1988

A taste of category theory for computer scientists

Benjamin C. Pierce
Carnegie Mellon University

A Taste of Category Theory for Computer Scientists

Benjamin C. Pierce

CMU-CS-88-203 2

Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213

This work was supported in part by DEC Systems Research Center and in part by the Office of Naval Research and the Defense Advanced Research Projects Agency (DOD) under contract number N00014-84-K-0415.

Abstract

Category theory is a field that impinges more and more frequently on the awareness of many computer scientists, especially those with an interest in programming languages and formal specifications. But there is still disagreement as to the real significance of category theory, even among specialists in these areas. Some dismiss it as "abstract nonsense," whereas for others it has become an important tool. Worse yet for the uninitiated, there are only a few introductory textbooks on the subject—and none of them are easy reading. Although the category theory literature is more accessible than it was a few years ago, there is still no good way to get an impression with only a modest amount of effort.

This paper is an "introduction to the introductions" to category theory—a brief answer to the questions, "What is category theory?" "What are computer scientists using it for?" "What are the basic concepts?" "Where can I learn more?" The first section aims to ground the reader in the most common categorical terms and idioms, assuming as little specific mathematical background as possible. The second section presents four case studies from the recent research literature applying category theory to the semantics of computation. A reading list at the end of the paper suggests pathways into the existing literature—including textbooks, standard reference works, and a sampling of important research papers.

Contents

Chapter 1

Introduction

...Our intention is not to use any deep theorems of category theory, but merely to employ the basic concepts of this field as organizing principles. This might appear as a desire to be concise at the expense of being esoteric. But in designing a programming language, the central problem is to organize a variety of concepts in a way which exhibits uniformity and generality. Substantial leverage can be gained in attacking this problem if these concepts are defined concisely within a framework which has already proven its ability to impose uniformity and generality upon a wide variety of mathematics.

— Reynolds [69]

Category theory is a relatively young branch of pure mathematics, stemming from an area—algebraic topology—that most computer scientists would consider esoteric. Yet its influence is being felt in many parts of theoretical computer science. A short list of important connections would include the design of both functional and imperative programming languages, implementation techniques for functional languages, semantic models of programming languages, semantics of concurrency, specification and development of algorithms, type theory and polymorphism, specification languages, algebraic semantics, constructive logic, and automata theory.

The breadth of this list points out an important point: category theory is not geared to work in a particular setting; it is a foundational framework in the same sense as set theory or graph theory. The difference is that it is more structured than, relying on more abstract constructions and heavier notation. The cost, of course, is that categorical formulations of concepts are often more difficult to grasp than their counterparts in other formalisms. The benefit is that concepts may be dealt with at a higher level and hidden commonalities allowed to emerge.

Recent issues of theoretical journals give ample evidence that, at least in some parts of computer science, category theory is already an important tool. In a few areas—notably domain theory and semantics of computation—it is now a standard language of discourse. But there are conflicting opinions in the research community on the question of how much category theory a computer scientist should know. Some authors see no use in bothering with it at all unless you become a serious student of the subject.

On the other hand, most computer science research papers draw only on the notation and some fairly shallow results of category theory. The "ADJ group," early proponents of category theory in computer science, set a reassuring note in the introduction to one of their papers [91]: "...do not succumb to a feeling that you must understand *all* of category theory before you put it to use. When one talks of a 'set theoretic' model for some computing phenomenon, [one] is not thinking of a formulation in terms of measurable cardinals! Similarly, a category theoretic model does not *necessarily* involve the Kan extension theorem or double categories."

The tutorial in Section 2 of this paper is intended to provide a deep enough treatment of basic category theory that the reader will feel prepared to tackle some of the current research papers applying category theory to computer science. It covers all of the essential notation and constructions, and also a few more advanced topics (notably adjoints) that are sometimes skipped in short introductions to the subject, but that seem relevant to an appreciation of the field. Section 3 illustrates the concepts presented in the tutorial with two applications to the design of programming languages, a summary of work in categorical models of the semantics of programming languages, and a more detailed description of category-theoretic tools for the solution of recursive domain equations. Section 4 briefly surveys the available textbooks, introductory articles, and reference works on category theory. A summary of notation used in the tutorial appears at the end.

Before moving on to the tutorial, I would like to express my gratitude to Nico Habermann for suggesting this project, to DEC Systems Research Center and Carnegie Mellon University for support while the paper was being written, and to Rod Burstall, Luca Cardelli, Peter Freyd, Peppe Longo, Simone Martini, Gordon Plotkin, John Reynolds, and Dana Scott for informative conversations. Comments and suggestions from Martín Abadi Violetta Cavalli-Sforza, Scott Dietzen, Conal Elliott, Nico Habermann, Nevin Heintze, Peter Lee, Mark Maimone, Spiro Michaylov, David Plaut, and John Reynolds have greatly improved my presentation of the material and eliminated a number of errors in previous drafts. I would also like to acknowledge an enormous debt to the labors of other authors, particularly to Robert Goldblatt [31], Saunders Mac Lane [54], and David Rydeheard [76,77,78]. Their books (along with many others) were frequent guides in choice of examples and exercises, organization of material, and proper presentation of the subject's "folklore." (There are some points—marked in the text—where their presentations of concepts or examples seemed sufficiently unimprovable that I chose to use them essentially verbatim. Errors introduced during transcription of these sections are, like those remaining in the rest of the text, my own responsibility.)

Chapter 2

Tutorial

This section is a brief tutorial on the most important concepts of category theory. Its purpose is threefold: first, to be complete enough that after finishing it the reader can proceed to a more difficult textbook—or even directly to research papers applying category theory to computer science—with relative ease; second, to cover the more advanced topics of natural transformations and adjoints in sufficient depth that the reader comes away with some sense of the contribution of category theory to mathematical thinking; and third, to be reasonably short. The tension between these three dictates a fairly terse presentation, with examples in each subsection chosen to simultaneously illustrate a concept and its applications. Usually a subsection will begin with a rigorous definition, perhaps prefaced with an intuitive explanation of the construction, followed by examples of its use in various contexts. Optional examples and sections (marked with a †) are not required for understanding what follows, except for subsequent optional parts. A few exercises are provided for the benefit of readers with time to spend on them.

2.1 Categories and Diagrams

What we are probably seeking is a "purer" view of functions: a theory of functions in themselves, not a theory of functions derived from sets. What, then, is a pure theory of functions? Answer: category theory.

— Scott [86, p. 406]

Category theory, like set theory, is a fundamental for mathematical discourse. In set theory, the most primitive concept is *element*. Sets are built out of elements; relations, functions, and the whole host of other mathematical entities are special kinds of sets. In category theory it is *functions*—or more abstractly, *arrows*—that are primitive. The source and target of an arrow are simply called *objects*; whatever internal structure they may possess is ignored. Similarly, an arrow between two objects is an atomic entity, described not in terms of its action on elements of its source, but instead by its properties under composition with other arrows.

The easiest way to define the notion of "category" is in terms of sets and functions. There is nothing contradictory about this: category theory is not being put forward as a *competitor* to set theory, but as an alternative with a different emphasis. In fact, our first example will be the category whose objects are sets and whose arrows are functions: throughout the tutorial, this category is an important source of intuition for more general constructions. There have been various attempts to develop alternative foundations for category theory, but they lie outside the scope of this discussion. For present purposes, "collections" are just sets (or possibly proper classes, since we will have occasion to talk about things like the "class of all sets," which is too big to be a set); "operations" are set-theoretic functions; "equality" is set-theoretic identity.

1 Definition A *category* $\mathbf{C}$ comprises

1. a collection of *objects*;
2. a collection of *arrows* (or *morphisms*);
3. operations assigning to each arrow f an object dom f, its *domain*, and an object cod f, its *codomain* (we write $f : A \to B$ or $A \xrightarrow{f} B$ to show that $A = \text{dom } f$ and $B = \text{cod } f$; the collection of all arrows with domain A and codomain B is written $\mathbf{C}(A, B)$);
4. a composition operator assigning to each pair of arrows f and g, with dom g = cod f, a *composite* arrow $g \circ f : \text{dom } f \to \text{cod } g$, such that the following law holds:

   Associative Law: for any arrows $f : A \to B$, $g : B \to C$, and $h : C \to D$ (with A, B, C, and D not necessarily distinct),

   $$h \circ (g \circ f) = (h \circ g) \circ f;$$

5. for each object A, an *identity* arrow $\text{id}_A : A \to A$, such that the following law holds:

   Identity Law: for any arrow $f : A \to B$,

   $$\text{id}_B \circ f = f \quad \text{and} \quad f \circ \text{id}_A = f.$$

2 Example The category **Set** has sets as objects and total functions between sets as arrows. Composition of arrows is set-theoretic function composition. Identity arrows are identity functions.

