Categories for Imperative Semantics

PLDG Seminar

Riccardo Pucella

The aim of these notes is to provide an introduction to category theory, and a motivation for its use in denotational semantics. I will do this by showing how to apply it to give an abstract semantics to a simple imperative language. These notes are loosely based on lectures at Cornell's Programming Language Discussion Group seminars in Spring and Summer 2004.

September 2004

Categories for Imperative Semantics **Lecture 1**
PLDG Seminar

One question that often arises about category theory is: when do you need it? The actual answer is that you rarely *need* it—you can do denotational semantics quite straightforwardly without ever invoking category theory. (Although you may have problems reading modern papers on the subject...) Category theory rarely saves you from hard work, if there is hard work to be done. The main thing is that category theory provides you with a nice, elegant, and abstract way to talk about semantics by abstracting away from possibly overly concrete details. Thus, categorical language is useful to highlight the commonality and relationship between denotational domains. For instance, how would you describe the intrinsic differences and similarities between state-transformer semantics, predicate-transformer semantics, partial-order semantics, and metric-space semantics [Bakker and Vink 1996]? Of course, an answer to this question is *topologically*, since denotational semantics can be fruitfully understood as the topological study of computation. But then again, category theory was originally developed to make sense of topological notions.

The language IMP. Consider the following simple imperative language [Gunter 1992; Winskel 1993].

$$
\begin{aligned}
S ::= \;& \\
& \texttt{skip} \\
& x_i := E \\
& S_1 ; S_2 \\
& \texttt{if } B \texttt{ then } S_1 \texttt{ else } S_2 \\
& \texttt{while } B \texttt{ do } S
\end{aligned}
$$

We will assume that the intuition behind the syntax is clear. Some things should be noted, however. We assume a finite set of variables $x_1, \ldots, x_n$ that can be used to write programs. Also, rather than provide an explicit syntax for expressions E and tests B, we simply assume that these are given as a collection of appropriate expressions. For the time being, we will not have much interesting things to say about them. (That may change, though.)

A state-transformer semantics. First off, consider the standard semantics given, for instance, in Winskel [1993]. The intuition is to view the meaning of a program as a function transforming a state into a new state, hence the moniker *state-transformer* semantics. There is a little difficulty, and it has to do with the fact that one can write nonterminating programs in IMP; to deal with this, we define the meaning of a program to be a *partial function* from states to states. There are a number of equivalent ways of presenting partial functions, let us pick one that is useful. A partial function f between a set X and a set Y, written $f : X \rightharpoonup Y$, a function from a subset of X to Y. The subset of X on which f is defined, called the support of f, is denoted $\lceil f \rceil$. If $x \in \lceil f \rceil$, then $f(x) \in Y$.

We will sometimes define a partial function from X to Y with support A as $\lambda x{\in}A.e[x]$ (where e is some expression with free variable x) as the partial function f which is defined as $e[x]$ on $x \in A$, and undefined on $x \notin A$. (In general, we will freely use λ-notation to define functions.)

Given two partial functions, we compose them as follows. Given $f : X \rightharpoonup Y$ and $g : Y \rightharpoonup Z$, then $g{\circ_p}f : X \rightharpoonup Z$ is the partial function with support $\lceil f \rceil \cup f^{-1}(\lceil g \rceil)$ defined as $(g{\circ_p}f)(x) = g(f(x))$.

Let V be a set where the variables take on their values. For instance, V could be $\mathbb{Z}$ if we want to model integer variables. The state of a program is a tuple of values, one for each variable. That is, the set of states is simply $V \times \cdots \times V = V^n$. We use the notation $\vec{v}$ to represent an element of V^n, that is, an n-tuple of values. As usual, $\pi_1, \ldots, \pi_n$ are the projection functions.

Assume an appropriate semantics for tests and expressions. Let $\mathbb{B}$ be a set with two elements, say $\{T, F\}$. A test B is given a semantics as a total function $[\![B]\!] : V^n \longrightarrow \mathbb{B}$, Note that because we take the semantics to be a total function, this means that tests always evaluate to a boolean value (i.e., they always terminate, and moreover they have no side-effects).

An expression E is given a semantics as a total function $[\![E]\!] : V^n \longrightarrow V$. Intuitively, an expression E computes a value from the values of all the variables. Again, because we take the function to be total, evaluation of E is assumed to always terminate, and moreover not to have any side-effects. (Later, we shall lift the restriction on termination.)

The semantics of a program $[\![S]\!]$, is a *partial* function from V^n to V^n, given by:

$$
\begin{array}{l}
[\![\texttt{skip}]\!] = \lambda z{\in}V^n.z \\
[\![x_i := E]\!] = \lambda z{\in}V^n.(\pi_1(z), \ldots, [\![E]\!](z), \ldots, \pi_n(z)) \\
[\![S_1; S_2]\!] = [\![S_2]\!] \circ_P [\![S_1]\!] \\
[\![\texttt{if } B \texttt{ then } S_1 \texttt{ else } S_2]\!] = \lambda z{\in}P. \begin{cases} [\![S_1]\!](z) & \text{if } [\![B]\!](z) = T \\ [\![S_2]\!](z) & \text{if } [\![B]\!](z) = F \end{cases} \\
\qquad \text{where } P = \{z \mid [\![B]\!](z) = T \text{ and } z \in \lceil [\![S_1]\!] \rceil\} \cup \\
\qquad\qquad\qquad \{z \mid [\![B]\!](z) = F \text{ and } z \in \lceil [\![S_2]\!] \rceil\} \\
[\![\texttt{while } B \texttt{ do } S]\!] = \lambda z{\in} \cup_{i=0}^{\infty} \lceil f_n \rceil . f_{n_z}(z) \\
\qquad \text{where } f_0 = \lambda z{\in}\varnothing.z \\
\qquad\qquad f_{n+1} = \lambda z{\in}P. \begin{cases} (f_n \circ_p [\![S]\!])(z) & \text{if } [\![B]\!](z) = T \\ z & \text{if } [\![B]\!](z) = F \end{cases} \\
\qquad\qquad\qquad \text{where } P = \{z \mid [\![B]\!](z) = F\} \cup \\
\qquad\qquad\qquad\qquad\qquad \{z \mid [\![B]\!](z) = T \text{ and } z \in \lceil f_n \circ_p [\![S]\!] \rceil\} \\
\qquad\qquad n_z \text{ is the least } n \text{ such that } z \in \lceil f_n \rceil .
\end{array}
$$

Thus, the semantics of a program is a partial function $[\![S]\!] : V^n \rightharpoonup V^n$. In other words, if $(v_1, \ldots, v_n) \in V^n$ is an initial state of the program, where variable x_i initially has value v_i, then we get:

- if $(v_1, \ldots, v_n) \notin \lceil [\![S]\!] \rceil$, the program S does not terminate on those inputs;
- otherwise, $[\![S]\!](v_1, \ldots, v_n) = (v'_1, \ldots, v'_n)$, meaning that at the end of the execution of S, variable x_i takes value v'_i.

We can check the "correctness" of this semantics against a more "intuitive" operational semantics given in terms of explicit rewriting rules [Winskel 1993]. We are not going to do that here (yet).

Complete partial orders. There is an alternative to using partial functions to give a semantics to IMP, of course, and that is to use partial orders. Roughly, in the above context, this amount to considering adding a special token to sets, a token that denotes the value "undefined", rather than considering functions that are sometimes undefined. (We will make this statement precise in a little while.) Partial orders are of course typically used to give semantics to higher-order languages. We will not need their full generality here. However, they will make the rule for `while` much more pleasant.

Let us go quickly over the definitions. We follow the treatment of Winskel [1993, chapter 8]. A *complete partial order* (cpo) is a tuple $(X, \leq)$ where X is a set (called the carrier set) and $\leq$ is a partial order on X such that every ω-chain $x_0 \leq x_1 \leq \cdots \leq x_n \leq \ldots$ of elements of X has a least upper bound $\sqcup_{n=0}^{\infty} x_n$ in X. We often refer to a cpo simply by the name of its carrier set, leaving the partial order implicit. A cpo $(X, \leq)$ is said to be a cpo with bottom if there is an element (typically written $\bot$) such that for all $x \in X$, $\bot \leq x$. (Such an element is necessarily unique.) Some authors reserve the term "cpo" for cpos with bottom (Plotkin [1983] does, for instance), and refer to the cpos we have defined either as bottomless ω-cpos, or predomains.

A function $f : (X, \leq_X) \longrightarrow (Y, \leq_Y)$ between cpos (really, a function between the carrier sets X and Y) is *continuous* if it is *monotonic* (for all $x, x' \in X$, we have $x \leq_X x'$ implies $f(x) \leq_Y f(x')$) and for all ω-chains $x_0 \leq_X x_1 \leq_X \ldots$ in X, we have $\sqcup_{n=0}^{\infty} f(x_n) = f(\sqcup_{n=0}^{\infty} x_n)$.

A cpo $(X, \leq)$ is called *discrete* if $x \leq x'$ implies $x = x'$. Thus, a discrete cpo has a completely trivial ordering relation.

Given a cpo $(X, \leq)$, we can *lift* the cpo by adjoining a bottom element. The lifted cpo $X_\bot$ is defined as $(X \uplus \{\bot\}, \leq_\bot)$ and obtained by taking the following ordering relation: $(\bot, 1) \leq_\bot v$ for all $v \in X \uplus \{\bot\}$, and $(x, 0) \leq_\bot (x', 0)$ if and only if $x \leq x'$. (Recall the tagging we use for disjoint unions.)

Given two cpos, we can create a product cpo $(X, \leq_X) \times (Y, \leq Y)$ in the obvious way, by considering $(X \times Y, \leq)$ where the ordering $\leq$ is defined by taking $(x, y) \leq (x', y')$ if and only if $x \leq_X x'$ and $y \leq_Y y'$.

The set of all continuous functions between two cpos is itself a cpo. Given cpos $(X, \leq_X)$ and $(Y, \leq_Y)$, define the cpo $[X \longrightarrow Y] = (F, \leq)$ where F is the set of all continuous functions between X and Y, and taking $f \leq g$ if and only for all $x \in X$, $f(x) \leq_Y g(x)$. Note that if the target cpo Y has a bottom element $\bot_Y$, then $[X \longrightarrow Y]$ has a bottom element, namely the constant function $\bot_Y$.

The key thing about cpos is that every continuous function on a given cpo with bottom has a fixed point. More precisely, if $(X, \leq)$ is a cpo with bottom and $f : X \longrightarrow X$ is a continuous function, then there exists an element $\mathit{fix}_f \in X$ such that

$$f(\mathit{fix}_f) = \mathit{fix}_f.$$

This fixed point can be obtained by taking the least upper bound of the ω-chain $\bot \leq f(\bot) \leq f(f(\bot)) \leq \ldots$.

We can give a semantics similar to that of partial functions using the above. Now, we choose a discrete cpo $(V, \leq_V)$ for the values of the variables. Thus, the state of the program will be an element of the product cpo V^n given by $(V, \leq_V) \times \cdots \times (V, \leq_V) = (V^n, \leq)$, the cpo of tuples of values from V, ordered pointwise.

Let $\mathbb{B}$ be the discrete cpo of any two-element set, say $\{T, F\}$. We assume for every test B a continuous function $[\![B]\!] : V^n \longrightarrow \mathbb{B}$, that is, from the cpo V^n to discrete cpo $\mathbb{B}$. The interpretation is the same as in the partial functions case. Similarly, we assume for every expression E a continuous function $[\![E]\!] : V^n \longrightarrow V$ from the cpo V^n to the discrete cpo of values.

