (i). Suppose $A \subseteq B$. Then $A \cup B = B$, so applying continuity to the two-element chain $\{A, B\}$,

$$R(A) \cup R(B) = R(A \cup B) = R(B),$$

which forces $R(A) \subseteq R(B)$. □

Theorem 3. *If R is chain-continuous, its least fixpoint is reached by step ω, i.e. $A_{\omega+1} = A_\omega$.*

Proof. Using Definition 2 and continuity,

$$A_{\omega+1} = R(A_\omega) = R\left(\bigcup_{n<\omega} A_n\right) = \bigcup_{n<\omega} R(A_n) = \bigcup_{n<\omega} A_{n+1} = A_\omega.$$ □

4 Denotational Semantics of IMP

4.1 Syntax

$$\begin{array}{lcl} \text{AExp} \ni a & ::= & n \mid x \mid a_0 + a_1 \mid \cdots \\ \text{BExp} \ni b & ::= & \mathsf{true} \mid \mathsf{false} \mid \neg b \mid b_0 \wedge b_1 \mid a_0 = a_1 \mid \cdots \\ \text{Com} \ni c & ::= & \mathsf{skip} \mid x := a \mid c_0; c_1 \mid \mathsf{if}\ b\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2 \mid \mathsf{while}\ b\ \mathsf{do}\ c \end{array}$$

An *environment* (state, valuation) is a function $s : \text{Var} \to \mathbb{N}$; the set of environments is Env.

4.2 Semantic functions

We interpret expressions as *total* functions:

$$\mathcal{A}[\![a]\!] : \mathrm{Env} \to \mathbb{N}, \qquad \mathcal{B}[\![b]\!] : \mathrm{Env} \to \mathbb{B}, \qquad \mathcal{C}[\![c]\!] : \mathrm{Env} \to \mathrm{Env}_\perp,$$

where $\mathbb{B} = \{\mathsf{true}, \mathsf{false}\}$ and $\mathrm{Env}_\perp = \mathrm{Env} \cup \{\perp\}$. The symbol $\perp$ ("bottom") means *undefined*. A command might fail to terminate, so $\mathcal{C}[\![c]\!]$ would naturally be a *partial* function $\mathrm{Env} \rightharpoonup \mathrm{Env}$; instead of leaving $\mathcal{C}[\![c]\!]\, s$ undefined we set $\mathcal{C}[\![c]\!]\, s = \perp$.

4.3 Expressions (by structural induction)

Arithmetic:

$$\mathcal{A}[\![n]\!]\, s = n, \qquad \mathcal{A}[\![x]\!]\, s = s(x), \qquad \mathcal{A}[\![a_1 + a_2]\!]\, s = \mathcal{A}[\![a_1]\!]\, s + \mathcal{A}[\![a_2]\!]\, s.$$

Boolean:

$$\mathcal{B}[\![\mathsf{true}]\!]\, s = \mathsf{true}, \qquad \mathcal{B}[\![\mathsf{false}]\!]\, s = \mathsf{false},$$

$$\mathcal{B}[\![\neg b]\!]\, s = \begin{cases} \mathsf{true} & \text{if } \mathcal{B}[\![b]\!]\, s = \mathsf{false} \\ \mathsf{false} & \text{if } \mathcal{B}[\![b]\!]\, s = \mathsf{true} \end{cases} \qquad \mathcal{B}[\![b_1 \wedge b_2]\!]\, s = \begin{cases} \mathsf{true} & \text{if } \mathcal{B}[\![b_1]\!]\, s = \mathcal{B}[\![b_2]\!]\, s = \mathsf{true} \\ \mathsf{false} & \text{otherwise} \end{cases}$$

$$\mathcal{B}[\![a_1 \le a_2]\!]\, s = \begin{cases} \mathsf{true} & \text{if } \mathcal{A}[\![a_1]\!]\, s \le \mathcal{A}[\![a_2]\!]\, s \\ \mathsf{false} & \text{otherwise.} \end{cases}$$