To see that **Set** is actually a category, let us restate its definition in the same format as Definition 1 and check that the laws hold:

1. An object in **Set** is a set.

2. An arrow $f : A \to B$ in **Set** is a total function from A into B. There is a small but important difference between this definition and the usual definition of sets and functions in set theory. In the category **Set**, the codomain of a function (the set B) is not the same as its range (the elements of B that are actually the value of f on some element of A). For example, the identity function on the nonnegative integers is *not* the same arrow as the inclusion function from the nonnegative integers to the integers, although they have the same function graph.

3. For each function f with domain A and codomain B, $\mathrm{dom}\, f = A$, $\mathrm{cod}\, f = B$, and $f \in \mathbf{Set}(A, B)$.

4. The composition of a total function $f : A \to B$ with another total function $g : B \to C$ is a total function from A to C. Composition of total functions on sets is associative: for any functions $f : A \to B$, $g : B \to C$, and $h : C \to D$, it is the case that $h \circ (g \circ f) = (h \circ g) \circ f$.

5. For each set A, the identity function id_A is a total function with domain and codomain A. For any function $f : A \to B$, the identity functions on A and B satisfy the equations required by the identity law: $\mathrm{id}_B \circ f = f$ and $f \circ \mathrm{id}_A = f$.

The category laws are so obviously satisfied by **Set** that it may seem as though Definition 1 is vacuous. The next example shows that there is usually some work to be done in showing that some given collections of objects and arrows form a category. (In more involved examples than these, the work can be substantial!)

3 Example A partial ordering $\leq$ on a set P is a reflexive, transitive, and antisymmetric relation on the elements of P. An order-preserving function from $(P, \leq_P)$ to $(Q, \leq_Q)$ is a function $f : P \to Q$ such that if $p \leq_P p'$, then $f(p) \leq_Q f(p')$.

The category **Poset** has as objects all partially-ordered sets and as arrows all order-preserving functions.

Let us go once more through the exercise of checking this carefully against Definition 1:

1. An object in **Poset** is a set P with a reflexive, transitive, antisymmetric relation $\leq_P$ defined on pairs of elements of P.

2. An arrow $f : (P, \leq_P) \to (Q, \leq_Q)$ in **Poset** is a total function from P into Q that preserves the ordering on P, that is, such that if $p \leq_P p'$, then $f(p) \leq_Q f(p')$.

3. For each total order-preserving function f with domain P and codomain Q, dom $f = (P, \leq_P)$, cod $f = (Q, \leq_Q)$, and $f \in \mathbf{Poset}((P, \leq_P), (Q, \leq_Q))$.

4. The composition of an total order-preserving function $f : P \to Q$ with another total order-preserving function $g : Q \to R$ is a total function $g \circ f$ from P to R. Furthermore, if $p \leq_P p'$, then since f preserves P's ordering, $f(p) \leq_Q f(p')$; then, since g preserves Q's ordering, $g(f(p)) \leq_R g(f(p'))$. So $g \circ f$ is order-preserving. Composition of order-preserving functions is associative because each order-preserving function on partially-ordered sets is just a function on sets, and composition of functions on sets is associative.

5. For each partial order $(P, \leq_P)$, the identity function id_P preserves the ordering on P and satisfies the equations of the identity law.

Another familiar algebraic structure, the monoid, also forms the basis of a category:

4 Example A monoid $(M, \cdot, e)$ is a set M equipped with a binary operation $\cdot$ from pairs of elements of M into M and a distinguished element e, such that $(x \cdot y) \cdot z = x \cdot (y \cdot z)$ for all $x, y, z \in M$ and $e \cdot x = x = x \cdot e$ for all $x \in M$. A monoid homomorphism from $(M, \cdot, e)$ to $(M', \cdot', e')$ is a function $f : M \to M'$ such that $f(e) = e'$ and $f(x \cdot y) = f(x) \cdot' f(y)$. The composition of two monoid homomorphisms is the same as their composition as functions on sets. (Monoids are sometimes called "semigroups with identity.")

The category **Mon** has monoids as objects and monoid homomorphisms as arrows. The check that **Mon** is actually a category follows exactly the same steps as for **Poset**. It is easy to check that the composition of two functions that are both homomorphisms is also a homomorphism.

A more general class of categories can be formed by taking all algebras with a given signature as the objects of a category:

5 Example Let Ω be a set of operator symbols, equipped with a mapping ar from elements of Ω to natural numbers (for each $o \in \Omega$, $ar(o)$ is the arity of o). An Ω-algebra A is a set $|A|$ (the carrier of A) together with, for each operator o of arity $ar(o)$, a function $a_o : |A|^{ar(o)} \to |A|$ (the interpretation of o, mapping $ar(o)$-tuples of elements of the carrier back into the carrier). An Ω-homomorphism from an Ω-algebra A to an Ω-algebra B is a function $h : |A| \to |B|$ such that for each operator $o \in \Omega$ and tuple $x_1, x_2 \ldots, x_{ar(o)}$ of elements of $|A|$, the following equation holds:

$$h(a_o(x_1, x_2, \ldots, x_{ar(o)})) = b_o(h(x_1), h(x_2), \ldots, h(x_{ar(o)})).$$

The category **Ω-Alg** has Ω-algebras as objects and Ω-homomorphisms as arrows.

In general, the objects in many categories can be thought of as "sets with structure," and the arrows as "structure preserving maps." (Such categories are often called *concrete categories*.) From this perspective, it is easy to see why category theory is often described as a generalization of universal algebra [14], which studies the common properties of algebraic structures. The following table (adapted from Goldblatt [31]) lists a few of the categories that fit this intuition:

CATEGORY	OBJECTS	MORPHISMS
Set	all sets	all total functions between sets
FinSet	finite sets	functions between finite sets
Pfn	sets	partial functions between sets
Rel	sets	binary relations between sets
Vect	vector spaces	linear transforms
Grp	groups	group homomorphisms
Mon	monoids	monoid homomorphisms
Poset	posets	monotone functions
CPO	complete partial orders	continuous functions
Met	metric spaces	contraction maps
Top	topological spaces	continuous functions
Ω-Alg	algebras with signature Ω	Ω-homomorphisms

The concrete categories form an important class, but there are many other interesting categories. For example, here are a few useful *finite* categories:

6 Example The category **0** has no objects and no arrows. The identity and associativity laws are vacuously satisfied.

7 Example The category **1** has one object and one arrow. By the identity law, the arrow must be the identity for the object. The composition of this arrow with itself can only be itself, which satisfies the identity and associativity laws. (Note that we didn't bother to specify the precise identity of the object or arrow—for example, that the object was the number 5 and the arrow was the total function mapping 5 to 5. What matters is only the algebraic properties of the object and arrow, whatever they are, and these properties are fully determined by the category laws.)

8 Example The category **2** has two objects, two identity arrows, and an arrow from one object to the other. Again, it doesn't matter what the objects and arrows represent, but to make it easier to talk about them, we might call the objects A and B and the non-identity arrow f. There is only one way to define composition; it is then easy to check that the identity and associativity laws are satisfied.

9 Example The category **3** has three objects (call them A, B, and C), three identity arrows, and three other arrows: $f : A \to B$, $g : B \to C$, and $h : A \to C$. (Again, composition can be defined in only one way and both laws are satisfied.)

The last example is beginning to be complicated enough that it is hard to grasp from a bare listing of the objects and arrows. To make such descriptions more manageable, category theorists often use a graphical style of presentation:

10 Definition A *diagram* in a category **C** is a collection of vertices and directed edges, consistently labeled with objects and arrows of **C** (where by "consistently" we mean that if an edge in the diagram is labeled with the arrow f and f has domain A and codomain B, then the endpoints of this edge must be labeled with A and B).

In this notation, the categories **2** and **3** are displayed as follows:

$$\begin{tikzcd} A \arrow[loop left, "\mathrm{id}_A"] \arrow[r, "f"] & B \arrow[loop right, "\mathrm{id}_B"] \end{tikzcd} \qquad \begin{tikzcd} & B \arrow[loop right, "\mathrm{id}_B"] \arrow[dr, "g"] & \\ A \arrow[loop left, "\mathrm{id}_A"] \arrow[ur, "f"] \arrow[rr, "h"'] & & C \arrow[loop right, "\mathrm{id}_C"] \end{tikzcd}$$

(Most authors blur the distinction between the vertices and edges in a diagram and the objects and arrows with which they are labeled. Also, it is unusual for a diagram to display the *whole* of a category as these two do. Diagrams are most often used to illustrate some configuration of a small number of objects and arrows of particular interest *within* some category.)
Note that since f, g, and h are the only non-identity arrows of **3**, it must be the case that $g \circ f = h$. (Why?)

Diagrams are used in category theory not only for visualizing categories, but also (and much more importantly) for stating and proving properties of categorical constructions. Such properties are often expressed by saying that a particular diagram "commutes."

11 Definition A diagram in a category **C** is said to *commute* if, for every pair of vertices X and Y, all the paths in the diagram from X to Y are equal (in the sense that each path in the diagram determines a composite arrow and these composites are equal in **C**). For example, saying that "the diagram

$$\begin{tikzcd} X \arrow[r, "f'"] \arrow[d, "g'"'] & Z \arrow[d, "g"] \\ W \arrow[r, "f"'] & Y \end{tikzcd}$$

commutes" is exactly the same as saying that $f \circ g' = g \circ f'$.