The semantics $[\![S]\!]$ of a program S is given by a continuous function from the cpo V^n to the *lifted* cpo $V^n_\bot$, where we interpret $\bot$ as nontermination. We write $\lambda_\bot x.e$ as an abbreviation for the *strict* function defined by:

$$\lambda y. \begin{cases} \bot & \text{if } y = \bot \\ (\lambda x.e)y & \text{otherwise.} \end{cases}$$

$$\begin{aligned}
&[\![\texttt{skip}]\!] = \lambda z.(z, 0) \\
&[\![x_i := F(\vec{x})]\!] = \lambda.(\langle \pi_1, \ldots, [\![F]\!], \ldots, \pi_n \rangle(z), 0) \\
&[\![S_1; S_2]\!] = \lambda z.((\lambda_\bot y.[\![S_2]\!](y))([\![S_1]\!](z))) \\
&[\![\texttt{if } B \texttt{ then } S_1 \texttt{ else } S_2]\!] = \lambda z. \begin{cases} [\![S_1]\!](z) & \text{if } [\![B]\!](z) = T \\ [\![S_2]\!](z) & \text{if } [\![B]\!](z) = F \end{cases} \\
&[\![\texttt{while } B \texttt{ do } S]\!] = \mathit{fix}_M \\
&\quad \text{where } M = \lambda f.\lambda z. \begin{cases} (f \circ [\![S]\!])(z) & \text{if } [\![B]\!](z) = T \\ z & \text{otherwise.} \end{cases}
\end{aligned}$$

In the definition of the `while`-loop, we use the fact that the set of continuous functions is a cpo; it must be verified that the function M so-defined is in fact continuous on the cpo $[V^n \longrightarrow V^n_\bot]$.

Again, the semantics of a program S is a continuous function from V^n to $V^n_\bot$, such that if $(v_1, \ldots, v_n) \in V^n$ is an initial state of the program, where variable x_i initially has value v_i, then we get:

- if $[\![S]\!](v_1, \ldots, v_n) = \bot$, the program S does not terminate on those inputs;
- otherwise, $[\![S]\!](v_1, \ldots, v_n) = (v'_1, \ldots, v'_n)$, meaning that at the end of the execution of S, variable x_i takes value v'_i.

Categories. Our goal is to isolate the key properties of the two semantics above, in as abstract a way as possible. Why? Because it's fun, and because it tells us the basic structure we need when giving a semantics to the basic constructs of IMP. That means that when we move to more involved extensions of IMP which may require different structure to give semantics, we will know what are the basic elements to look for to give a semantics to the IMP fragment, that is, conditionals and loops.

A *category*[1] $\mathcal{C}$ is a collection of *objects* (often referred to as $\mathcal{C}$ as well), and for each pair of objects A, B, a collection $Hom(A, B)$ of *morphisms* from A to B. A morphism f in $Hom(A, B)$ will be written $A \xrightarrow{f} B$. Morphisms are subject to the following conditions:

(1) There is an operation $\circ$ on morphisms such that if $A \xrightarrow{f} B$ and $B \xrightarrow{g} C$ are morphisms, then $A \xrightarrow{g \circ f} C$ is a morphism. In other words, if f is a morphism in $Hom(A, B)$ and g is a morphism in $Hom(B, C)$, then $g \circ f$ is a morphism in $Hom(A, C)$.

(2) The operation $\circ$ is associative, so that if we have $A \xrightarrow{f} B$, $B \xrightarrow{g} C$, and $C \xrightarrow{h} D$, then $h \circ (g \circ f) = (h \circ g) \circ f$.

(3) For every object A, there is a distinguished morphism $A \xrightarrow{1_A} A$ such that for all morphisms $A \xrightarrow{g} B$, we have $g \circ 1_A = g$, and for all morphisms $B \xrightarrow{h} A$, we have $1_A \circ h = h$.

An *isomorphism* f between objects A and B is a morphism $A \xrightarrow{f} B$ such that there exists a morphism $B \xrightarrow{g} A$ with $g \circ f = 1_A$ and $f \circ g = 1_B$.

Examples. Here are some examples of categories, where the objects are all sets. The crucial thing is that the morphisms are different in each case, which means that the categories have a vastly different feel from each other.

The category $\mathbf{Set}$ has sets as objects, and morphisms $X \xrightarrow{f} Y$ are functions $f : X \longrightarrow Y$. The composition operation on morphisms is simply function composition, and the identity morphisms are just the identity functions.

The category $\mathbf{Set}^{\mathrm{op}}$ also has sets as objects, and morphisms $X \xrightarrow{f} Y$ are reverse functions $f : Y \longrightarrow X$. Composition is "flipped" function composition (that is, $g \circ f$ is the function defined by $(g \circ f)(x) = f(g(x))$), and the identity morphisms are just the identity functions.

The category $\mathbf{Sub}$ also has sets as objects, but now there is at most a single morphism between any two objects, and there is a morphism $X \longrightarrow Y$ if and only if $X \subseteq Y$. Composition is transitivity of inclusion, and identity reflects $X \subseteq X$ for all X.

Finally, the category $\mathbf{Pfn}$ also has sets as objects, and morphisms $X \xrightarrow{f} Y$ are *partial* functions $f : X \rightharpoonup Y$, as defined earlier. Composition is partial function composition; identity morphisms

[1] For references, see MacLane [1971], or Barr and Wells [1990].

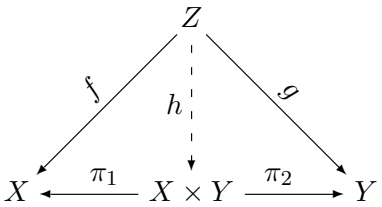
are identify functions. Note that every object of $\mathbf{Set}$ is an object of $\mathbf{Pfn}$, every morphism in $\mathbf{Set}$ is a morphism in $\mathbf{Pfn}$, composition agrees, and identities agree; in other words, $\mathbf{Set}$ is a *subcategory* of $\mathbf{Pfn}$.

The category $\mathbf{CPO}$ with cpos as objects and continuous functions as morphisms can also be defined.

We will present more examples of categories in the course of these notes, as the need arise.

Products. Many constructions that can be used to construct sets can be generalized to other categories. As an example, consider the cartesian product of sets. To generalize this to other categories, we need to isolate a property of cartesian productsa that somehow captures what's useful about them. It turns out that this is the fact that cartesian products have projections associated with them, which satisfy some rules. (We will not justify why this choice of property of cartesian product is the right one.)

A category is said to have *products* if for every object X and Y, there exists an object (denoted $X \times Y$) and morphisms π_1, π_2 such that the following diagram commutes:[2]



In other words, for all sets Z, if $Z \xrightarrow{f} X$ and $Z \xrightarrow{g} Y$, then there exists a unique $Z \xrightarrow{h} X \times Y$ (typically written $\langle f, g \rangle$) such that $\pi_1 \circ h = f$ and $\pi_2 \circ h = g$.

One can check that, for any two objects X and Y, there are two objects Z_1 and Z_2 satisfying these properties (so that either one of them could arguably be called the product of X and Y), then Z_1 and Z_2 are isomorphic, that is, there is an isomorphism from Z_1 to Z_2.

Examples. Justifying the choice of notation, one can check that in the category $\mathbf{Set}$, the product $X \times Y$ exists, and is simply given by the usual cartesian product:

$$\begin{aligned} X \times Y &= \{(x, y) \mid x \in X, y \in Y\} \\ \pi_1(x, y) &= x \\ \pi_2(x, y) &= y. \end{aligned}$$

Given any set Z, the mediating morphism $h = \langle f, g \rangle$ from Z to $X \times Y$ is also easy to describe:

$$h(z) = (f(z), g(z)).$$

[2] A diagram is said to commute if, roughly, following different paths from one object to another yields equal morphisms.

To illustrate a perhaps less intuitive construction, consider the category $\mathbf{Pfn}$. This category also has products, but they are not given by the cartesian product. Indeed, given X and Y, the following set satisfies the properties of products given above:

$$X \times Y = (X \uplus Y) \uplus \{(x, y) \mid x \in X, y \in Y\},$$

where $\uplus$ is set-theoretic disjoint union, defined as follows: $X \uplus Y = \{(x, 0) \mid x \in X\} \cup \{(y, 1) \mid y \in Y\}$. (The choice of "tags" $\{0, 1\}$ is rather arbitrary.) The projection functions are:

$$\pi_1(v) = \begin{cases} x & \text{if } v = ((x, 0), 0) \\ x & \text{if } v = ((x, y), 1) \end{cases}$$
$$\pi_2(v) = \begin{cases} y & \text{if } v = ((y, 1), 0) \\ y & \text{if } v = ((x, y), 1), \end{cases}$$

where $\lceil \pi_1 \rceil = \{((x, 0), 0) \mid x \in X\} \cup \{((x, y), 1) \mid x \in X, y \in Y\}$ and $\lceil \pi_2 \rceil = \{((y, 1), 0) \mid y \in Y\} \cup \{((x, y), 1) \mid x \in X, y \in Y\}$. Given a set Z, the mediating morphism $h = \langle f, g \rangle$ is given by the partial function h from Z to $X \times Y$ with support $\lceil f \rceil \cup \lceil g \rceil$ and defined by

$$h(z) = \begin{cases} ((f(z), 0), 0) & \text{if } z \notin \lceil g \rceil \\ ((g(z), 1), 0) & \text{if } z \notin \lceil f \rceil \\ ((f(z), g(z)), 1) & \text{otherwise.} \end{cases}$$

We also saw that the category $\mathbf{CPO}$ supports a notion of product. We can check that the construction we gave actually establishes that $\mathbf{CPO}$ has products.

Functors. We can often relate two categories $\mathcal{C}$ and $\mathcal{D}$ by mapping one to the other in a way that preserves their structure. Since categories are characterized both by their objects and their morphisms, these maps, called *functors*, need to map both objects and morphisms from $\mathcal{C}$ to $\mathcal{D}$, in a way that preserves the relationships between objects and morphisms.

A *functor* F between categories $\mathcal{C}$ and $\mathcal{D}$, written $F : \mathcal{C} \longrightarrow \mathcal{D}$, is a pair of maps on objects and on morphisms, both actually denoted by F, such that F maps an object X of $\mathcal{C}$ to an object FX of $\mathcal{D}$, and F maps a morphism $X \xrightarrow{f} Y$ of $\mathcal{C}$ to a morphism $FX \xrightarrow{Ff} FY$ of $\mathcal{D}$, subject to the conditions:

$$F(1_X) = 1_{F(X)}$$
$$F(g \circ f) = Fg \circ Ff.$$

Note that functors need not preserve further structure on the category. For instance, it need not be the case that a functor maps products to products. (Indeed, the target category may not even have products.)

Examples. For any category $\mathcal{C}$, the identity functor $1_{\mathcal{C}}$ is the identity on objects and the identity on morphisms. (A functor from a category to itself is often called an *endofunctor*.)

As a less trivial example, recall that I said that we could view complete partial orders as adding to sets a token denoting the value "undefined", which lets us do away with partial functions. We can actually make this precise, by exhibiting a functor $C : \mathbf{Pfn} \longrightarrow \mathbf{CPO}$ that does just that. This functor maps a set X in $\mathbf{Pfn}$ to the lifted discrete cpo $X_\perp$. The functor maps a morphism $X \xrightarrow{f} Y$ of $\mathbf{Pfn}$, that is, a partial function $f : X \rightharpoonup Y$, to the continuous function $f' : X_\perp \longrightarrow Y_\perp$ between the lifted discrete cpos $X_\perp$ and $Y_\perp$ defined as follows:

$$f'(x,0) = \begin{cases} (\perp, 1) & \text{if } x \notin \lceil f \rceil \\ (f(x), 1) & \text{if } x \in \lceil f \rceil \end{cases}$$
$$f'(\perp, 1) = (\perp, 1).$$

(Recall that the lifted cpo $X_\perp$ has a carrier set $X \uplus \{\perp\}$.) I leave it as an exercise to verify that f' is actually continuous, and that the functor C satisfies the required conditions.