4.4 Commands

$$\mathcal{C}[\![\mathsf{skip}]\!]\, s = s, \qquad \mathcal{C}[\![x := a]\!]\, s = s[\mathcal{A}[\![a]\!]\, s / x],$$

$$\mathcal{C}[\![\mathsf{if}\ b\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2]\!]\, s = \begin{cases} \mathcal{C}[\![c_1]\!]\, s & \text{if } \mathcal{B}[\![b]\!]\, s = \mathsf{true} \\ \mathcal{C}[\![c_2]\!]\, s & \text{if } \mathcal{B}[\![b]\!]\, s = \mathsf{false}. \end{cases}$$

For sequencing we must propagate non-termination: if the first command diverges, so does the composite.

$$\mathcal{C}[\![c_1; c_2]\!]\, s = \begin{cases} \mathcal{C}[\![c_2]\!]\big(\mathcal{C}[\![c_1]\!]\, s\big) & \text{if } \mathcal{C}[\![c_1]\!]\, s \neq \perp \\ \perp & \text{if } \mathcal{C}[\![c_1]\!]\, s = \perp. \end{cases}$$

4.5 Kleisli lifting

The case split for sequencing is exactly the bookkeeping needed to feed a possibly-$\bot$ value into a function expecting an ordinary environment. We package this once and for all.

Definition 4 (Kleisli lifting)**.** For $f : D \to E_\bot$ define $f^\dagger : D_\bot \to E_\bot$ by

$$f^\dagger = \lambda x.\ \mathsf{if}\ x = \bot\ \mathsf{then}\ \bot\ \mathsf{else}\ f(x).$$

With this notation sequencing becomes clean:

$$\mathcal{C}[\![c_1; c_2]\!]\, s = \mathcal{C}[\![c_2]\!]^\dagger\big(\mathcal{C}[\![c_1]\!]\, s\big), \qquad \text{i.e.} \qquad \mathcal{C}[\![c_1; c_2]\!] = \mathcal{C}[\![c_2]\!]^\dagger \circ \mathcal{C}[\![c_1]\!].$$

5 The `while` Loop and the Need for Fixpoints

We want the semantics to validate the loop-unwinding equivalence

$$\mathsf{while}\ b\ \mathsf{do}\ c\ \equiv\ \mathsf{if}\ b\ \mathsf{then}\ (c; \mathsf{while}\ b\ \mathsf{do}\ c)\ \mathsf{else}\ \mathsf{skip}.$$

Writing $W = \mathcal{C}[\![\mathsf{while}\ b\ \mathsf{do}\ c]\!]$, this says W must satisfy

$$W\, s = \mathsf{if}\ \mathcal{B}[\![b]\!]\, s\ \mathsf{then}\ W^\dagger(\mathcal{C}[\![c]\!]\, s)\ \mathsf{else}\ s,$$

i.e. W is a *fixpoint* of the higher-order functional

$$F = \lambda w : \mathrm{Env} \to \mathrm{Env}_\bot.\ \lambda s : \mathrm{Env}.\ \mathsf{if}\ \mathcal{B}[\![b]\!]\, s\ \mathsf{then}\ w^\dagger(\mathcal{C}[\![c]\!]\, s)\ \mathsf{else}\ s,$$

which has type

$$F : (\mathrm{Env} \to \mathrm{Env}_\bot) \longrightarrow (\mathrm{Env} \to \mathrm{Env}_\bot).$$

We want the *least* such $W = F\, W$. But this lives in a *function space*, not a powerset, so Theorem 1 does not directly apply.

5.1 Approximating the loop

Intuitively, take the limit of the loop unwound more and more times, starting from the totally undefined function:

$$\begin{aligned} W_0 &= \lambda s.\ \bot, \\ W_1 &= F\, W_0 = \lambda s.\ \mathsf{if}\ \mathcal{B}[\![b]\!]\, s\ \mathsf{then}\ \bot\ \mathsf{else}\ s, \\ &\vdots \\ W_{n+1} &= F\, W_n. \end{aligned}$$

Here W_n correctly handles all runs that terminate within n iterations, and returns $\bot$ otherwise. Two questions remain: do the W_n form a chain in some order, and does their "limit" give the least fixpoint of F? To answer this we need an order structure on function spaces—this leads to CPOs.