A useful refinement of this convention is to require that two paths be equal only when one of them is longer than a single arrow. Thus the commutativity of the diagram

$$\begin{tikzcd} X \arrow[r, shift left, "f"] \arrow[r, shift right, "g"'] & Y \arrow[r, "h"'] & Z \end{tikzcd}$$

implies that $h \circ f = h \circ g$, but not that $f = g$.

When a property is stated in terms of commutative diagrams, proofs involving that property can often be given directly by "diagram chasing." The following simple proof demonstrates the technique. (Observe how the equations correspond to paths in the diagram, and how these paths are transformed by replacing one path through a commuting subdiagram with another.)

12 Proposition If both inner squares of the following diagram commute, then so does the outer rectangle.

$$\begin{tikzcd}
\cdot \arrow[r, "f"] \arrow[d, "a"'] & \cdot \arrow[r, "f'"] \arrow[d, "b"] & \cdot \arrow[d, "c"] \\
\cdot \arrow[r, "g"'] & \cdot \arrow[r, "g'"'] & \cdot
\end{tikzcd}$$

Proof:

$$\begin{aligned}
(g' \circ g) \circ a &= g' \circ (g \circ a) && \text{(by associativity)} \\
&= g' \circ (b \circ f) && \text{(by the commutativity of the first square)} \\
&= (g' \circ b) \circ f && \text{(by associativity)} \\
&= (c \circ f') \circ f && \text{(by the commutativity of the second square)} \\
&= c \circ (f' \circ f) && \text{(by associativity).}
\end{aligned}$$

(*End of Proof*)

A different class of categories is obtained by considering an *individual* algebraic structure as forming a category.

13 Example A poset $(P, \leq)$ gives rise to a category whose objects are the elements of P. Between each pair of objects p and p' such that $p \leq p'$ there is a single arrow (representing this fact). There is no arrow between two objects p and p' when $p \not\leq p'$. Composition of arrows is clearly associative (since there is at most one arrow between any given pair of objects). The identity law of Definition 1 now corresponds to the reflexivity condition for partial orders, while the existence of composite arrows corresponds to transitivity. The antisymmetry condition is actually not required: in fact, every preorder (set P with transitive and reflexive relation $\leq$) gives rise to a category.

14 Example A monoid $(M, \cdot, e)$ may be represented by a category with a single object. The elements of M are represented by arrows from this object to itself, the identity element e is represented by the identity arrow, and the operation $\cdot$ is represented by composition of arrows. Conversely, any category with a single object gives rise to a monoid. (We will not bother to check the category axioms for this and future examples. The reader is still encouraged to do so, however.)

Many branches of mathematics besides algebra have proved amenable to categorical treatment. Of particular interest to computer science is the new field of "categorical logic," which arises from the following observation.

15 Example By a twist of perspective, we can call the objects in an arbitrary category *formulas* and the arrows *proofs*. An arrow $f : A \rightarrow B$ is viewed as a proof of the implication $A \rightarrow B$. In particular, the identity arrow $\mathrm{id}_A : A \rightarrow A$ is an instance of the reflexivity axiom, and the composition of arrows

$$\frac{f : A \rightarrow B \quad g : B \rightarrow C}{g \circ f : A \rightarrow C}$$

is a rule of inference asserting the transitivity of implication.

In addition to categories of mathematical objects from other domains, there are many kinds of categories that can be built up from simpler categories.

16 Example For each category **C**, the objects of the *dual category* $\mathbf{C}^{\mathrm{op}}$ are the same as those of **C**; the arrows in $\mathbf{C}^{\mathrm{op}}$ are the opposites of the arrows in **C**. (That is, if $f : A \rightarrow B$ is a arrow of **C**, then $f : B \rightarrow A$ is a arrow of $\mathbf{C}^{\mathrm{op}}$.) Composite and identity arrows are defined in the obvious way.

Each of the definitions and theorems of category theory can be restated "with arrows reversed" as an equivalent definition or theorem in the dual category. In fact, most definitions come in pairs—product/coproduct, equalizer/coequalizer, monomorphism/epimorphism, pullback/pushout—with a "co-x" in a category **C** being the same thing as an "x" in $\mathbf{C}^{\mathrm{op}}$.

17 Example For any pair of categories **C** and **D**, the *product category* $\mathbf{C}\times\mathbf{D}$ has as objects pairs $\langle A, B\rangle$ of a **C**-object A and a **D**-object B, and as arrows pairs $\langle f, g\rangle$ of a **C**-arrow f and a **D**-arrow g. Composition and identity arrows are defined pairwise: $\langle f, g\rangle \circ \langle h, i\rangle = \langle f \circ h, g \circ i\rangle$ and $\mathrm{id}_{\langle A,B\rangle} = \langle \mathrm{id}_A, \mathrm{id}_B\rangle$.

18 Example $\mathbf{Set}^{\rightarrow}$ is the *category of arrows* of **Set**. Each arrow $f : A \rightarrow B$ of **Set** is an object of $\mathbf{Set}^{\rightarrow}$. A $\mathbf{Set}^{\rightarrow}$-arrow from $f : A \rightarrow B$ to $f' : A' \rightarrow B'$ is a pair $\langle a, b\rangle$ of **Set**-arrows such that the diagram

$$\begin{tikzcd}
B' \arrow[r, "a"] \arrow[d, "f"'] & A' \arrow[d, "f'"] \\
B \arrow[r, "b"'] & A
\end{tikzcd}$$

commutes (in **Set**). Composition in $\mathbf{Set}^{\rightarrow}$ is defined by $\langle a', b'\rangle \circ \langle a, b\rangle = \langle a' \circ a, b' \circ b\rangle$. (This is a difficult example, included to give a little practice with the mind-bending feats of abstract manipulation sometimes required to follow categorical arguments. Try checking the category axioms with pencil and paper.)

Actually, this construction is completely general. For each category **C** we can define the category of arrows over **C** by substituting **C** for **Set** above.

19 Exercises

1. A group $(G, \cdot, ^{-1}, e)$ is a set G equipped with an operation $\cdot$ from pairs of elements of G into G, a unary operation $^{-1}$, and a distinguished element e, such that:
   (a) $(x \cdot y) \cdot z = x \cdot (y \cdot z)$ for all x, y, and z in G;
   (b) $e \cdot x = x = x \cdot e$ for all x in G;
   (c) $x \cdot x^{-1} = e = x^{-1} \cdot x$ for all x in G.

   Show how an arbitrary group can be considered as a category. (Hint: see Example 14.)
2. Show how an arbitrary set can be considered as a category. (Hint: see Example 13.)
3. Verify that each of the categories **0**, **1**, **2**, and **3** corresponds to a partial order. What would the category **4** look like? The category **5**? The category **Omega**?

20 Remarks†

- Some authors, especially computer scientists, prefer to think of composites in "diagrammatic" order rather than "functional," writing $f;g$ instead of $g \circ f$. The notation given in Definition 1, though less convenient for some purposes, is standard and will be used exclusively in this paper.
- Our definition of category is actually somewhat sloppy: It says that each arrow has a particular domain and codomain—that is, that each arrow goes between two particular objects. So we would expect $\mathbf{C}(A, B) \cap \mathbf{C}(A', B') = \{\}$ unless $A = A'$ and $B = B'$. But we also said that a function could be considered as an arrow from its domain to any set containing its range.

  This inconsistency could be rectified in at least two ways: First, the sets of arrows between distinct pairs of objects in a category can be allowed to overlap. Second, the notion of "function" (and similar notions like "relation") can be refined to explicitly include a codomain as part of the function's identity. Some authors (Reynolds, for example) favor the first approach, where notions like "function" have their ordinary mathematical meaning. The cost of this approach is some extra complication in later definitions (e.g., the definition of "functor"). Following what seems to be the more standard practice, we will use the second approach in this paper, though (also following standard practice) we will not bother to explicitly carry through the details of making the sets of arrows be disjoint.

According to Mac Lane [54, pp. 29-30], the fundamental idea of representing a function by an arrow first appeared in topology around 1940, probably in the work of Hurewicz (c.f. [41]). Commutative diagrams were probably also first used by Hurewicz. Categories, functors (Definition 60 below), and natural transformations (Definition 69) were discovered by Eilenberg and Mac Lane [19], who also used commutative diagrams for the first time in print. A direct treatment of categories in their own right appeared in 1945 [18]. The word "category" was borrowed from Aristotle and Kant, "functor" from Carnap, and "natural transformation" from the informal practice of the time.

2.2 Monomorphisms, Epimorphisms, and Isomorphisms

When we reason about sets and functions, we are often interested in functions with special properties, such as being injective (one-to-one) or surjective (onto) or defining an isomorphism. Appropriate analogues of these concepts also play an important role in categorical reasoning.