Another functor that turns out to be important is the endofunctor from $\mathbf{CPO}$ to $\mathbf{CPO}$ that lifts cpos. More precisely, the functor $L : \mathbf{CPO} \longrightarrow \mathbf{CPO}$ maps a cpo X to the lifted cpo $C_\perp$, and maps a morphism $X \xrightarrow{f} Y$ to the morphism $X_\perp \xrightarrow{Lf} Y_\perp$ defined by:

$$Lf(\perp, 0) = (\perp, 0)$$
$$Lf(x, 1) = (f(x), 1).$$

We can check that L in fact satisfies the functor properties.

The category of categories Functors are maps between categories. For every category $\mathcal{C}$, we have an identity functor $1_{\mathcal{C}}$. One can also check that functors compose: if $F : \mathcal{C} \longrightarrow \mathcal{D}$ and $G : \mathcal{D} \longrightarrow \mathcal{E}$ are functors, then we can define the functor $GF : \mathcal{C} \longrightarrow \mathcal{E}$ in the obvious way: it maps an object X of $\mathcal{C}$ to an object GFX of $\mathcal{E}$, and a morphism $X \xrightarrow{f} Y$ of $\mathcal{C}$ to a morphism $GFX \xrightarrow{GFf} GFY$ of $\mathcal{E}$. It is an exercise to check that the functor properties are met. This forms the category $\mathbf{Cat}$ of categories, which has categories as objects[3] and functors as morphisms.

[3]There are of course some nagging issues of size at play here, the same kind that arise when forming the set of all sets. Whatever favorite cure for the problem in set theory you may have works quite fine in this setting as well: assuming universes, distinguishing sets and classes, etc. Refer to the literature for further discussion.

Last time, we saw two semantics for the simple imperative programming language IMP.

$$
\begin{aligned}
S ::= \quad & \\
& \texttt{skip} \\
& x_i := E \\
& S_1 ; S_2 \\
& \texttt{if } B \texttt{ then } S_1 \texttt{ else } S_2 \\
& \texttt{while } B \texttt{ do } S
\end{aligned}
$$

(As we noted last time, we take expressions E and Boolean tests B as abstract elements of some vocabulary of expressions. This lets us concentrate on modeling control flow. We do assume that E and B are side-effect free and always terminate. We will examine E and B more carefully in future lectures.)

Of the two semantics that we discussed last time, one was based on partial functions, where the semantics of a program S is a partial function from the set of states to the set of states, and the other was based on complete partial orders, where the semantics of a program S is a continuous functions from the set of states (viewed as a partially ordered structure) to the set of states (viewed as a partially ordered structure).

There is one obvious question lurking here: what is the commonality between those two semantics? Put another way, these two semantics are trying to capture the operational meaning of the programs in IMP in a mathematically convenient framework. Of course, what is convenient for one purpose may not be convenient for another (partial functions are sometimes easier to work with than CPO's, and sometimes metric spaces are nicer to work with), and so it makes sense to look at what it takes for a mathematical framework to be able to express the semantics of IMP. If we equate mathematical frameworks with categories (a bold move), then the question above becomes: what kind of structure does a category need to have in order to support a semantics for IMP along the lines of the partial function semantics and the CPO-based semantics? A satisfactory answer to this question will come in handy when we extend IMP with some interesting features like nondeterminism and probability, and seek to develop the corresponding semantics.

Abstract semantics. So, what do we need to give a state-transition semantics to IMP, in an abstract sense? In other words, what do we need from a semantic category $\mathcal{C}$ to act as a reasonable category in which to give a state-transition semantics to IMP? Following the intuitions underlying the partial functions semantics, as well as the CPO-based semantics, we assume that the variables will take values from some space of values that correspond to an object in the category. To be shamelessly set-theoretical, you can view such an object as consisting of the set of values that the variable can take. For the sake of presentation, we assume that there is a single object as the space of values, so that all the variables share a common space of values. (In other words, there is a single type of values. It is straightforward to extend this to multiple types of values, but it does complicate

the bookkeeping required to write down the semantics, with no real new insight.) Let X be the object representing the space of values.

In the partial functions semantics as well as the CPO-based semantics, the state of the programs are tuples of values, representing the values of each variable of the program at that state. The semantics of the programs are functions from states to states. We will use this intuition as the basis of our abstract semantics, and take the state space of programs (with n variables) to be the product $X^n = X \times \cdots \times X$, the n-fold product of X. In order for this to make sense, we need the semantic category to have enough structure. This is therefore our first assumption on $\mathcal{C}$:

A1. $\mathcal{C}$ has all finite products.

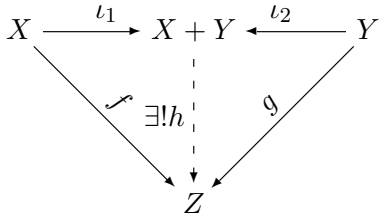
(A finite product is an obvious generalization of the binary products we saw in the last lecture.)

With this assumption, we now have enough structure to give a semantics to the first few statements of our language. Assume an appropriate semantics for expressions. More precisely, assume that every expression E has an associated morphism $X^n \xrightarrow{[\![E]\!]} X$. The following semantics is defined for program using only variables $x_1, \ldots, x_n$:

$$
\begin{aligned}
&[\![\texttt{skip}]\!] = 1_{X^n} \\
&[\![x_i := E]\!] = \langle \pi_1, \ldots, [\![E]\!], \ldots, \pi_n \rangle \\
&[\![S_1; S_2]\!] = [\![S_2]\!] \circ [\![S_1]\!]
\end{aligned}
$$

Coproducts. To account for tests and conditionals, we need to somehow be able to talk about choice. This turns out to be nicely captured by the notion of sums, also known as coproducts. Coproducts are the generalization of disjoint unions for sets. (The name "coproduct" is meant to indicate a relationship with the notion of product. Indeed, there is one, called duality, which we will not talk about.)

A category is said to have (binary) *coproducts* if for every object X and Y, there exists an object (denoted $X+Y$) and morphisms ι_1, ι_2 such that for all Z and morphisms $X \xrightarrow{f} Z$ and $Y \xrightarrow{g} Z$, the following diagram commutes:



In other words, for all objects Z, if $X \xrightarrow{f} Z$ and $Y \xrightarrow{g} Z$, then there exists a unique $X + Y \xrightarrow{h} Z$ (typically written $[f, g]$) such that $h \circ \iota_1 = f$ and $h \circ \iota_2 = g$.

Examples. The coproduct $X + Y$ in $\mathbf{Set}$ is simply given by disjoint union of sets:

$$X + Y = X \uplus Y = \{(x, 0) \mid x \in X\} \cup \{(y, 1) \mid y \in Y\}$$

The mediating morphism $[f, g]$ is easy to characterize:

$$[f, g](w) = \begin{cases} f(x) & \text{if } w = (x, 0) \\ g(y) & \text{if } w = (y, 1). \end{cases}$$

The category $\mathbf{PFn}$ also has coproducts, which turn out to be defined the same way, using set disjoint union.

Conditionals. In order to give a semantics to conditionals, we need a semantics for Boolean tests B. As we did for expressions E, assume that we are given, for every test B, a morphism capturing the meaning of B. More precisely, we take $[\![B]\!]$ to be a morphism $X^n \xrightarrow{[\![B]\!]} X^n + X^n$. Intuitively, $[\![B]\!]$ maps a state into the sum of the state spaces X^n and X^n, where the state is injected into the left component of the sum if B is true, and into the right component of the sum if B is false. Thus, we need the following assumption on the semantic category $\mathcal{C}$:

A2. $\mathcal{C}$ has binary coproducts.

To capture the fact that the state is unmodified, we need to state that B acts as the "identity" on those states. This can be captured by imposing the following condition on $[\![B]\!]$:

$$[1_{X^n}, 1_{X^n}] \circ [\![B]\!] = 1_{X^n}.$$

(Work out the intuition underlying this definition in the category $\mathbf{Set}$.) With such a semantics for Boolean tests, we can derive the semantics of conditional statements:

$$[\![\texttt{if } B \texttt{ then } S_1 \texttt{ else } S_2]\!] = [[\![S_1]\!], [\![S_2]\!]] \circ [\![B]\!]$$

This treatment of Boolean expressions B is not very natural. It would be more natural to capture the semantics of a test B my mapping a state into an object "with 2 elements", the way we did in the semantics based on partial functions and the one based on CPO's. This turns out to be more difficult than expected in the setting we have been developing. We will return to this topic in a few lectures. But first, in the next lectures, we will take care of the semantics of the while statement.

We now consider the problem of giving an abstract semantics to the looping construct. If we look at the semantics based on partial functions, or the one based on CPO's, we see that the semantics of the while loop relies on constructing a "limit" of appropriate functions from X^n to X^n (where X^n is the state space of the program). To model this with some kind of generality, we follow the approach systematized by Manes and Arbib [1986]. The idea is to ensure that there is enough structure to the morphisms between objects in the semantic category to take "limits".

Partially additive monoids. One way to capture this notion of limits is to consider partially additive monoids. Roughly speaking, a partially additive monoid is like a monoid (that is, a set with a single associate operation and an identity element), except that the operation is only partially defined, and is infinitary.

First, some terminology. Let M be a fixed set. An *I-indexed family* in M is a function $x : I \longrightarrow M$, written $\{x_i \mid i \in I\}$. We usually write x_i rather than $x(i)$. The (necessarily unique) $\varnothing$-indexed family is called the *empty family*. A family $\{y_j \mid j \in J\}$ is said to be a *subfamily* of $\{x_i \mid i \in I\}$ if $J \subseteq I$ and $y_j = x_j$ for all $j \in J$. A family $\{x_i \mid i \in I\}$ is *countable* if I is a countable set (that is, finite or countably infinite). Given an index set I, the family $\{I_j \mid j \in J\}$ is a *partition* of I if $I_j \cap I_k =$ whenever $j \neq k$, and $I = \cup_{j \in J} I_j$. Note that we allow an element I_j of the partition to be empty.

A *partially additive monoid* is a pair (M, Σ) where M is a nonempty set and Σ is a partial function mapping countable families in M to elements of M (the family $\{x_i \mid i \in I\}$ is *summable* if $\Sigma\{x_i \mid i \in I\}$ is defined) subject to the following three conditions:

(1) *Partition-associativity axiom:* If $\{x_i \mid i \in I\}$ is a countable family and $\{I_j \mid j \in J\}$ is a partition of I with J countable, then $\{x_i \mid i \in I\}$ is summable if and only if it the case both that for every $j \in J$, $\{x_i \mid i \in I_j\}$ is summable, and $\{\Sigma\{x_i \mid i \in I_j\} \mid j \in J\}$ is summable. In that case, $\Sigma\{x_i \mid i \in I\} = \Sigma\{\Sigma\{x_i \mid i \in I_j\} \mid j \in J\}$.

(2) *Unary sum axiom:* Any family $\{x_i \mid i \in I\}$ where $|I| = 1$ is summable, and $\Sigma\{x_i \mid i \in I\} = x_j$ if $I = \{j\}$.

(3) *Limit axiom:* If $\{x_i \mid i \in I\}$ is a couinatble family, and if the subfamily $\{x_i \mid i \in F\}$ is summable for every finite set $F \subseteq I$, then $\{x_i \mid i \in I\}$ is summable.

Alternative notations for $\Sigma\{x_i \mid i \in I\}$ include $\Sigma_{i \in I} x_i$, and $x_{i_1} + x_{i_2} + \cdots + x_{i_k} + \ldots$, if $I = \{i_1, \ldots, i_k, \ldots\}$. This latter notation gives preference to a particular ordering of the elements of the index set I, but by partition-associativity, this order is immaterial. We will often use $+$ when talking about summation of finite families.