6 Partial Orders and CPOs

Definition 5. A relation $\sqsubseteq$ on S is a *partial order* if it is reflexive, transitive, and antisymmetric ($x \sqsubseteq y$ and $y \sqsubseteq x$ imply $x = y$). The pair $(S, \sqsubseteq)$ is a *poset*. Two elements are *comparable* if $x \sqsubseteq y$ or $y \sqsubseteq x$; a *total order* is one in which all pairs are comparable.

Definition 6. Let $(S, \sqsubseteq)$ be a poset and $A \subseteq S$.

- b is an *upper bound* of A if $a \sqsubseteq b$ for all $a \in A$.
- b is a *least upper bound (supremum)* of A if it is an upper bound and $b \sqsubseteq c$ for every upper bound c. The supremum, if it exists, is unique.
- A *chain* is a subset whose elements are pairwise comparable.

$(S, \sqsubseteq)$ is a *chain-complete partial order (CPO)* if every chain has a supremum. The supremum of a chain C is written $\bigsqcup C$.

Remark. Taking $C = \emptyset$, every CPO has a least element $\bot = \bigsqcup \emptyset$. Any set becomes a CPO by adjoining a fresh $\bot$ below all original (mutually incomparable) elements; this is a *flat domain*. The space $\mathrm{Env}_\bot$ is exactly the flat domain on Env, and $(2^S, \subseteq)$ is a CPO.

Definition 7. Let $(X, \sqsubseteq)$ and $(Y, \sqsubseteq)$ be CPOs and $f : X \to Y$.

- f is *monotone* if $x \sqsubseteq y \Rightarrow f(x) \sqsubseteq f(y)$.
- f is *chain-continuous* if for every nonempty chain $C \subseteq X$, $f(\bigsqcup C) = \bigsqcup\{f(a) \mid a \in C\}$.

7 Knaster–Tarski for CPOs

The earlier theorem now lifts almost verbatim from powersets to arbitrary CPOs, with $\bot$ playing the role of $\emptyset$ and $\bigsqcup$ the role of $\bigcup$.

Theorem 4 (Fixpoints in a CPO)**.** *Let $(X, \sqsubseteq)$ be a CPO. Every monotone $f : X \to X$ has a least fixpoint. If f is chain-continuous, the least fixpoint is $\bigsqcup_n f^n(\bot)$.*

Proof sketch. Mirror Definition 2:

$$c_0 = \bot, \qquad c_{\alpha+1} = f(c_\alpha), \qquad c_\beta = \bigsqcup_{\alpha<\beta} c_\alpha \quad (\beta \text{ a limit}).$$

These form a transfinite chain $\bot \sqsubseteq f(\bot) \sqsubseteq f^2(\bot) \sqsubseteq \cdots$; by the same cardinality argument as before it stabilizes at some c_κ with $c_{\kappa+1} = c_\kappa$, the least fixpoint. When f is continuous the stabilization happens at ω, giving the displayed formula. □

To apply this to the loop functional F, we need the *function space* itself to be a CPO.

Theorem 5 (Function-space CPO)**.** *For CPOs C and D, the set $[C \to D]$ of continuous functions $C \to D$, ordered pointwise*

$$f \sqsubseteq g \iff \forall x \in C.\ f(x) \sqsubseteq g(x),$$

is a CPO with bottom $\bot = \lambda x.\, \bot$.

This is what lets us reason about higher-order functionals such as F using the same fixpoint machinery.