21 Definition An arrow $f : B \to C$ in a category **C** is a *monomorphism* (or just *monic*) if for any pair of **C**-arrows $g : A \to B$ and $h : A \to B$ with the same domain and with codomain B, the equality $f \circ g = f \circ h$ implies that $g = h$.

22 Proposition In **Set**, the monomorphisms are just the injective functions.

Proof: Let $f : B \to C$ be an injective function, and let $g, h : A \to B$ be such that $f \circ g = f \circ h$ but $g \neq h$. Then there is some element $a \in A$ for which $g(a) \neq h(a)$. But then, since f is injective, $f(g(a)) \neq f(h(a))$, which contradicts our assumption that $f \circ g = f \circ h$. This shows that f is a monomorphism.
Conversely, let $f : B \to C$ be a monomorphism. If f is not injective, then there are unequal elements $b, b' \in B$ for which $f(b) = f(b')$. Let A be the one-element set $\{a\}$, and let $g : A \to B$ map a to b while $h : A \to B$ maps a to b'. Then $f(g(a)) = f(h(a))$, contradicting the assumption that f is a monomorphism. *(End of Proof)*

23 Convention The category to which an object or arrow belongs is often omitted when it is unimportant or clear from context. We adopt this convention from now on, using explicit qualifications like "for any pair of **C**-arrows..." rather than "for any pair of arrows..." only when there is a possibility of confusion.

24 Definition An arrow $f : A \to B$ is a *epimorphism* (or just *epic*) if for any pair of arrows $g : B \to C$ and $h : B \to C$ with domain B and common codomain, the equality $g \circ f = h \circ f$ implies that $g = h$.

25 Proposition In **Set**, the epimorphisms are just the surjective functions.

This correspondence provides a good mnemonic for remembering the definitions of monomorphisms and epimorphisms. But be careful: the correspondence does not hold in arbitrary categories. The "internal structure" of the objects in some categories can be used to construct epimorphisms, for example, that are not surjective when considered as functions on sets:

26 Example Both $(\mathbf{Z}, +, 0)$, the monoid of integers under addition, and $(\mathbf{Z}^+, +, 0)$, the monoid of nonnegative integers under addition, are objects of the category **Mon**. The inclusion function $i : (\mathbf{Z}^+, +, 0) \to (\mathbf{Z}, +, 0)$ that maps each nonnegative integer z to the integer z is a monomorphism, as we would expect by analogy with **Set**. But it is also an epimorphism. To see this, assume that $f \circ i = g \circ i$ for two homomorphisms f and g from

$(\mathbf{Z},+,0)$ to some monoid $(M,*,E)$. Take any $z \in \mathbf{Z}$. If $z \geq 0$, then it is the image under i of the same z considered as an element of $\mathbf{Z}^+$, so $f(z) = f(i(z)) = g(i(z)) = g(z)$. If $z < 0$, then $-z \geq 0$ and $-z \in \mathbf{Z}^+$. We reason as follows:

$$\begin{aligned}
f(i(-z)) &= g(i(-z)) \\
f(-z) &= g(-z) \\
f(-z) * f(z) &= g(-z) * f(z) \\
f(-z+z) &= g(-z) * f(z) \\
f(0) &= g(-z) * f(z) \\
E &= g(-z) * f(z) \\
g(z) &= g(z) * g(-z) * f(z) \\
g(z) &= g(0) * f(z) \\
g(z) &= f(z).
\end{aligned}$$

In the rich world of categories, the analogues of injective and surjective functions over sets are not sufficient to describe the full range of special kinds of arrows. These are the most common, but many textbooks define others—retractions, sections, constant arrows, coconstant arrows, zero arrows, bimorphisms, subobjects, quotient objects, and so on. We will limit ourselves here to mentioning just one more variety.

27 Definition An arrow $f : A \to B$ is an *isomorphism* (or just *iso*) if there is an arrow $g : B \to A$ such that $f \circ g = \mathrm{id}_B$ and $g \circ f = \mathrm{id}_A$. The objects A and B are said to be *isomorphic* if there is an isomorphism between them.

28 Example A group is essentially the same thing as a one-object category where every arrow is an isomorphism.

29 Exercises

1. Prove Proposition 25.
2. Show that in any category, if two arrows f and g are both monic, then their composition $g \circ f$ is monic. Also, if $g \circ f$ is monic, then so is f.
3. Dualize the previous exercise: state and prove the analogous proposition for epics. (Be careful of the second part.)
4. Show that the arrow g in Definition 27 is unique if it exists.

2.3 Initial and Terminal Objects

For the next several sections, we will be examining several sorts of "universal constructions" in categories. The simplest of these is the notion of "initial object" (and its dual):

30 Definition An object *0* is called an *initial object* if, for every object A, there is exactly one arrow from *0* to A.

31 Definition Dually, an object *1* is called a ***terminal object*** if, for every object A, there is exactly one arrow from A to *1*.

Arrows from an initial object or to a terminal object are often labeled "!" to highlight their uniqueness.

32 Example In **Set**, the empty set $\{\}$ is the unique initial object: for every set S, the empty function is the only function from $\{\}$ to S. Each one-element set is a terminal object in **Set**, since for every set S there is a function from S to a one-element set $\{x\}$ mapping every element of S to x, and furthermore this is the only total function from S to $\{x\}$.

33 Example In the category **Ω-Alg** of algebras with signature Ω, the initial object is the "initial algebra" (or "term algebra") whose carrier consists of all finite trees where each node is labeled with an operator from Ω and where each node labeled with o has exactly $ar(o)$ subtrees. (It is easy to see that this defines an Ω-algebra. The initiality of this algebra is a standard result of universal algebra [14].) The unique homomorphism from the term algebra to another Ω-algebra is often called a "semantic interpretation function."

34 Example Terminal objects can be used to provide a category-theoretic analogue of "elements" of objects. The motivating observation is that, in the category **Set**, the functions from a singleton set to a set S are in one-to-one correspondence with the elements of S. Moreover, if x is an element of S, considered as an arrow $x : 1 \to S$ from some one-element set *1*, and f is a function from S to some other set T, then the element $f(x)$ in T is exactly the one picked out by the composite function $f \circ x$.

In categorical terms, an arrow from a terminal object *1* to an object S is called a ***global element*** or ***constant*** of S.

35 Exercises

1. Show that two terminal objects in the same category must be isomorphic. Use duality to easily obtain a proof that any two initial objects are also isomorphic.
2. What are the terminal objects in **Set×Set**? In $\mathbf{Set}^{\to}$? In a poset considered as a category? What are the initial objects in these categories?
3. Name a category with no initial objects. Name one with no terminal objects. Name one where the initial and terminal objects are the same.

2.4 Products

The usual set-theoretic definition of the "cartesian product" of two sets A and B is

$$A \times B = \{\langle a, b\rangle \mid a \in A \text{ and } b \in B\}.$$

Using the observation of Example 34, we could perhaps define some sort of categorical product construction using global elements. But this would be contrary to the style of category theory, which tries to abstract away from elements, treating objects as black boxes with unexamined internal structure and focusing attention of the properties of arrows *between* objects. What we need is an "arrow-theoretic" characterisation of products. (Note that here we are considering products *within* a category rather than products *of* categories as in Example 17.)

What arrows are particularly relevant to products? Well, in order for an element of the product to be useful there must be some way of breaking it apart to inspect the values of its first and second components—that is, whenever we form a product of two sets A and B we also specify "projection functions" $\pi_1 : A \times B \to A$ and $\pi_2 : A \times B \to B$. The projection functions π_1 and π_2 capture the "productness" of $A \times B$ in the following sense. Assume that for some other set C, there are two functions $f : C \to A$ and $g : C \to B$. Then we can form the "product function" $(f, g) : C \to A \times B$, defined by:

$$(f,g)(x) = \langle f(x), g(x) \rangle.$$

This function is unique, in the sense that it is uniquely determined by f and g.

This motivates the general definition of categorical products:

36 Definition A *product* of two objects A and B is an object $A \times B$, together with two projection arrows $\pi_1 : A \times B \to A$ and $\pi_2 : A \times B \to B$, such that for any pair of arrows $f : C \to A$ and $g : C \to B$ there is exactly one arrow $(f, g) : C \to A \times B$ making the diagram

$$\begin{tikzcd} & C \arrow[dl, "f"'] \arrow[d, dashed, "{(f,g)}"] \arrow[dr, "g"] & \\ A & A \times B \arrow[l, "\pi_1"] \arrow[r, "\pi_2"'] & B \end{tikzcd}$$

commute—that is, such that $\pi_1 \circ (f,g) = f$ and $\pi_2 \circ (f,g) = g$.

(Dashed arrows in commutative diagrams are used to represent arrows that are *asserted to exist uniquely* when the rest of the diagram is filled in appropriately.)