The following result is immediate, and provides a converse to the Limit axiom.

Theorem 1 *Let (M, Σ) be a partially additive monoid. Then any subfamily of a summable family is summable.*

Proof. Immediate by Partition-associativity. Let $\{x_i \mid i \in I\}$ be summable, and let $K \subseteq I$. Define $I_1 = K$, $I_2 = I - K$, so that $\{I_j \mid j \in \{1, 2\}\}$ partitions I. By the Partition-associatitivity axiom, $\{x_i \mid i \in K\}$ is summable. □

Examples. Let X, Y be two sets and let $Pfn(X, Y)$ be the set of partial functions from X to Y. For a partial function $f \in Pfn(X, Y)$, let $\lceil f \rceil$ be the domain of definition of f, that is, the set of all $x \in X$ such that $f(x)$ is defined. There are two ways of imposing a partially additive structure on Pfn. Define the partial operation Σ^{di} such that $\{f_i \mid i \in I\}$ is summable if and only if $\lceil f_i \rceil \cap \lceil f_j \rceil = \varnothing$ for all $i \neq j$, in which case take

$$(\Sigma^{di}\{f_i \mid i \in I\})(x) = \begin{cases} f_j(x) & \text{if } x \in \lceil f_j \rceil \\ \text{undefined} & \text{otherwise.} \end{cases}$$

Alternatively, define the partial operation Σ^{ov} such that $\{f_i \mid i \in I\}$ is summable if and only if $f_i(x) = f_j(x)$ for all $x \in \lceil f_i \rceil \cap \lceil f_j \rceil$, in which case we take

$$(\Sigma^{di}\{f_i \mid i \in I\})(x) = \begin{cases} f_j(x) & \text{if } x \in \lceil f_j \rceil \\ \text{undefined} & \text{otherwise.} \end{cases}$$

It is straightforward to check that both $(Pfn(X, Y), \Sigma^{di})$ and $(Pfn(X, Y), \Sigma^{ov})$ are partially additive monoids.

Zero elements. The partially additive monoids $(Pfn(X, Y), \Sigma^{di})$ and $(Pfn(X, Y), \Sigma^{ov})$ have a special element, the nowhere defined function, that can always be added to or removed from a summable family to yield a summable family. This is a general fact about partially additive monoids.

Theorem 2 *Let (M, Σ) be a partially additive monoid, and let $! : \varnothing \longrightarrow M$ be the empty family in M. Then $0 = \Sigma!$ exists.*

Proof. Since M is nonempty, there exists $x \in M$, so by the Unary sum axiom, $\{x\}$ is summable. Since the empty familyis a subfamily of every family, including $\{x\}$, the empty family is summable by Theorem 1. □

Theorem 3 *Let (M, Σ) be a partially additive monoid. If $\{x_i \mid i \in I\}$ is a summable family in M, J a countable set disjoint from I, and $x_i = 0$ for $i \in J$, then $\{x_i \mid i \in I \cup J\}$ is summable, and $\Sigma\{x_i \mid i \in I \cup J\} = \Sigma\{x_i \mid i \in I\}$.*

Proof. For $j \in I \cup J$, define

$$I_j = \begin{cases} \{j\} & \text{if } j \in I \\ \varnothing & \text{if } j \notin I, \end{cases}$$

and thus $\{I_j \mid j \in I \cup J\}$ partitions I. By the Partition-associativity axiom, $\{\Sigma\{x_i \mid i \in I_j\} \mid j \in J\}$ is summable, and $\{x_i \mid i \in I = \Sigma\{\Sigma\{x_i \mid i \in I_j\} \mid j \in J\}$. Since $\Sigma\{x_i \mid i \in I_j\} = x_j$ if $j \in I$ and$\Sigma\{x_i \mid i \in I_j\} = 0$ if $j \notin I$, we get the desired result. □

There are no additive inverses in partially additive monoids. In fact, if $x + y = 0$, then $x = y = 0$. More generally, we have the following result.

Theorem 4 *Let (M, Σ) be a partially additive monoid, and let $\Sigma\{x_i \mid i \in I\} = 0$. Then $x_i = 0$ for all $i \in I$.*

Proof. By the Partition-associativity axiom, for each $i \in I$, we have $x_i + \Sigma\{x_j \mid j \in I - \{i\}\} = \Sigma\{x_i \mid i \in I\} = 0$; therefore, it is sufficient to prove that $x = 0$ whenever $x + y = 0$. This follows from Theorem 3 and Partition-associativity:

$$\begin{aligned} x &= x + 0 + 0 + \dots \\ &= x + (y + x) + (y + x) + \dots \\ &= (x + y) + (x + y) + (x + y) + \dots \\ &= 0 + 0 + 0 + \dots \\ &= 0 \end{aligned}$$

□

The reason for introducing partially additive monoids in the previous lecture is, essentially, that to model the while loop, we require the hom-sets of the semantic category to at least have the structure of a partially additive monoid (for a suitably defined sum operation). Recall that the hom-set $Hom(X, Y)$ is the set of all morphisms from object X to object Y in a category. (Assume that categories under considerations are such that $Hom(X, Y)$ is indeed a set; such categories are sometimes called *locally small*.)

Partially additive structure. Let $\mathcal{C}$ be a category. A *partially additive structure* on $\mathcal{C}$ assigns to every hom-set $Hom(X, Y)$ a partially defined operation $\Sigma_{X,Y}$ such that $(Hom(X, Y), \Sigma_{X,Y})$ is a partially additive monoid and composition distributes over $\Sigma_{X,Y}$: if $\{f_i \mid i \in I\}$ is a summable family in $Hom(X, Y)$, then:

- for all W and $W \xrightarrow{g} X$, $\{f_i \circ g \mid i \in I\}$ is summable, and $(\Sigma_{X,Y}\{f_i \mid i \in I\}) \circ g = \Sigma_{W,Y}\{f_i \circ g \mid i \in I\}$;
- for all Z and $Y \xrightarrow{h} Z$, $\{h \circ f_i \mid i \in I\}$ is summable, and $h \circ (\Sigma_{X,Y}\{f_i \mid i \in I\}) = \Sigma_{X,Z}\{h \circ f_i \mid i \in I\}$.

We typically write Σ for $\Sigma_{X,Y}$ when the context makes it clear the hom-set under consideration.

Recall that partially additive monoids have zeroes. For a category with a partially additive structure, zeroes translate into the following notion. A category $\mathcal{C}$ has *zero morphisms* if there is a distinguihsed morphism $X \xrightarrow{0_{X,Y}} Y$ in every hom-set $Hom(X, Y)$ such that for all objects W, X, Y, Z and morphisms $W \xrightarrow{f} X$, $Y \xrightarrow{g} Z$, the following diagram commutes:

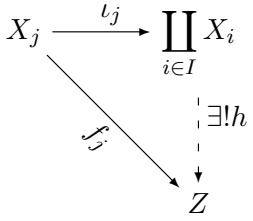
$$\begin{array}{ccc} W & \xrightarrow{f} & X \\ {\scriptstyle 0_{W,Z}}\downarrow & & \downarrow{\scriptstyle 0_{X,Y}} \\ Z & \xrightarrow{g} & Y. \end{array}$$

Theorem 5 *If a category has a partially additive structure, it has zero morphisms.*

Proof. Take $0_{X,Y} = \Sigma_{X,Y}\{\}$, where $\{\}$ is the empty family in $Hom(X, Y)$. □

Examples. The category $\mathbf{Pfn}$ can be given two partially additive structures, one corresponding to the partial operation Σ^{di}, the other to Σ^{ov}. It is straightforward to check that composition distributes over Σ^{di} and over Σ^{ov}.

Countable coproducts. The theory we develop in the coming lectures requires a generalization of coproducts to coutably infinite families of objects. A category has *countable coproducts* if for every countable famility $\{X_i \mid i \in I\}$, there exists an object (denoted $\coprod_{i\in I} X_i$ and morphisms ι_i (for $i \in I$) such that for all objects Z and all families of morphisms $\{X_i \xrightarrow{f_i} Z \mid i \in I\}$, there exists a unique f such that for all $j \in I$, the following diagram commutes:



The mediating morphism h is typically written $[f_i \mid i \in I]$, generalizing the notation for the mediating morphism of binary coproducts.

When the family $\{X_i \mid i \in I\}$ is such that $X_i = X$ for all $i \in I$, we write $I \cdot X$ for $\coprod_{i\in I} X_i$.

The following property of coproducts provides for an easy test of equality of morphisms with a coproduct as domain.

Theorem 6 *Let $\mathcal{C}$ be a category with countable coproducts, let $\{X_i \mid i \in I\}$ be a countable family of objects of $\mathcal{C}$, and let Y be an object of $\mathcal{C}$. If $\coprod_{i\in I} X_i \xrightarrow{f} Y$ and $\coprod \xrightarrow{g} Y$ are morphisms such that for all $j \in I$, $f \circ \iota_j = g \circ \iota_j$, then $f = g$.*

Proof. This follows immediately from the uniqueness of the mediating morphism for countable coproducts. Consider the morphisms $X_j \xrightarrow{f\circ\iota_j} Y$ and $X_j \xrightarrow{g\circ\iota_j} Y$. There is a unqiue morphism $\coprod_{i\in I} X_i \xrightarrow{h} Y$ such that $h \circ \iota_j = f \circ \iota_j$ for all $j \in I$. Since both f and g clearly satisfy the conditions for being an h, by uniqueness of h, we have that $h = f = g$. □

In a category with coproducts and zero morphisms, a special class of morphisms can be defined.

Theorem 7 *Let $\mathcal{C}$ be a category with countable coproducts and zero morphisms. For any countable family $\{X_i \mid i \in I\}$ of objects of $\mathcal{C}$, there exists morphisms $\rho_j : \coprod_{i\in I} X_i \longrightarrow X_j$ for every $j \in I$ satisfying*

$$\rho_j \circ \iota_i = \begin{cases} 1_{X_j} & \text{if } i = j \\ 0_{X_i,X_j} & \text{if } i \neq j. \end{cases}$$

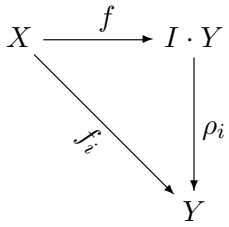
The morphisms ρ_j are called quasi-projections.

Proof. Fix a $j \in I$, and take $\rho_j = [f_i \mid i \in I]$, where $\{\coprod_{i\in I} X_i \xrightarrow{f_i} X_j \mid i \in I\}$ is the family of morphisms where f_i to be 1_{X_j} if $i = j$, and $0_{X_i,X_j}$ if $i \neq j$. □

Intuitively, a quasi-projection ρ_j takes an element of the countable coproduct, and returns the value if it is in the jth component of the coproduct, otherwise, it is undefined (corresponds to the zero morphism). Note that quasi-projections require zero morphisms. Thus, we cannot have quasi-projections in $\mathbf{Set}$, since $\mathbf{Set}$ does not have zero morphisms. In contrast, quasi-projections exist in $\mathbf{Pfn}$, which has both countable coproducts and zero morphisms.

Partially additive categories. A *partially additive category* $\mathcal{C}$ is a category with countable coproducts and partially additive structure satisfying the following two axioms:

(1) *Compatible sum axiom:* If $\{f_i \mid i \in I\}$ is a countable familty of morphisms $X \xrightarrow{f_i} Y$ and there exists a morphism $X \xrightarrow{f} I \cdot Y$ such that the following diagram commutes for all i



then $\{f_i \mid i \in I\}$ is summable.

(2) *Untying axiom:* If the morphisms $X \xrightarrow{f} Y$ and $X \xrightarrow{g} Y$ are summable, then the morphisms $X \xrightarrow{\iota_1 \circ f} Y + Y$ and $X \xrightarrow{\iota_2 \circ g} Y + Y$ are summable.