8 Semantics of the `while` Loop, Completed

Recall we seek a least fixpoint of

$$F = \lambda w : \mathrm{Env} \to \mathrm{Env}_\bot.\ \lambda s.\ \mathsf{if}\ \mathcal{B}[\![b]\!]\, s\ \mathsf{then}\ w^\dagger(\mathcal{C}[\![c]\!]\, s)\ \mathsf{else}\ s.$$

The domain is a CPO. By Theorem 5, $[\mathrm{Env} \to \mathrm{Env}_\bot]$ is a CPO. In fact every monotone function $\mathrm{Env} \to \mathrm{Env}_\bot$ is automatically continuous: chains in the (discretely ordered) source Env contain at most one element, so the continuity condition is vacuous. Moreover, the lift $w^\dagger : \mathrm{Env}_\bot \to \mathrm{Env}_\bot$ of any w is continuous.

F is monotone. Suppose $d_1 \sqsubseteq d_2$. For every s,

$$F(d_1)(s) = \mathsf{if}\ \mathcal{B}[\![b]\!]\, s\ \mathsf{then}\ d_1^\dagger(\mathcal{C}[\![c]\!]\, s)\ \mathsf{else}\ s \ \sqsubseteq\ \mathsf{if}\ \mathcal{B}[\![b]\!]\, s\ \mathsf{then}\ d_2^\dagger(\mathcal{C}[\![c]\!]\, s)\ \mathsf{else}\ s = F(d_2)(s),$$

since lifting is monotone (and the false-branch is identical).

F is continuous. Let C be a chain in $[\mathrm{Env} \to \mathrm{Env}_\bot]$. Then

$$\begin{aligned} F\Big(\bigsqcup C\Big) &= \lambda s.\ \mathsf{if}\ \mathcal{B}[\![b]\!]\, s\ \mathsf{then}\ \Big(\bigsqcup C\Big)^\dagger(\mathcal{C}[\![c]\!]\, s)\ \mathsf{else}\ s \\ &= \lambda s.\ \mathsf{if}\ \mathcal{B}[\![b]\!]\, s\ \mathsf{then}\ \bigsqcup_{d \in C} d^\dagger(\mathcal{C}[\![c]\!]\, s)\ \mathsf{else}\ s \\ &= \bigsqcup_{d \in C} \lambda s.\ \mathsf{if}\ \mathcal{B}[\![b]\!]\, s\ \mathsf{then}\ d^\dagger(\mathcal{C}[\![c]\!]\, s)\ \mathsf{else}\ s \ =\ \bigsqcup_{d \in C} F(d). \end{aligned}$$

Conclusion. By Theorem 4, F has a least fixpoint, given concretely by the limit of the approximations from Section 5:

$$\boxed{\mathcal{C}[\![\textsf{while } b \textsf{ do } c]\!] \;=\; \bigsqcup_{n<\omega} F^n(\lambda s.\,\bot) \;=\; \bigsqcup_{n<\omega} W_n.}$$

This is the meaning of the loop: the least function consistent with all finite unwindings, assigning $\bot$ precisely to the states from which the loop diverges.

9 Domain Constructions

Domain theory is the study of CPOs and continuous maps. It provides constructions that are closed under "being a CPO," so meanings of complex types can be built compositionally.

For CPOs D and E:

- The **product** $D \times E$ of ordered pairs, ordered componentwise, is a CPO; the projections are continuous.
- The **coproduct** $D+E$ (disjoint union with the two $\bot$'s identified) is a CPO; the coprojections are continuous.
- The **function space** $[D \to E]$ with pointwise order is a CPO (Theorem 5).

The following maps are all continuous, which is what makes the semantics itself continuous and hence amenable to fixpoints:

$\text{apply} : [D \to E] \times D \to E$, $\quad \text{compose} : [E \to F] \times [D \to E] \to [D \to F]$,
$\text{curry} : [D \times E \to F] \to [D \to [E \to F]]$, $\quad \text{uncurry} : [D \to [E \to F]] \to [D \times E \to F]$,
$\text{fix} : [D \to D] \to D$ (returns the least fixpoint).

In particular the loop semantics above is just an application of fix to the continuous functional F.