Although it is customary to refer to just the object $A \times B$ as a product object, it is important to remember that the projection arrows are also part of the definition. Formally, we might define the product as the tuple $\langle A \times B, \pi_1, \pi_2 \rangle$. This will become clearer in Section 2.7, where products are shown to be an instance of a more general construction called limits.

We can now define arrows *between* product objects in terms of projection arrows:

37 Definition If the product objects $A \times C$ and $B \times D$ exist, then for every pair of arrows $f : A \to B$ and $g : C \to D$, the *product map* $f \times g : A \times C \to B \times D$ is the arrow $\langle f \circ \pi_1, g \circ \pi_2 \rangle$.

As with set-theoretic cartesian products, the categorical binary product construction can be generalized to arbitrary "indexed products":

38 Definition† A product of a family $(A_i)_{i\in I}$ of objects indexed by a set I consists of an object $\prod_{i\in I} A_i$ and a family of projection arrows $(\pi_i : (\prod_{i\in I} A_i) \to A_i)_{i\in I}$ such that for each object C and family of arrows $(f_i : C \to A_i)_{i\in I}$ there is a unique arrow $(f_i)_{i\in I} : C \to (\prod_{i\in I} A_i)$ such that the following diagram commutes for all $i \in I$:

$$\begin{tikzcd}
C \arrow[d, dashed, "(f_i)_{i\in I}"'] \arrow[dr, "f_i"] & \\
\prod\limits_{i\in I} A_i \arrow[r, "\pi_i"'] & A_i
\end{tikzcd}$$

39 Exercises

1. Note that Definition 36 says "*a* product..." rather than "*the* product...." Products (like all universal constructions) are defined only up to an isomorphism. Show that this is the case—i.e., that if an object X with arrows $\pi_A : X \to A$ and $\pi_B : X \to B$ also satisfies the definition of "X is a product of A and B," then X is isomorphic to $A \times B$. Conversely, show that any object isomorphic to a product object $A \times B$ is also a product of A and B.
2. Show that $(f \circ h, g \circ h) = (f, g) \circ h$. (Hint: begin by drawing a diagram.)
3. Show that $(f \times h) \circ (g, k) = (f \circ g, h \circ k)$.
4. The dual construction, ***coproduct***, corresponds to set-theoretic disjoint union. Dualize definitions 36 and 38 to give the details of the construction.
5. To what does Definition 38 reduce when the index set I is empty?

2.5 Equalizers

Another useful categorical construct is the "equalizer" of two arrows.

40 Definition An arrow $e : X \to A$ is an ***equalizer*** of a pair of parallel arrows $f : A \to B$ and $g : A \to B$ if

1. $f \circ e = g \circ e$;
2. whenever $e' : X' \to A$ also satisfies $f \circ e' = g \circ e'$, there is a unique arrow $k : X' \to X$ such that $e \circ k = e'$:

$$\begin{tikzcd}
X \arrow[r, "e"] & A \arrow[r, shift left, "f"] \arrow[r, shift right, "g"'] & B \\
X' \arrow[u, dashed, "k"] \arrow[ur, "e'"'] & &
\end{tikzcd}$$

41 Example The category **Set** again provides the most intuitive illustration. Let f and g be two functions in **Set** with common domain A and codomain B, and let X be the subset of A on which f and g are equal, that is:

$$X = \{x \mid x \in A \text{ and } f(x) = g(x)\}.$$

Then the inclusion function $e : X \to A$, which maps each element $x \in X$ to the same x as an element of A, is an equalizer of f and g.

The dual construction, *coequalizer*, provides a categorical analogue to the set-theoretic notion of an equivalence relation. (See Arbib and Manes [1, p. 19], Goldblatt[31, p. 61], or any standard textbook.)

42 Exercises

1. Show that in a poset considered as a category, the only equalizers are the identity arrows.
2. Show that every equalizer is monic.
3. Show that the converse is not true in all categories—that there is a category with a monic arrow that does not equalize any pair of arrows. (Hint: think about the natural numbers considered as a poset.)
4. Show that every epic equalizer is an isomorphism.

2.6 Pullbacks

The pullback is a very common and important construction.

43 Definition A *pullback* of the pair of arrows $f : A \to C$ and $g : B \to C$ is an object P and a pair of arrows $g' : P \to A$ and $f' : P \to B$ such that

1. $f \circ g' = f' \circ g$:

$$\begin{tikzcd} P \arrow[r, "f'"] \arrow[d, "g'"'] & B \arrow[d, "g"] \\ A \arrow[r, "f"'] & C \end{tikzcd}$$

2. if $i : X \to A$ and $j : X \to B$ are such that $f \circ i = g \circ j$, then there is a unique $k : X \to P$ such that $i = g' \circ k$ and $j = f' \circ k$:

$$\begin{tikzcd} X \arrow[dr, dotted, "k"'] \arrow[drr, bend left, "j"] \arrow[ddr, bend right, "i"'] & & \\ & P \arrow[r, "f'"'] \arrow[d, "g'"] & B \arrow[d, "g"] \\ & A \arrow[r, "f"'] & C \end{tikzcd}$$

This situation is commonly described by saying that f' is the ***pullback*** (or ***inverse image***) ***of*** f ***along*** g and that g' is the pullback of g along f. The example that motivates this terminology comes, as usual, from **Set**:

44 Example Let $f : B \to C$ be a function in **Set** and $A \subseteq C$. Write $f^{-1}(A)$ for the inverse image of A under f (the set $\{b \mid f(b) \in A\}$), and $f|_S$ for the restriction of f to S ($S \subseteq B$). Then the following diagram is a pullback (that is, it depicts a situation where the top and left sides of the square are the pullbacks of the bottom and right sides):

$$\begin{tikzcd} f^{-1}(A) \arrow[r, "i"] \arrow[d, "f|_{f^{-1}(A)}"'] & B \arrow[d, "f"] \\ A \arrow[r, "i"'] & C \end{tikzcd}$$

(To avoid a proliferation of names, i is used both as the name of the inclusion function from A to C and of the one from $f^{-1}(A)$ to B.)

Saying that "the diagram is a pullback" is actually more rigorous than it sounds, as we shall see in a few pages.

45 Example If A and B are subsets of the set C, then

$$\begin{tikzcd} A \cap B \arrow[r, "i"] \arrow[d, "i"'] & B \arrow[d, "i"] \\ A \arrow[r, "i"'] & C \end{tikzcd}$$

is a pullback.

46 Example More generally, let f and g be two Set-functions with common codomain C. The pullback object P (it is unique in this case) is defined by:

$$P = \{\langle a, b\rangle \mid a \in A,\ b \in B, \text{ and } f(a) = g(b)\}.$$

(P is a subset of the cartesian product $A \times B$.) The projections f' and g' are defined by

$$\begin{aligned} f'(\langle a, b\rangle) &= b, \\ g'(\langle a, b\rangle) &= a. \end{aligned}$$

Pullbacks are sometimes known as "fibered products."

The next two examples begin to demonstrate the interdefinability of many category constructs:

47 Example In any category with a terminal object, if

$$\begin{tikzcd} P \arrow[r, "f"] \arrow[d, "g"'] & B \arrow[d, "!"] \\ A \arrow[r, "!"'] & 1 \end{tikzcd}$$

is a pullback, then P is a product of A and B (f and g are projections).

48 Example In any category, if

$$\begin{tikzcd} X \arrow[r, "e"] \arrow[d, "e"'] & A \arrow[d, "g"] \\ A \arrow[r, "f"'] & B \end{tikzcd}$$

is a pullback, then e is an equalizer of f and g.

49 Exercises

1. (The "Pullback Lemma.") Consider the following diagram:

$$\begin{tikzcd}
\cdot \arrow[r] \arrow[d] & \cdot \arrow[r] \arrow[d] & \cdot \arrow[d] \\
\cdot \arrow[r] & \cdot \arrow[r] & \cdot
\end{tikzcd}$$

   (a) Prove that if both of the squares are pullbacks, then the outside rectangle (with top and bottom edges the evident composites) is a pullback.

   (b) Prove that if the outside rectangle and the right-hand square are pullbacks, then the left-hand square is a pullback.

2. Show how to construct pullbacks from products and equalizers. That is, show that in any category where every two objects have a product and every two arrows have an equalizer, it is also the case that every two arrows with the same codomain have a pullback. (Hint: see Example 46.)

3. State the dual notion, ***pushout*** and check that in **Set** the pushout of $f : A \to B$ and $g : A \to C$ is obtained by forming the disjoint union of A and B, and then identifying $f(x)$ with $g(x)$ for each $x \in A$ by a coequalizer.

2.7 Limits

The reader may already have discerned a pattern in the definitions of products, equalizers, and pullbacks. In each case, we describe a property and define a certain object (with some accompanying arrows) to have the property "canonically," in the sense that any other object with the property "factors uniquely through" the canonical one. (For example, if e is an equalizer of f and G, then any arrow e' such that $f \circ e' = g \circ e'$ can be expressed as the composite arrow of e with some unique arrow k, that is, $e' = e \circ k$.) Such definitions are called ***universal constructions***; the entities they define are said to be ***universal*** among entities satisfying the given property, or simply to have the ***universal property***.