Intuitively, the axioms of a partially additive category relate the partially additive structure of the category and the coproduct structure of the category. The compatible sum axiom says if a family of morphisms can be "bundled" into the coproduct $I \cdot Y$, then they are summable. The untying axiom says that if two morphism are summable, then they are summable when they inject into the left and right of a coproduct. The fact that the untying axiom only talks about two morphisms rather than a countable family is not a restriction, since the following result is easy to establish.

Theorem 8 *In a partially additive category, if $\{f_i \mid i \in I\}$ is a summable family of morphisms $X \xrightarrow{f_i} Y$, then $\{\iota_i \circ f_i \mid i \in I\}$ is a summable family of morphisms $X \xrightarrow{\iota_i \circ f_i} I \cdot Y$.*

Proof. By the unary sum axiom of the partially additive structure of the category, any singleton family is summable. It is easy to prove by induction on the size of summable families that if $\{f_i \mid i \in F\}$ is a subfamily of $\{f_i \mid i \in I\}$ for any finite I, and hence is summable, then $\{\iota_i \circ f_i \mid i \in F\}$ is summable, using the untying axiom. Thus, any finite subfamily of $\{\iota_i \circ f_i \mid i \in I\}$ is summable, so by the limit axiom of the partially additive structure of the family, we have $\{\iota_i \circ f_i \mid i \in I\}$ is summable. □

Why are partially additive categories so interesting?

(1) Morphisms into coproducts can be uniquely decomposed

(2) The partially additive structure is unique

(3) They support a natural notion of iteration

(4) They form a natural family of traced monoidal categories [Joyal, Street, and Verity 1996], which are useful in concurrency and topology.

Before venturing into establishing the first three points (the fourth is for edification only), we need to highlight a few morphisms that always exist in partially additive categories (beyond zero morphisms and quasi-projections).

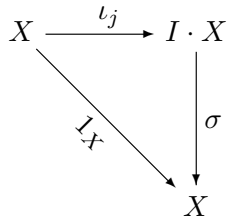
Theorem 9 *Let $\mathcal{C}$ be a category with countable coproducts and zero morphisms. Let $\{X_i \mid i \in I\}$ be a countable family of objects of $\mathcal{C}$.*

(1) *There exists a morphism $\coprod_{i\in I} X_i \xrightarrow{\Delta} I \cdot \coprod_{i\in I} X_i$ (the* diagonal injection*) such that the following diagram commutes for all j:*

$$\begin{array}{ccc} X_j & \xrightarrow{\iota_j} & \coprod_{i\in I} X_i \\ \downarrow{\scriptstyle \iota_j} & & \downarrow{\scriptstyle \iota_j} \\ \coprod_{i\in I} X_i & \xrightarrow{\Delta} & I \cdot \coprod_{i\in I} \end{array}$$

In other words, an element of X_j is sent to the jth copy of $\coprod_{i\in I} X_i$.

(2) *If $X_i = X$ for all i, there exists a morphism $I \cdot X \xrightarrow{\sigma} X$ such that the following diagram commutes for all j:*



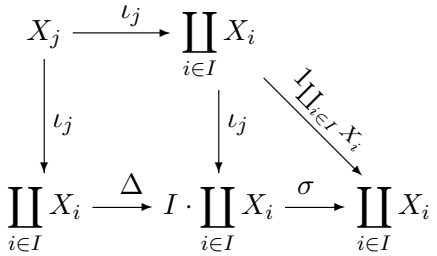
Proof. Δ exists by the defining property of coproducts, taking $\Delta = [\iota_i \circ \iota_i \mid i \in I]$.

Similarly, σ exists by the defining property of coproducts, taking $\sigma = [1_X \mid i \in I]$. □

The morphisms Δ and σ are strongly related.

Theorem 10 *If $\mathcal{C}$ is a category with countable coproducts and zero morphisms, and $\{X_i \mid i \in I\}$ is a countable family of objects of $\mathcal{C}$, then $\sigma \circ \Delta = 1_{\coprod_{i\in I} X_i}$.*

Proof. By Theorem 6, it suffices to show that for all j, $\sigma \circ \Delta \circ \iota_j = 1_{\coprod_{i\in I} X_i} \circ \iota_j$. For any j, the following diagram is easily seen to commute (and since every square[4] in the diagram commutes):



Alternatively, we can derive that

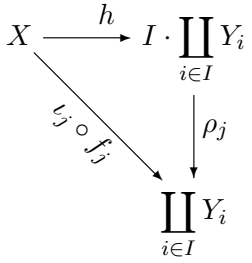
$$\begin{aligned}\sigma \circ \Delta \circ \iota_j &= \sigma \circ (\iota_j \circ \iota_j) \\ &= 1_{\coprod_{i\in I} X_i} \circ \iota_j.\end{aligned}$$

For such simple equalities, the diagram proof is much more convenient. □

Theorem 11 (Decomposition) *Let $\mathcal{C}$ be a partially additive category. If $X \xrightarrow{f} \coprod_{i\in I} Y_i$ be a morphism in $\mathcal{C}$, then there is a unique family $\{f_i \mid i \in I\}$ of morphisms $X \xrightarrow{f_i} Y_i$ such that $f = \sum_{i\in I} \iota_i \circ f_i$, namely, $f_i = \rho_i \circ f$.*

Proof. Let $f_i = \rho_i \circ f$.

(1) The first step of the proof is to check that $\{\iota_i \circ f_i \mid i \in I\}$ is summable. By the compatible sum axiom, it suffices to show that there exists an h such that the following diagram commutes:



The morphism $\Delta \circ f$ is such an h.

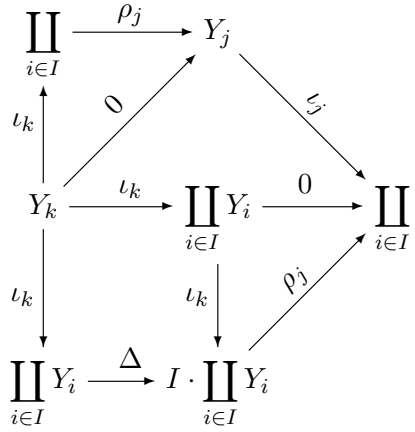
We first prove that

$$\iota_j \circ \rho_j = \rho_j \circ \Delta. \tag{1}$$

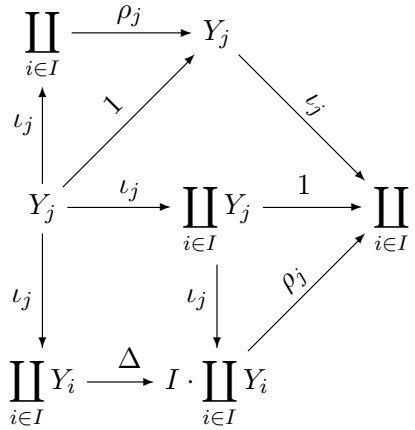
By Theorem 6, it suffices to show that for all k, $\iota_j \circ \rho_j \circ \iota_k = \rho_j \circ \Delta \circ \iota_k$. We show this for the cases where $k \neq j$ and $k = j$, by exhibiting appropriate commutative diagrams. If $k \neq j$, the following

[4] Including triangles...

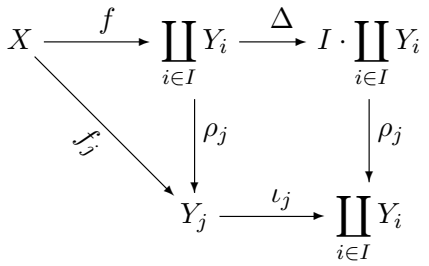
diagram commutes because all squares commute:



Similarly, if $k = j$, the following diagram commutes because all squares commute:



With (1), it is easy to see that $\Delta \circ f$ satisfies the prerequisites of the compatible sum axiom:



(The square on the left is the definition of f_j, while the square on the right is just equation (1).)

(2) Similarly, we can show that the family $\{\rho_i \mid i \in I\}$ is summable. Again, by appealing to the compatible sum axiom, taking $1_{I\cdot\coprod_{i\in I} Y_i}$ as the required morphism. Therefore, $\sum_{i\in I} \rho_i$ exists, and thus by the untying axiom, $\sum_{i\in I} \iota_i \circ \rho_i$ exists. We show that $\sum_{i\in I} \iota_i \circ \rho_i = 1_{I\cdot\coprod_{i\in I} Y_i}$. By Theorem 6, it suffices to show $(\sum_{i\in I} \iota_i \circ \rho_i) \circ \iota_j = 1_{I\cdot\coprod_{i\in I} Y_i} \circ \iota_j$. This is a straightforward

application of distributivity and properties of zero morphisms:

$$\begin{aligned}(\sum_{i\in I} \iota_i \circ \rho_i) \circ \iota_j &= \sum_{i\in I} (\iota_i \circ \rho_i \circ \iota_j) \\ &= \sum_{i\neq j} (\iota_i \circ 0) + 1 \circ \iota_j \\ &= \sum_{i\neq j} (0 \circ \iota_i) + 1 \circ \iota_j \\ &= ((\sum_{i\neq j} 0) + 1) \circ \iota_j \\ &= 1 \circ \iota_j.\end{aligned}$$

(3) Finally, we can show that $f = \sum_{i\in I} \iota_i \circ f_i$ using Theorem 10:

$$\begin{aligned}f &= \sigma \circ \Delta \circ f \\ &= \sigma \circ 1 \circ \Delta \circ f \\ &= \sigma \circ (\sum_{i\in I} \iota_i \circ \rho_i) \circ \Delta \circ f \\ &= \sum_{i\in I} \sigma \circ \iota_i \circ \rho_i \circ \Delta \circ f \\ &= \sum_{i\in I} 1 \circ \iota_i \circ \rho_i \circ f \\ &= \sum_{i\in I} \iota_i \circ f_i.\end{aligned}$$

Uniqueness is easy to check; if $f = \sum_{i\in I} \iota_i \circ g_i$, then

$$\begin{aligned}f_j &= \rho_j \circ f \\ &= \rho_j \circ (\sum_{i\in I} \iota_i \circ g_i) \\ &= \sum_{i\in I} (\rho_j \circ \iota_i \circ g_i) \\ &= g_j.\end{aligned}$$

□

We now prove the remaining two properties of partially additive categories, namely that the partially additive strutcure is unique, and that there is a natural notion of iteration that can be defined.

Theorem 12 (Uniqueness) *The Σ operation of a partially additive category is* unique*, as follows: if $\mathcal{C}$ is a partially additive category, then $\{f_i \mid i \in I\}$ in $Hom(X, Y)$ is summable if and only if it is compatible; in that case, the morphism $X \xrightarrow{f} I \cdot Y$ with $\rho_i \circ f = f_i$ is unique, and $\sum_{i \in I} f_i = \sigma \circ f$.*

Proof. If $\{f_i \mid i \in I\}$ is compatible, then it is summable by a direct application of the compatible sum axiom. Conversely, suppose $\sum_{i \in I} f_i$ exists. By the untying axiom, $f = \sum_{i \in I} \iota_i \circ f_i$ exists, and we can check that

$$\begin{aligned} \rho_j \circ f &= \rho_j \circ (\sum_{i \in I} \iota_i \circ f_i) \\ &= \sum_{i \in I} \rho_j \circ \iota_i \circ f_i \\ &= f_j. \end{aligned}$$

Therefore, by the compatible sum axiom, $\{f_i \mid i \in I\}$ is summable.

To see that the morphism f is unique, note that if we have morphisms $X \xrightarrow{f} I \cdot Y$ and $X \xrightarrow{g} I \cdot Y$ both satisfying $\rho_i \circ f = f_i = \rho_i \circ g$, then by the Decomposition theorem, $\{f_i \mid i \in I\}$ is the unique family such that $f = \sum_{i \in I} \iota_i \circ f_i$ and $g = \sum_{i \in I} \iota_i \circ f_i$, so that $f = g$.

Finally, note that

$$\begin{aligned} \sigma \circ f &= \sigma \circ \sum_{i \in I} \iota_i \circ f_i \\ &= \sum_{i \in I} \sigma \circ \iota_i \circ f_i \\ &= \sum_{i \in I} 1 \circ f_i \\ &= \sum_{i \in I} f_i. \end{aligned}$$

□

Theorem 13 (Iteration) *Given a morphism $X \xrightarrow{f} X + Y$ in a partially additive category, then there exists morphisms $X \xrightarrow{f_1} X$ and $X \xrightarrow{f_2} Y$ such that $f = \iota_1 \circ f_1 + \iota_2 \circ f_2$ and the sum*

$$f^\dagger = \sum \{f_2 \circ f_1^i \mid i \in \mathbb{N}\}$$

exists, where $f^0 = 1$ and $f^{i+1} = f \circ f^i$.

Proof. The required f_1 and f_2 exist by the Decomposition theorem.

We first establish the following preliminary result: for any morphism $X \xrightarrow{g} Y$, the sum $(g \circ f_1) + f_2$ exists. This result follows by observing that

$$\begin{aligned}[g, 1_Y] \circ f &= [g, 1_Y] \circ (\iota_1 \circ f_1 + \iota_2 \circ f_2) \\ &= [g, 1_Y] \circ \iota_1 \circ f_1 + [g, 1_Y] \circ \iota_2 \circ f_2 \\ &= g \circ f_1 + 1_Y \circ f_2 \\ &= (g \circ f_1) + f_2.\end{aligned}$$

To show that $f^\dagger$ exists, it is sufficient, by the limit axiom for partially additive monoids, to show that every for every finite subset $F \subseteq \mathbb{N}$, $\{f_2 \circ f_1^i \mid i \in F\}$ is summable. Since for every finite subset $F \subseteq \mathbb{N}$ there is an n such that $F \subseteq \{0, 1, \ldots, n\}$, and since every subfamily of a summable family is summable, it is sufficient to show that every family of the form $\{f_2 \circ f_1^i \mid i \in \{0, 1, 2, \ldots, n\}\}$ is summable. We can show this easily by induction. By the unary sum axiom, $f_2 = f_2 \circ f_1^0$ is summable. The inductive step is straightforward. Assume that $f_2 \circ f_1^n + f_2 \circ f_1^{n-1} + \cdots + f_2 \circ f_1 + f_2$ is summable. Taking this expression as g in the preliminary result above, we get that $(f_2 \circ f_1^n + \cdots + f_2 \circ f_1 + f_2) \circ f_1 + f_2$ exists. By distributivity of $\circ$ over $+$, we get that $f_2 \circ f_1^{n+1} + \cdots + f_2 \circ f_1^2 + f_2 \circ f_1 + f_2$ exists, as required. □

The $f^\dagger$ operation can be understood as computing a fixed point of a particular iterative process expressed by the morphism $X \xrightarrow{f} X + Y$. Intuitively, the process arises as follows. From X, the morphisms f either gives yields something in X, or something in Y. If it yields something in X, we can apply f again, otherwise, we're done with a value in Y. Thus, the result of the process is a morphism from X to Y, which may be the zero morphism if the process never terminates. This view of iteration is due to Elgot [1975], and has been studied extensively by Bloom and Esik [1993]. We will soon see how this pertains to modeling the while loop in IMP. First, let us formally establish the relevant properties of $f^\dagger$.

Theorem 14 *Let $\mathcal{C}$ be a partially additive category, let f be a morphism $X \xrightarrow{f} X + Y$, and let $f_1, f_2, f^\dagger$ be as in the Iteration theorem.*

(1) *The equation $(f^\dagger \circ f_1) + f_2$ holds.*

(2) *The morphism $f^\dagger$ is a solution to the Elgot equation $[\xi, 1_Y] \circ f = \xi$, where $X \xrightarrow{\xi} Y$.*

Proof. Two straightforward computations:

$$\begin{aligned}f^\dagger \circ f_1 + f_2 &= (\sum_{i \in \mathbb{N}} f_2 \circ f_1^i) \circ f_1 + f_2 \\ &= (\sum_{i \in \mathbb{N}} f_2 \circ f_1^{i+1}) + f_2\end{aligned}$$

$$= \sum_{i \in \mathbb{N}} f_2 \circ f_1^i$$
$$= f^\dagger$$

$$\begin{aligned}[f^\dagger, 1_Y] \circ f &= [f^\dagger, 1_Y] \circ (\iota_1 \circ f_1 + \iota_2 \circ f_2) \\ &= [f^\dagger, 1_Y] \circ \iota_1 \circ f_1 + [f^\dagger, 1_Y] \circ \iota_2 \circ f_2 \\ &= f^\dagger \circ f_1 + 1_Y \circ f_2 \\ &= f^\dagger \circ f_1 + f_2.\end{aligned}$$

□

Semantics of the while loop. We now have all the ingredients to complete out semantics for IMP (finally!). To interpret the while loop, we make the following assumption on the semantic category $\mathcal{C}$:

A3. $\mathcal{C}$ is a partially additive category.

Since a partially additive category has countable coproducts, property A3 implies property A2.

Recall that the state space of a program is represented by the object X^n in the category $\mathcal{C}$, and that a Boolean express B is interpreted by a morphism $X^n \xrightarrow{[\![B]\!]} X^n + X^n$. A single iteration of the while loop `while` B `do` S can be interpreted by the morphism $X^n \xrightarrow{[\![B]\!]} X^n + X^n \xrightarrow{[\iota_1 \circ [\![S]\!], \iota_2]} X^n + X^n$, where the result is injected on the left of the coproduct if the iteration is performed, and in the right of the coproduct if the iteration was not performed (because the condition was false). Following the intuition underlying the Iteration theorem, we should be able to use the $\cdot^\dagger$ operation to find a fixed-point for applying these iterations. Thus, we take the semantics of the while loop as:

$$[\![\texttt{while } B \texttt{ do } S]\!] = ([\iota_1 \circ [\![S]\!], \iota_2] \circ [\![B]\!])^\dagger.$$

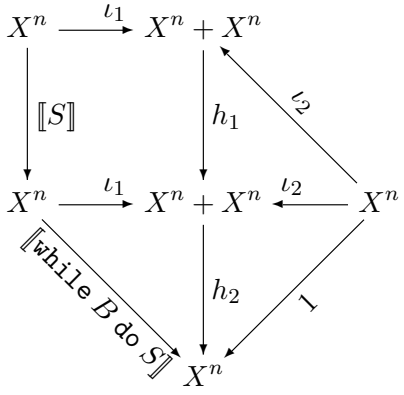
To convince ourselves that the above sematnics works, we should check that it satisfies the fixed point property of the while loop: if B is true, then $[\![\texttt{while } B \texttt{ do } S]\!]$ should be the same as $[\![S; \texttt{while } B \texttt{ do } S]\!]$. If B is false, then $[\![\texttt{while } B \texttt{ do } S]\!]$ should be the same as $[\![\texttt{skip}]\!]$. Equationally, this means that we want:

$$[[\![\texttt{while } B \texttt{ do } S]\!] \circ [\![S]\!], 1] \circ [\![B]\!] = [\![\texttt{while } B \texttt{ do } S]\!]. \tag{2}$$

Now, by Theorem 14, taking $f = [\iota_1 \circ [\![S]\!], \iota_2] \circ [\![B]\!]$ and $f^\dagger = [\![\texttt{while } B \texttt{ do } S]\!]$ in the Elgot equation gives us:

$$[[\![\texttt{while } B \texttt{ do } S]\!], 1] \circ [\iota_1 \circ [\![S]\!], \iota_2] \circ [\![B]\!] = [\![\texttt{while } B \texttt{ do } S]\!]. \tag{3}$$

Therefore, to establish (2), it suffices to show that $[[\![\texttt{while}\ B\ \texttt{do}\ S]\!], 1] \circ [\iota_1 \circ [\![S]\!], \iota_2] = [[\![\texttt{while}\ B\ \texttt{do}\ S]\!] \circ [\![S]\!], 1]$. This is established by constructing the following commutative diagram, where all squares can easily be seen to commute.



where $h_1 = [\iota_1 \circ [\![S]\!], \iota_2]$ and $h_2 = [[\![\texttt{while}\ B\ \texttt{do}\ S]\!], 1]$. Note that by the property of coproducts, there is a *unique* morphism $X^n + X^n \xrightarrow{h} X^n$ (namely, $[[\![\texttt{while}\ B\ \texttt{do}\ S]\!] \circ [\![S]\!], 1]$) such that $h \circ \iota_1 = [\![\texttt{while}\ B\ \texttt{do}\ S]\!] \circ [\![S]\!]$ and $h \circ \iota_2 = 1$. Since, according to the above diagram, the morphism $h_2 \circ h_1$ has those properties, we get that $h_2 \circ h_1 = h$, that is, $[[\![\texttt{while}\ B\ \texttt{do}\ S]\!] \circ [\![S]\!], 1] = [[\![\texttt{while}\ B\ \texttt{do}\ S]\!], 1] \circ [\iota_1 \circ [\![S]\!], \iota_2]$, as required.

Up until now, we have focused on describing a categorical semantics for IMP. We have done fairly well: any category satisfying axioms A1–3 can be used to give a state-transition semantics to IMP.

There are, however, a number of little difficulties that need to be resolved. These lead to important insight. Recall the partial function semantics of IMP. We assumed that basic expressions E are not partial, but always return a value; that is, $[\![E]\!]$ is assumed to be a total function from X^n to X. This is one aspect that we have not captured in our categorical semantics; there is no easy way to say that the morphisms associated with $[\![E]\!]$ are in some sense "total".

There are a number of ways of addressing this problem. Here is one. Intuitively, we can think of a partial function as total functions over sets with a distinguished element $\perp$ representing the value "undefined". So, one approach would be as follows: starting with a base category $\mathcal{C}$ to give semantics to expressions (where every morphism is assumed total), extend it to a category $\mathcal{D}$ where objects are extended with an "undefined" element. If we do this right, we can get $\mathcal{C}$ embedded in $\mathcal{D}$, so that expressions can be given a semantics in $\mathcal{C}$, while statements (which can be partially defined due to the presence of the while loop) can be given semantics in $\mathcal{D}$.

Before being able to do this, we need a number of concepts. As the plan above shows, we need to be able to relate categories. This is what the notion of functor is all about. Recall that a functor $F : \mathcal{C} \longrightarrow \mathcal{D}$ is a map from objects of $\mathcal{C}$ to objects of $\mathcal{D}$ and from morphisms of $\mathcal{C}$ to morphisms of $\mathcal{D}$ such that if $X \xrightarrow{f} Y$ is a morphism in $\mathcal{C}$, then $FX \xrightarrow{Ff} FY$ is a morphism in $\mathcal{D}$, subject to $F(1_X) = 1_{FX}$ and $F(g \circ f) = Fg \circ Ff$.

Examples. The following functors are well known. The functor $I : \mathbf{Set} \longrightarrow \mathbf{Set}$ is defined as follows:

$$IX = X \uplus \{\perp\}$$
$$(If)(x) = \begin{cases} f(x) & \text{if } x \neq \perp \\ \perp & \text{if } x = \perp. \end{cases}$$

(The notation $x \neq \perp$ and $x = \perp$ is a simplified way to express the check that x is different or equal to the new value $\perp$ adjoined to the set X; a more careful statement would depend on the definition of $\uplus$.) Thus, IX adjoins a new element $\perp$ to the set X, and the map If carries $\perp$ to $\perp$, otherwise acts as f.