With this in mind, we proceed to the general definition of the "limit of a diagram."

50 Definition Let **C** be a category and **D** a diagram in **C**. A *cone* for **D** is a **C**-object X and arrows $f_i : X \to D_i$ (one for each object D_i in **D**), such that for each arrow g in **D**, the diagram

$$\begin{tikzcd}
& X \arrow[dl, "f_i"'] \arrow[dr, "f_j"] & \\
D_i \arrow[rr, "g"'] & & D_j
\end{tikzcd}$$

commutes. We use the notation $\{f_i : X \to D_i\}$ for cones.

51 Definition A *limit* for a diagram $\mathbf{D}$ is a cone $\{f_i : X \to D_i\}$ with the property that if $\{f_i' : X' \to D_i\}$ is another cone for $\mathbf{D}$, then there is a unique arrow $k : X' \to X$ such that the diagram

$$\begin{tikzcd} X' \arrow[rr, dashed, "k"] \arrow[dr, "f_i'"'] & & X \arrow[dl, "f_i"] \\ & D_i & \end{tikzcd}$$

commutes for every D_i in $\mathbf{D}$.

The cones for a diagram $\mathbf{D}$ actually form a category; a limit is a terminal object in this category. It follows that since terminal objects are unique up to isomorphism, so are limits.

52 Example Given two $\mathbf{C}$-objects A and B, let $\mathbf{D}$ be the diagram

$$\begin{tikzcd} A & & B \end{tikzcd}$$

with two nodes labeled A and B and no edges. Then a cone for this diagram is just an object X with two arrows f and g of the form

$$\begin{tikzcd} A & X \arrow[l, "f"'] \arrow[r, "g"] & B \end{tikzcd}$$

A limiting $\mathbf{D}$-cone, if it exists, is one through which all such cones factor. But this is just a product of A and B.

53 Example Let $\mathbf{D}$ be the empty diagram with no nodes and no edges. A cone for $\mathbf{D}$ in a category $\mathbf{C}$ is any $\mathbf{C}$-object. ($\mathbf{D}$ has no nodes so the cone has no arrows.) A limiting cone is then an object C with the additional requirement that for any $\mathbf{C}$-object C' (i.e. for any $\mathbf{D}$-cone) there is exactly one arrow from C' to C. In other words, C is a terminal object.

54 Example Let $\mathbf{D}$ be the diagram

$$\begin{tikzcd} & B \arrow[d, "g"] \\ A \arrow[r, "f"'] & C \end{tikzcd}$$

with three nodes and two edges. Strictly speaking, a cone for $\mathbf{D}$ is an object P and three arrows f', g', and h:

$$\begin{tikzcd} P \arrow[r, "f'"] \arrow[d, "g'"'] \arrow[dr, "h"] & B \arrow[d, "g"] \\ A \arrow[r, "f"'] & C \end{tikzcd}$$

But this is equivalent to

$$\begin{tikzcd}
P \arrow[r, "f'"] \arrow[d, "g'"'] & B \arrow[d, "g"] \\
A \arrow[r, "f"'] & C
\end{tikzcd}$$

because h is completely determined as the common composite $f \circ g' = h = g \circ f'$.

Moreover, since P, f', and g' form a limit, they have the universal property among objects and arrows that make this diagram commute—that is, given any object P' with arrows f'', g'', and h' making the analogous diagram commute, there will be a unique arrow k from P' to P such that f'', g'', and h factor uniquely through k. Ignoring h' as before, this shows that the limit of $\mathbf{D}$ is the pullback of f and g.

55 Definition Dually, a *cocone* for a diagram $\mathbf{D}$ in a category $\mathbf{C}$ is a $\mathbf{C}$-object X and a collection of arrows $f_i : D_i \to X$. A *colimit* or *inverse limit* for $\mathbf{D}$ is then a cocone $\{f_i : D_i \to X\}$ with the "couniversal property" that for any other cocone $\{f_i' : D_i \to X'\}$ there is a unique arrow $k : X \to X'$ such that the diagram

$$\begin{tikzcd}
& D_i \arrow[dl, "f_i"'] \arrow[dr, "f_i'"] & \\
X \arrow[rr, dashed, "k"'] & & X'
\end{tikzcd}$$

commutes for every object D_i in $\mathbf{D}$.

56 Exercises

1. Let $\mathbf{D}$ be the diagram

$$\begin{tikzcd}
A \arrow[r, shift left, "f"] \arrow[r, shift right, "g"'] & B
\end{tikzcd}$$

   Show that a limit for $\mathbf{D}$ is an equalizer of f and g.

2. In a poset $(P, \leq)$ considered as a category (Example 13), all diagrams commute. To what do the limit and colimit of a diagram correspond?

2.8 The Limit Theorem†

Of course, not all diagrams have limits. For example, in a category with no terminal object, the empty diagram has no limit. In a category where the only arrows are identities—a "discrete category"—*no* diagrams have limits. In fact, knowing just which diagrams *do* have limits in a given category tells a lot about what that category is like.

One pleasant possibility is that *all* limits exist in the category in question—that given any diagram $\mathbf{D}$, there are always some object and arrows forming a cone for $\mathbf{D}$ and universal among such cones. In fact, there is a general theorem showing that whenever limits exist for very simple diagrams—products and equalizers—they must exist for arbitrary diagrams. (To be more precise, the theorem does not give the existence of all limits, but only limits of "small" diagrams—those whose sets of nodes and edges are really sets and not proper classes. See Remark 66 for more details.)

57 Theorem Let **D** be a diagram in a category **C**, with sets V of vertices and E of edges. If every V-indexed and every E-indexed family of objects in **C** has a product, and if every pair of arrows in **C** has an equalizer, then **D** has a limit.

Proof Sketch: A good candidate for the limit object would be the product $\prod_{I \in V} D_I$, since it comes equipped with an arrow to each D_I. But this won't quite work because the arrows from the product don't necessarily form commuting triangles with the edges D_e. If we form the product $\prod_{I \xrightarrow{e} J \in E} D_J$ of the *targets* of the edges of **D**, then for each edge $D_e : D_I \to D_J$, there are two ways to get from the product of all vertices to the product of the targets of edges: directly (by π_J), or via D_e (by $D_e \circ \pi_I$). Thus we can form two families of edges from $\prod_{I \in V} D_I$ to the D_J's. Since $\prod_{I \xrightarrow{e} J \in E} D_J$ is a product object, each family induces a mediating arrow from $\prod_{I \in V} D_I$ to $\prod_{I \xrightarrow{e} J \in E} D_J$. The restriction of $\prod_{I \in V} D_I$ that we are interested in is the one for which each arrow directly to D_J is equal to the one via D_e. This is the equalizer of the two mediating arrows. (*End of Proof*)

The full proof appears in Appendix A. Readers are encouraged to examine it, both for its own interest and as a nice example of categorical reasoning.

2.9 Exponentiation

Our final example of a universal construction is also the first that explicitly relates to computation. Basically, we give a categorical interpretation of the notion of "currying" whereby a two-argument function is reduced to a one-argument function by fixing the value of the other argument.

We begin by sketching the construction in **Set**, and then give the categorical definition. (The presentation is adapted from Goldblatt's.)

Given two sets A and B, we can form the set B^A of all functions with domain A and codomain B—that is,

$$B^A = \{f : A \to B\}.$$

(Part of what makes this observation interesting is that we cannot do this in every category. In **Set**, it happens to be the case that B^A is itself a set. In some other categories **C**, it is similarly the case that $\mathbf{C}(A, B)$ is represented by an object of **C**. It is this representing object that we want to identify in the general case.)

As usual, we want to characterize B^A by arrows instead of elements. To do this, we observe that associated with B^A is a special *evaluation* function $eval : B^A \times A \to B$, defined by the rule $eval(\langle f, a \rangle) = f(a)$. (On input $\langle f, a \rangle$, with $f : A \to B$ and $a \in A$ it yields as output $f(a) \in B$.)

The categorical description of B^A hinges on the observation that *eval* posseses a universal property among all functions of the form $(C \times A) \xrightarrow{g} B$. Given any such g, there is exactly one function $g^* : C \to B^A$ such that the following diagram commutes:

$$\begin{tikzcd}
B^A \times A \arrow[dr, "eval"] & \\
& B \\
C \times A \arrow[uu, dashed, "g^* \times \mathrm{id}_A"] \arrow[ur, "g"'] &
\end{tikzcd}$$

(Recall that $g^* \times \mathrm{id}_A$ denotes a product mapping (Definition 37). On input $\langle c, a\rangle$ it yields $\langle g^*(c), a\rangle$.)
The idea here is that any particular $c \in C$ determines a function in B^A if the first argument at c, leaving the second argument free. Define g_c to be the curried version of g for a particular $c \in C$:

$$g_c(a) = g(\langle c, a\rangle).$$

Then g^* is the function that takes each c to the appropriate curried version of g:

$$g^*(c) = g_c.$$

Now, for any $\langle c, a\rangle \in C \times A$, we have:

$$\begin{aligned}
(eval \circ (g^* \times \mathrm{id}_A))(\langle c, a\rangle) &= eval(\langle g^*(c), a\rangle) \\
&= eval(\langle g_c, a\rangle) \\
&= g_c(a) \\
&= g(\langle c, a\rangle).
\end{aligned}$$

This shows that the function g^* makes the diagram commute. To see that it is the only function that makes the diagram commute, note that $eval(\langle g^*(c), a\rangle) = g(\langle c, a\rangle)$ implies that $(g^*(c))(a) = g(\langle c, a\rangle)$, that is, that $g^*(c)$ must be a function that, for input a, yields $g(\langle c, a\rangle)$. This function is g_c.