The functor $P : \mathbf{Set} \longrightarrow \mathbf{Set}$ is defined as follows:

$$PX = \{W \mid W \subseteq X\}$$
$$(Pf)(W) = \{f(x) \mid x \in W\}.$$

The functor P is called the *powerset* functor, as it associates to every set X its powerset.

Interestingly, there is a relationship between the functors I and P. Consider a set X. For simplicity, take $X = \{x_1, x_2, \ldots, x_n\}$. (The argument goes through for arbitrary X.) We have $IX = \{x_1, \ldots, x_n, \bot\}$. There is a way to map this set IX into PX that in some sense preserves the behaviour of the set (with respect to morphisms), by mapping an element x_i into $\{x_i\}$, and $\bot$ into $\varnothing$. This map is defined uniformly for all sets X. Such a relationship between functors, that lets one map the image of an object under one functor into its image under the other functor in a uniform way, is an important concept in category theory, that we presently define.[5]

Natural transformations. Given two functors $F : \mathcal{C} \longrightarrow \mathcal{D}$ and $G : \mathcal{C} \longrightarrow \mathcal{D}$, a *natural transformation* η between F and G, written $\eta : F \stackrel{\cdot}{\longrightarrow} G$, is a family $FX \xrightarrow{\eta_X} GX$ of morphisms of $\mathcal{D}$ indexed by objects of $\mathcal{C}$, such that for every morphism $X \xrightarrow{f} Y$ in $\mathcal{C}$, the following diagram commutes:

$$\begin{array}{ccc} FX & \xrightarrow{\eta_X} & GX \\ {\scriptstyle Ff}\downarrow & & \downarrow{\scriptstyle Gf} \\ FY & \xrightarrow{\eta_Y} & GY. \end{array}$$

Thus, a natural transformation η tells us how to map the image of object X under F into its image under G, via the morphism $FX \xrightarrow{\eta_X} GX$.

Note in passing that is we take all functors from $\mathcal{C}$ to $\mathcal{D}$ (for fixed $\mathcal{C}$ and $\mathcal{D}$) and natural transformations between them, this forms a category.

Example. We can check that the relationship between I and P alluded to above is in fact a natural transformation. Define $\eta : I \stackrel{\cdot}{\longrightarrow} P$ by taking the morphism η_X (which is a function from $X \uplus \{\bot\}$ to the powerset of X) as:

$$\eta_X(x) = \begin{cases} \{x\} & \text{if } x \neq \bot \\ \varnothing & \text{if } x = \bot. \end{cases}$$

To prove that η forms a natural transformation, let $X \xrightarrow{f} Y$ be a morphism in $\mathbf{Set}$. We check that the appropriate diagram commutes, that is, for all $x \in X \cup \{\bot\}$, $\eta_Y((If)(x)) = (Pf)(\eta_X(x))$. We check this for $x \neq \bot$ first: $\eta_Y((If)(x)) = \eta_Y(f(x)) = \{f(x)\}$, and $(Pf)(\eta_X(x)) = (Pf)(\{x\}) = \{f(x)\}$, as required. If $x = \bot$, we have: $\eta_Y((If)(\bot)) = \eta_Y(\bot) = \varnothing$, and $(Pf)(\eta_X(\bot)) = (Pf)(\varnothing) = \varnothing$, as required. Thus, η is a natural transformation from I to P.

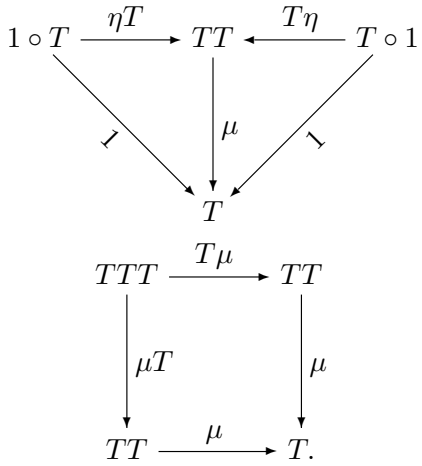
Monads. The motivation for introducing natural transformations is really to get at the next concept, that of a monad. Recall that our goal is to start with a category $\mathcal{C}$ representing the semantics of expressions, and to derive from this a category $\mathcal{D}$ that includes $\mathcal{C}$ in some sense, but where every

[5] According to Saunders Mac Lane, category theory was in fact developed in order to make this concept precise.

object is extended with an undefined element, that will let us interpret nontermination. Focusing on **Set** as the base category for the semantics of expression, we see that the functor I given above essentially does part of the job; it adds to every object a new element $\perp$ that we are free to interpreted as the undefined element. However, we still need to turn this into a category. Just having a functor is not quite enough. We need more structure. And this is what a monad gives us: enough structure to derive a category from a functor.[6]

First, some terminology. A functor $F : \mathcal{C} \longrightarrow \mathcal{C}$ is often called an *endofunctor*. For any category $\mathcal{C}$, the identify functor $1_{\mathcal{C}}$ is well-defined. Let $F, G : \mathcal{C} \longrightarrow \mathcal{D}$ be functors, and $\eta : F \stackrel{\cdot}{\longrightarrow} G$ be a natural transformation. If $T : \mathcal{C} \longrightarrow \mathcal{C}$ is an endofunctor, define the natural transformation $\eta T : FT \stackrel{\cdot}{\longrightarrow} GT$ by taking $\eta T_X = \eta_{TX}$; if $S : \mathcal{D} \longrightarrow \mathcal{D}$ is an endofunctor, define the natural transformation $S\eta : SF \stackrel{\cdot}{\longrightarrow} SG$ by taking $S\eta_X = S(\eta_X)$. It is easy to check that ηT and $S\eta$ are natural tranformations.

A *monad* (also called a triple) in a category $\mathcal{C}$ is a triple (T, η, μ), where $T : \mathcal{C} \longrightarrow \mathcal{C}$ is an endofunctor, and $\eta : 1_{\mathcal{C}} \stackrel{\cdot}{\longrightarrow} T$ and $\mu : TT \stackrel{\cdot}{\longrightarrow} T$ are natural transformations satisfying



(These laws bear a relationship with monoid laws; intuitively, a monad is a monoid in the category $\mathcal{C}^{\mathcal{C}}$ of endofunctors on $\mathcal{C}$, hence the name.)

Examples. It is easy to check that (I, η, μ) is a monad in **Set**, where $\eta_X : X \longrightarrow X \cup \{\perp\}$ and $\mu_X : (X \uplus \{\perp_1\}) \uplus \{\perp_2\} \longrightarrow X \uplus \{\perp_3\}$, if we take:

$$(\eta_X)(x) = x$$
$$(\mu_X)(x) = \begin{cases} x & \text{if } x \neq \perp_1, x \neq \perp_2 \\ \perp_3 & \text{otherwise.} \end{cases}$$

Similarly, it is easy to check that (P, η, μ) is a monad in **Set**, if we take:

$$(\eta_X)(x) = \{x\}$$
$$(\mu_X)(W) = \cup W.$$

[6]The use of monads in semantics goes back to Moggi [1989], who was the first to realize that they could be used to model effects in semantics. This was put to good use in the context of functional programming by Wadler [1992].

Kleisli categories. Why are monads interesting? Among other things, because they let us derive new categories from existing categories.

Let $\mathcal{C}$ be a category and let (T, η, μ) be a monad in $\mathcal{C}$. Construct the category $\mathcal{C}_T$, called the *Kleisli category of the monad* T as follows:

- the objects of $\mathcal{C}_T$ are simply the objects of $\mathcal{C}$;
- a morphism $X \xrightarrow{f} Y$ in $\mathcal{C}_T$ is a morphism $X \xrightarrow{f} TY$ in $\mathcal{C}$.

In order for $\mathcal{C}_T$ to be a category, it needs to have identity morphisms and a composition operator. The natural transformations η and μ in the monad basically give us these two things. (Which is why it was not sufficient to have a functor to be able to derive the category.) More precisely, η_X tells us that for every object X, there is a morphism $X \xrightarrow{\eta_X} TX$ in $\mathcal{C}$, and thus, there is a morphism $X \xrightarrow{\eta_X} X$ in $\mathcal{C}_T$, as required. Composition is more interesting. Suppose we have $X \xrightarrow{f} Y$ and $Y \xrightarrow{g} Z$ in $\mathcal{C}_T$; we want a way to define $g \circ f$ so that $X \xrightarrow{g \circ f} Z$. Consider the corresponding $X \xrightarrow{f} TY$ and $Y \xrightarrow{g} TZ$ in $\mathcal{C}$, and the chain:

$$X \xrightarrow{f} TY \xrightarrow{Tg} TTZ \xrightarrow{\mu_Z} TZ,$$

which is a morphism of the required form. Thus, we can take $g \circ f$ in $\mathcal{C}_T$ to be $\mu_Z \circ Tg \circ f$. Using the monad laws, we can verify that identity and composition behave as they should in $\mathcal{C}_T$, and that $\mathcal{C}_T$ is indeed a category.

A bit of work shows that the Kleisli category of (I, η, μ) on $\mathbf{Set}$ is simply $\mathbf{PFn}$, the category of partial functions. This gives us a systematic way of deriving $\mathbf{PFn}$ from $\mathbf{Set}$, by adjoining an "undefined" element to each set in $\mathbf{Set}$. As it turns out, many interesting categories in semantics are Kleisli categories of suitable monads.

In the past few lectures, we have derived properties for ensuring that a category can be used to give a transition-based semantics to IMP, and gave the semantics in abstract terms based on these properties. This means, concretely, that if we can establish that a particular category has these properties, then we can immediately derive how IMP can be interpreted in that category. But how do you choose a category to start with? Often, this happens when the language has an extension to IMP that requires a particular category to interpret it. By using the abstract semantics, the recipe is to come up with a category to model the addition to IMP, and ensure that the category satisfies axioms A1–3 gives earlier, from which the semantics of the IMP fragment can be derived immediately. We examine two such examples in this lecture.

Nondeterministic choice. Consider the following extension to IMP:

$$
\begin{aligned}
S ::= \;& \\
& \texttt{skip} \\
& x_i := e \\
& S_1; S_2 \\
& \texttt{if } B \texttt{ then } S_1 \texttt{ else } S_2 \\
& \texttt{while } B \texttt{ do } S \\
& S_1 + S_2
\end{aligned}
$$

The additional statement $S_1 + S_2$ is meant to be interpreted as: "choose nondeterministically which statement to execute, S_1 or S_2." We want a category in which to interpret such statements.

Recall that one semantics for IMP that we gave way back in the first lecture was in terms of partial functions between sets, where such a function maps the current state to the potential next state obtained by executing the statement (if the statement ever terminates). We can do something similar here, but we need to associate not a next state to the current state and the program, but rather a *set* of states, those that possibly can be reached by executing the program; because of the nondeterministic choice, many states can be considered as the next state.

The category of sets with relations. Define $\mathbf{Rel}$ to be the category with objects sets (just like $\mathbf{Set}$), but where a morphism $X \xrightarrow{R} Y$ is a *relation* $R \subseteq X \times Y$. If we define the identity morphism $X \xrightarrow{1_X} X$ as the identity relation $1_X = \{(x, x) \mid x \in X\}$ and define composition $S \circ R = \{(x, z) \mid \exists y.(x, y) \in R, (y, z) \in S\}$, it is easy to check that this forms a category.

Properties. We can check that $\mathbf{Rel}$ has all finite products (products are given by disjoint unions), and is a partially additive category (coproducts are given by disjoint unions, just like in $\mathbf{Set}$, and the partially additive structure is given by defining $\sum\{R_i \mid i \in I\} = \cup_{i \in I} R_i$). Moreover, it is easy

to check that $\mathbf{Rel}$ is the Kleisli category of the powerset monad P defined in the last lecture, that is, $\mathbf{Rel}$ is just $\mathbf{Set}_P$.