The general categorical definition is as follows:

58 Definition A category has *exponentiation* if it has a product for any two objects, and if for any objects A and B there is an object B^A and an evaluation arrow $eval_{AB} : (B^A \times A) \to B$ such that for any object C and arrow $g : (C \times A) \to B$ there is a unique arrow $g^* : C \to B^A$ making

$$\begin{tikzcd}
B^A \times A \arrow[dr, "eval_{AB}"] & \\
& B \\
C \times A \arrow[uu, dashed, "g^* \times \mathrm{id}_A"] \arrow[ur, "g"'] &
\end{tikzcd}$$

commute—that is, a unique g^* such that $eval_{AB} \circ (g^* \times \mathrm{id}_A) = g$.

59 Exercises

1. Exponentiation in **Set** $\times$ **Set** is "componentwise" the same as in **Set**. Check the details of the construction.
2. Give the construction of exponentiation in $\mathbf{Set}^{\rightarrow}$. (Difficult. See Goldblatt [31, p. 88].)

2.10 Functors

It should be observed that the whole concept of a category is essentially an auxiliary one; our basic concepts are essentially those of of a functor and a natural transformation...

— Eilenberg and Mac Lane [18]

In Section 2.1 we observed that many kinds of mathematical domains can be thought of as categories. Since categories themselves consititute a mathematical domain, it makes sense to ask whether there is a category of categories. In fact, there is: its objects are categories and its arrows are certain structure-preserving maps between categories, called functors.

60 Definition Let **C** and **D** be categories. A *functor* $F : \mathbf{C} \rightarrow \mathbf{D}$ is a map taking each **C**-object A to a **D**-object $F(A)$ and each **C**-arrow $f : A \rightarrow B$ to a **D**-arrow $F(f) : F(A) \rightarrow F(B)$, such that for all **C**-objects A and composable **C**-arrows f and g,

1. $F(\mathrm{id}_A) = \mathrm{id}_{F(A)}$;
2. $F(g \circ f) = F(g) \circ F(f)$.

Type constructors provide a familiar example from computer science:

61 Example (Adapted from Rydeheard [77].) Given a set S, we can form the set $\mathit{List}(S)$ of finite lists with elements drawn from S. This definition can be thought of as a mapping $\mathit{List} : \mathbf{Set} \rightarrow \mathbf{Set}$, the object part of a functor. The arrow part takes a function $f : S \rightarrow S'$ to a function $\mathit{List}(f) : \mathit{List}(S) \rightarrow \mathit{List}(S')$ that, given a list $L = [s_1, s_2, \ldots, s_n]$, "maps" f over the elements of L:

$$\mathit{List}(f)(L) = \mathit{maplist}(f)(L) = [f(s_1), f(s_2), \ldots, f(s_n)].$$

Actually, the set of lists with elements from a set S has some additional structure that we have not yet taken into account. There is an associative binary concatenation operation $*$ over $\mathit{List}(S)$, and an empty list $[]$ that acts as an identity for $*$ (i.e. $[] * L = L = L * []$). Thus, $(\mathit{List}(S), *, [])$ is a monoid (Example 4), and $\mathit{List} : \mathbf{Set} \rightarrow \mathbf{Mon}$ is a functor taking each set S to the monoid of lists with elements drawn from S.

The arrow part of List takes each set function f to a monoid homomorphism $\mathit{List}(f) = \mathit{maplist}(f)$. Indeed, the fact that $\mathit{maplist}(f)$ is a homomorphism corresponds exactly to the first two lines in the usual definition of *maplist*:

$$\begin{array}{lcl} maplist(f)([]) & = & [] \\ maplist(f)(L * L') & = & maplist(f)(L) * maplist(f)(L') \\ maplist(f)([s]) & = & [f(s)]. \end{array}$$

The third line, as we shall see in Section 2.12, ensures that these equations *define* the homomorphism *maplist*, in the sense that it satisfies them uniquely.

One simple but very important class of functors is the *forgetful functors*, which operate by forgetting part of the structure of structured objects. The letter U is often used to denote a forgetful functor because one thinks of it as extracting an "underlying" structure (often just a set).

62 Example The forgetful functor $U : \mathbf{Mon} \rightarrow \mathbf{Set}$ sends each monoid $(M, \cdot, e)$ to the set M and each monoid homomorphism $h : (M, \cdot, e) \rightarrow (M', \cdot', e')$ to the corresponding function $h : M \rightarrow M'$ on the underlying sets.

Another simple functor is the *identity functor* on a category:

63 Example For each category $\mathbf{C}$, the identity functor $I_{\mathbf{C}}$ takes C-object and every C-arrow to itself.

Another innocent-looking class of functors will prove quite useful later on:

64 Example Let $\mathbf{C}$ be a category with a product $X \times Y$ for each pair X and Y of objects. Then each C-object A determines a functor $— \times A : \mathbf{C} \rightarrow \mathbf{C}$ taking each object B to $B \times A$ and each arrow $f : B \rightarrow C$ to $f \times \mathrm{id}_A$. (The "—" is used to show where the argument object or arrow goes.)

The composition of two functors is defined by separately composing their effects on objects and on arrows. Given functors $F : \mathbf{A} \rightarrow \mathbf{B}$ and $G : \mathbf{B} \rightarrow \mathbf{C}$, the composite functor $G \circ F$ maps each A-object A to a C-object $G(F(A))$ and each A-arrow $f : A \rightarrow A'$ to a C-arrow $G(F(f)) : G(F(A)) \rightarrow G(F(A'))$. Is is easy to check that this composition operation is associative, and that the identity functors defined above are identities for composition of functors.

With this observation, we are ready to define the category of all categories:

65 Example The category **Cat** has categories as objects and functors as arrows.

66 Remark At this point, we should mention an important technical concern. The "size" of **Cat** is clearly enormous, so that we are led to wonder whether it can possibly be considered as one of its own objects. (Readers familiar with Russell's Paradox may be concerned that there is a similar problem lurking in this definition. There is.) To avoid such questions, category theorists generally distinguish between "large" and "small" categories. Small categories are those whose collections of objects and arrows are both sets. Then **Cat** is defined more precisely to be the category of all small categories (which is itself a large category).

Readers interested in foundational issues should refer to a standard text on category theory such as Mac Lane [54] or Herrlich and Strecker [38]. There are also papers by Mac Lane [55], Feferman [22], Grothendieck [32], Blass [8], Lawvere [48], and Bénabou [6], and a chapter in Hatcher's book [37].

Our final class of functors "internalizes" the notion of sets of arrows between objects:

67 Example Given a category **C**, each **C**-object A determines a functor $\mathbf{C}(A, —) : \mathbf{C} \to \mathbf{Set}$. This functor takes each **C**-object B to the set $\mathbf{C}(A, B)$ of arrows from A to B, and each **C**-arrow $f : B \to C$ to the function $\mathbf{C}(A, f) : \mathbf{C}(A, B) \to \mathbf{C}(A, C)$ that yields $f \circ g$ for input g:

$$\begin{tikzcd}
A \arrow[r, "g"] \arrow[dr, "f \circ g"'] & B \arrow[d, "f"] \\
 & C
\end{tikzcd}$$

Using the "— notation" we might write this as $\mathbf{C}(A, f) = (f \circ —)$.

$\mathbf{C}(A, —)$ is called a ***hom-functor***. (The origin of the term is in the frequent use of arrows to model homomorphisms of various sorts.) The set $\mathbf{C}(A, B)$ is often called a "hom set."

Again, concern for consistency motivates a restriction on this definition: the collections of arrows between each two **C**-objects must be actual sets—not proper classes—for the definition to make sense.

The functors we have considered so far have all been *covariant*. A *contravariant* functor is one that maps objects to objects as before, but maps arrows to arrows going the opposite direction. This is not really a new concept, however, since a contravariant functor $F : \mathbf{C} \to \mathbf{D}$ is exactly the same as a covariant functor $F : \mathbf{C}^{\mathrm{op}} \to \mathbf{D}$. Similarly, product categories can be used to define n-ary functors.

Using functors from opposite and product categories, it is possible to give a contravariant version—and even a two-argument version (contravariant in the first argument and covariant in the second)—of the hom-functor construction.