Semantics of extended IMP. We can use the fact that $\mathbf{Rel}$ satisfies axioms A1–3 to give a semantics to the IMP fragment of our extended IMP, and use the features of the particular category to give a semantics to $S_1 + S_2$. Again, let X be the object in $\mathbf{Rel}$ representing the values of the variables, and assume appropriate morphisms to give a semantics $[\![B]\!]$ and $[\![E]\!]$.

$$
\begin{aligned}
&[\![\texttt{skip}]\!] = 1_{X^n} \\
&[\![x_i := E]\!] = \langle \pi_1, \ldots, [\![E]\!], \ldots, \pi_n \rangle \\
&[\![S_1; S_2]\!] = [\![S_2]\!] \circ [\![S_1]\!] \\
&[\![\texttt{if } B \texttt{ then } S_1 \texttt{ else } S_2]\!] = [[\![S_1]\!], [\![S_2]\!]] \circ [\![B]\!] \\
&[\![\texttt{while } B \texttt{ do } S]\!] = ([\iota_1 \circ [\![S]\!], \iota_2] \circ [\![B]\!])^\dagger \\
&[\![S_1 + S_2]\!] = [\![S_1]\!] \cup [\![S_2]\!].
\end{aligned}
$$

Thus, the semantics of $S_1 + S_2$, in this category, relies on the fact that the morphisms are just sets of tuples in the relation. If we compute, say, what the morphism corresponding to the while loop actually looks like in $\mathbf{Rel}$, then we get the well-known relation corresponding to the reflexive transitive closure of $[\![S]\!]$ subject to the restriction that the final state does not satisfy the test B.

Probabilistic choice. As a different but somewhat related extension, consider extending IMP with a probabilistic choice:

$$
\begin{aligned}
S ::= \quad & \\
& \texttt{skip} \\
& x_i := e \\
& S_1; S_2 \\
& \texttt{if } B \texttt{ then } S_1 \texttt{ else } S_2 \\
& \texttt{while } B \texttt{ do } S \\
& S_1 +_p S_2
\end{aligned}
$$

Here, $S_1 +_p S_2$ is meant to be interpreted as: "execute S_1 with probability p, and execute S_2 with probability $1 - p$.". We again want a category in which to interpret such statements.

The intuition is quite similar to that of the nondeterministic-choice extension seen above. Roughly speaking, instead of associating with a statement a function from the current state to a set of possible states, we associate with a statement a measure that gives the probability, from any current state, of reaching a state in a set of possible state. (See Kozen [1981, 1985] for a direct account of such a semantics and the basic motivation.)

To do this, rather than take a category based on sets as a base category, we take a category based on measurable spaces.[7] Recall that a measurable space is a pair $(X, \mathcal{M})$ where X is a set, and $\mathcal{M}$

[7] See Billingsley [1995], for instance.

is a σ-algebra of subsets of X, that is, $\mathcal{M}$ is closed under complements and countable unions. A function $f : (X, \mathcal{M}) \longrightarrow (Y, \mathcal{N})$ between measurable spaces is measurable if $f^{-1}(E) \in \mathcal{M}$ for all $E \in \mathcal{N}$. A probability measure μ on $(X, \mathcal{M})$ is a function $\mu : \mathcal{M} \longrightarrow [0, 1]$ such that $\mu(X) = 1$ and for $E_1, E_2, \ldots$ a countable sequence of disjoint sets in $\mathcal{M}$, $\mu(\cup_{i<\omega} E_i) = \sum_{i<\omega} \mu(E_i)$. A subprobability measure is like a probability measure, except that we only require $\mu(X) \leq 1$.

The category of stochastic relations. Define $\mathbf{SRel}$ to be the category with objects measurable spaces and where a morphism $(X, \mathcal{M}) \xrightarrow{f} (Y, \mathcal{N})$ is a regular conditional subprobability measure, that is, a function $f : X \times \mathcal{N} \longrightarrow [0, 1]$ such that

(1) for each fixed $B \in \mathcal{N}$, the function $f(\cdot, B)$ is measurable;

(2) for each fixed $x \in X$, $f(x, \cdot)$ is a subprobability measure on $(Y, \mathcal{N})$.

The identity morphism is the so-called Dirac delta function, the function δ defined by

$$\delta(x, A) = \begin{cases} 1 & \text{if } x \in A \\ 0 & \text{otherwise.} \end{cases}$$

The composition rule is as follows. Suppose that $(X, \mathcal{M}) \xrightarrow{f} (Y, \mathcal{N})$ and $(Y, \mathcal{N}) \xrightarrow{g} (Z, \mathcal{O})$. Define $g \circ f$ by

$$(g \circ f)(x, C) = \int_Y g(y, C) f(x, dy).$$

Recall that $f(x, \cdot)$ is a subprobability measure, so the above is an integral with respect to a measure, in the measure-theoretic sense. It is a straightforward exercise in measure theory to check that $\mathbf{SRel}$ is in fact a category. See Panangaden [1999] for the details. One way to think of this category is as a category of a form of Markov chains, where the space space is arbitrary. If we restrict to finite spaces and take the set of all subsets as the measurable sets, then a morphism becomes simply a stochastic matrix, and composition is matrix mulitplication. (Because of this, the morphisms above are sometimes called Markov kernels, a generalization of stochastic matrices, modulo the fact that they define subprobability measures rather than probability measures.)

It is an interesting exercise to prove that $\mathbf{SRel}$ has finite products (obtained by taking the product of measurable spaces), countable coproducts (obtained by taking the disjoint union of measurable spaces with the σ-algebra generated by the measurable sets of each summand), and is a partially additive category (where the partially additive structure is given by defining $(\sum\{f_i \mid i \in I\})(x, A) = \sum_{i \in I} f_i(x, A)$, as long as the result is a subprobability measure, that is, as long as $\sum_{i \in I} f_i(x, A) \leq 1$ for all x and A) [Panangaden 1999; Haghverdi 2000].

It turns out that $\mathbf{SRel}$ is also a Kleisli category for a monad related to the powerset monad. The monad (Π, η, μ), a variant of a monad originally described by Giry [1981], is defined as follows. Let $\mathbf{Meas}$ be the category of measurable spaces, with objects measurable spaces, and morphisms measurable functions. The endofunctor $\Pi : \mathbf{Meas} \longrightarrow \mathbf{Meas}$ associates to every measurable space $(X, \mathcal{M})$ the set of all subprobability measures $\{\nu \mid \nu \text{ is a subprobability measure on } (X, \mathcal{M})\}$,

equipped with the least σ-algebra such that all the functions p_A (for $A \in \mathcal{M}$) given by $p_A(\nu) = \nu(A)$ are measuable. The natural transformation η is given by the morphisms $(X, \mathcal{M}) \xrightarrow{\eta_X} \Pi(X, \mathcal{M})$ defined by

$$\eta_X(x, A) = \delta(x, A)$$

and the natural transformation μ is given by the morphisms $\Pi\Pi(X, \mathcal{M}) \xrightarrow{\mu_X} \Pi(X, \mathcal{M})$ defined by

$$\mu_X(\Sigma, B) = \int_{\Pi(X,\mathcal{M})} p_B \Sigma.$$

(See Giry [1981] or Panangaden [1999] for careful motivation of these definitions.)

Semantics of extended IMP. Here again, we can use the fact that $\mathbf{SRel}$ satisfies axioms A1–3 to give a semantics to the IMP fragment of our extended IMP, and use the features of $\mathbf{SRel}$ to give a semantics to $S_1 +_p S_2$. Let X be the object in $\mathbf{SRel}$ representing the values of the variables (notice that this is now a measurable space), and assume appropriate morphisms to give a semantics $[\![B]\!]$ and $[\![E]\!]$.

$$\begin{aligned}
&[\![\texttt{skip}]\!] = 1_{X^n} \\
&[\![x_i := E]\!] = \langle \pi_1, \ldots, [\![E]\!], \ldots, \pi_n \rangle \\
&[\![S_1; S_2]\!] = [\![S_2]\!] \circ [\![S_1]\!] \\
&[\![\texttt{if } B \texttt{ then } S_1 \texttt{ else } S_2]\!] = [[\![S_1]\!], [\![S_2]\!]] \circ [\![B]\!] \\
&[\![\texttt{while } B \texttt{ do } S]\!] = ([\iota_1 \circ [\![S]\!], \iota_2] \circ [\![B]\!])^\dagger \\
&[\![S_1 +_p S_2]\!](x, A) = p[\![S_1]\!](x, A) + (1 - p)[\![S_2]\!](x, A).
\end{aligned}$$

Thus, the semantics of $S_1 +_p S_2$, in this category, relies on the fact that the morphisms are just regular conditional subprobability measures. If we compute what the morphisms above actually look like in $\mathbf{SRel}$, then we recover the semantics due to Kozen [1981, 1985].

We can also accomodate, with this semantics, an expression in E such as $random$, which returns a random value (assume that the space of values X is finite for this to really make sense). This simply calls for associating a morphism $(X^n, \mathcal{M}^n) \xrightarrow{[\![random]\!]} (X, \mathcal{M})$ defined by $[\![random]\!](x, A) = |A|/|X|$.

References

Bakker, J. de and E. de Vink (1996). *Control Flow Semantics*. MIT Press.

Barr, M. and C. Wells (1990). *Category Theory for Computing Science*. Prentice-Hall.

Billingsley, P. (1995). *Probability and Measure*. Wiley Series in Probability and Mathematical Statistics. John Wiley & Sons.

Bloom, S. and Z. Esik (1993). *Iteration Theories*. EATCS Monographs on Theoretical Computer Science. Springer-Verlag.

Elgot, C. C. (1975). Monadic computation and iterative algebraic theories. In *Proc. Logic Colloquium '73*. North-Holland.

Giry, M. (1981). A categorical approach to probability theory. In B. Banaschewski (Ed.), *Categorical Aspects of Topology and Analysis*, Volume 915 of *Lecture Notes in Mathematics*, pp. 68–85. Springer-Verlag.

Gunter, C. A. (1992). *Semantics of Programming Languages: Structures and Techniques*. MIT Press.

Haghverdi, E. (2000). *A Categorical Approach to Linear Logic, Geometry of Proofs and Full Completeness*. Ph. D. thesis, University of Ottawa.

Joyal, A., R. Street, and D. Verity (1996). Traced monoidal categories. *Mathematical Proceedings of the Cambridge Philosophical Society 119*(3), 447–468.

Kozen, D. (1981). Semantics of probabilistic programs. *Journal of Computer and Systems Sciences 22*(3), 328–350.

Kozen, D. (1985). A probabilistic PDL. *Journal of Computer and Systems Sciences 30*(2), 162–178.

Mac Lane, S. (1971). *Categories for the Working Mathematician*. Number 5 in Graduate Texts in Mathematics. Springer-Verlag.

Manes, E. and M. Arbib (1986). *Algebraic Approaches to Program Semantics*. Springer-Verlag.

Moggi, E. (1989). Computational lambda-calculus and monads. In *Proc. 4th Annual IEEE Symposium on Logic in Computer Science (LICS'89)*, pp. 14–23. IEEE Computer Society Press.

Panangaden, P. (1999). The category of Markov kernels. In *Proc. 1st International Workshop on Probabilistic Methods in Verification (PROBMIV'98)*, Volume 22 of *Electronic Notes in Theoretical Computer Science*. Elsevier Science Publishers.

Plotkin, G. D. (1983). Domains. Pisa Notes on Domain Theory.

Wadler, P. (1992). The essence of functional programming. In *Proc. 19th Annual ACM Symposium on Principles of Programming Languages (POPL'92)*, pp. 1–14. ACM Press.

Winskel, G. (1993). *The Formal Semantics of Programming Languages*. MIT Press.