68 Exercises

1. Check carefully that the constructions in examples 62 through 67 define functors.
2. The powerset operator $\mathcal{P}$ takes each set S to the set $\mathcal{P}(S) = \{T \mid T \subseteq S\}$ (the set of all subsets of S). Show that $\mathcal{P}$ can be extended to a functor $\mathcal{P} : \mathbf{Set} \rightarrow \mathbf{Set}$.
3. Let $\mathbf{M}$ and $\mathbf{N}$ be two monoids considered as one-object categories. What are the functors from $\mathbf{M}$ to $\mathbf{N}$?
4. By analogy with Example 67, define the "contravariant hom-functor" $\mathbf{C}(—, B)$ and the "bifunctor" $\mathbf{C}(—,—)$.

2.11 Natural Transformations

We are now reaching the realm where category theory begins to show its power—and also its confusing habit of proceeding by continually adding new layers of abstraction. Having defined mappings from one category to another—functors—we now proceed to define "structure-preserving mappings," called natural transformations between functors! The concept of "naturality" is central in many mathematical applications of category theory; indeed, category theory itself was originally developed in order to deal systematically with natural transformations.

What is a structure-preserving map between functors? Given two functors $F : \mathbf{C} \rightarrow \mathbf{D}$ and $G : \mathbf{C} \rightarrow \mathbf{D}$, we can think of each of them as "projecting a picture of $\mathbf{C}$ inside of $\mathbf{D}$." Natural transformations arise when we try to imagine "sliding" the picture of F onto the picture of G in such a way that the "$\mathbf{C}$-ness" of the picture is preserved. For each $\mathbf{C}$-object A, we define an arrow τ_A from the F-image of A to its G-image. To ensure that the structure of $\mathbf{C}$ is preserved, we require that for each $\mathbf{C}$-arrow $f : A \rightarrow B$, the transformations τ_A and τ_B take the endpoints of the F-image of f to the endpoints of the G-image of f.

The formal definition is as follows:

69 Definition Let $\mathbf{C}$ and $\mathbf{D}$ be categories, and F and G functors from $\mathbf{C}$ to $\mathbf{D}$. A ***natural transformation*** τ from F to G (written $\tau : F \stackrel{\cdot}{\rightarrow} G$) is a function that assigns to every $\mathbf{C}$-object A a $\mathbf{D}$-arrow $\tau_A : F(A) \rightarrow G(A)$ such that for any $\mathbf{C}$-arrow $f : A \rightarrow B$ the diagram on the right commutes (in $\mathbf{D}$):

$$\begin{tikzcd} A \arrow[d, "f"'] & F(A) \arrow[r, "\tau_A"] \arrow[d, "F(f)"'] & G(A) \arrow[d, "G(f)"] \\ B & F(B) \arrow[r, "\tau_B"] & G(B) \end{tikzcd}$$

If each component of τ is an isomorphism in $\mathbf{D}$, then τ is called a ***natural isomorphism***.

70 Example For any functor F, the identity natural transformation $\iota_F : F \stackrel{\cdot}{\rightarrow} F$ takes each object A to the identity arrow $\mathrm{id}_{F(A)}$. (In fact, ι_F is a natural isomorphism.)

In categories with exponentiation (Definition 58) it turns out that the evaluation mapping forms a natural transformation. For simplicity, we give the construction in the category **Set**.

71 Example For a fixed set A, the map taking B to $B^A \times A$ can be extended to a functor $F_A : \mathbf{Set} \to \mathbf{Set}$ as follows:

$$\begin{aligned} F_A(B) &= B^A \times A, \\ F_A(f) &= (f \circ —) \times \mathrm{id}_A. \end{aligned}$$

Alternatively:

$$F_A = (— \times A) \circ (—)^A,$$

where $(—)^A$ takes each B to B^A and each $f : C \to B$ to $(f \circ —) : C^A \to B^A$ and $(— \times A)$ is the "right product" functor of Example 64.

The fact that $\mathit{eval} : F_A \dot{\to} I_{\mathbf{Set}}$ is a natural transformation follows from the commutativity of the diagram

$$\begin{tikzcd} C^A \times A \arrow[r, "\mathit{eval}_{AC}"] \arrow[d, "F_A(g)"'] & C \arrow[d, "g"] \\ B^A \times A \arrow[r, "\mathit{eval}_{AB}"'] & B \end{tikzcd}$$

which in turn follows by checking that for any $f : A \to C$ and $a \in A$,

$$\begin{aligned} (g \circ \mathit{eval}_{AC})(\langle f, a \rangle) &= g(\mathit{eval}_{AC}(\langle f, a \rangle)) \\ &= g(f(a)) \\ &= (g \circ f)(a) \\ &= \mathit{eval}_{AB}(\langle g \circ f, a \rangle) \\ &= \mathit{eval}_{AB}(F_A(g)(a)) \\ &= (\mathit{eval}_{AB} \circ F_A(g))(\langle f, a \rangle). \end{aligned}$$

Natural transformations are ubiquitous in category theory. One of their uses is in defining categories of functors.

72 Example Let **C** and **D** be categories. Let F, G, and H be functors from **C** to **D**. Let $\sigma : F \dot{\to} G$ and $\tau : G \dot{\to} H$ be natural transformations. Then for each **C**-arrow $f : A \to B$ we can draw the following composite diagram:

$$\begin{tikzcd} F(A) \arrow[r, "\sigma_A"] \arrow[d, "F(f)"'] & G(A) \arrow[r, "\tau_A"] \arrow[d, "G(f)"'] & H(A) \arrow[d, "H(f)"'] \\ F(B) \arrow[r, "\sigma_B"] & G(B) \arrow[r, "\tau_B"] & H(B) \end{tikzcd}$$

Since both squares commute, so does the outer rectangle. This shows that the composite transform $(\tau \circ \sigma) : F \dot{\rightarrow} H$, defined by $(\tau \circ \sigma)_A = \tau_A \circ \sigma_A$, is natural.

Composition of natural transforms is associative and has, for each functor F, the identity ι_F. Thus, for every two categories $\mathbf{C}$ and $\mathbf{D}$, we can form a *functor category* $\mathbf{D}^{\mathbf{C}}$, whose objects are functors from $\mathbf{C}$ to $\mathbf{D}$ and whose arrows are natural transformations between such functors.

73 Example A category with no arrows other than identity arrows is essentially a set. If $\mathbf{C}$ and $\mathbf{D}$ are both sets, then $\mathbf{D}^{\mathbf{C}}$ is also a set, namely the set of total functions from $\mathbf{C}$ to $\mathbf{D}$.

74 Example For any category $\mathbf{C}$, $\mathbf{C}^{\mathbf{1}}$ is isomorphic to $\mathbf{C}$ itself. ($\mathbf{1}$ is the one-point category defined in Example 7.)

75 Example Recall (Example 8) that $\mathbf{2}$ is a category with two objects and one non-identity arrow from one object to the other. For any category $\mathbf{C}$, the functor category $\mathbf{C}^{\mathbf{2}}$ is exactly the arrow category $\mathbf{C}^{\rightarrow}$ (Example 18). Its objects are the arrows $f : A \rightarrow B$ of $\mathbf{C}$. (Technically, they are functors from $\mathbf{2}$ to $\mathbf{C}$, but each such functor picks out just one arrow in $\mathbf{C}$, and this choice completely determines the object part of the functor.) Its arrows are pairs $\langle h, k\rangle$ of $\mathbf{C}$-arrows (why?) for which the following diagram commutes:

$$\begin{tikzcd}
A \arrow[r, "h"] \arrow[d, "f"'] & A' \arrow[d, "f'"] \\
B \arrow[r, "k"'] & B'
\end{tikzcd}$$

76 Exercises

1. Show that every category $\mathbf{C}$ is naturally isomorphic to the category $\mathbf{C} \times \mathbf{1}$.
2. Recall the *List* functor of Example 61. Let *rev* be the "reverse" operation on lists, that is, $rev(S) : List(S) \rightarrow List(S)$ takes each list with elements in S to its reverse. Show that *rev* is a natural transformation.
3. Let $\mathbf{P}$ be a preorder regarded as a category and $\mathbf{C}$ be an arbitrary category. Let $S, T : \mathbf{C} \rightarrow \mathbf{P}$ be functors. Show that there is a natural transformation $\tau : S \dot{\rightarrow} T$ (it will be unique) if and only if $S(C) \leq T(C)$ for every $\mathbf{C}$-object C.

2.12 Adjoints

The slogan is, "Adjoint functors arise everywhere".

— Mac Lane [54]

Adjoints, developed by Kan in 1958, are considered one of the most important ideas in category theory, and also perhaps the most significant contribution of category theory to the broader arena of mathematical thinking. A great variety of mathematical constructions—including many parts of category theory itself—are examples of adjoints.

The concept is quite a bit more intricate than anything we have encountered so far. The best way to grasp it is by working through the details of as many examples as possible. We begin with one important example, the "free monoid," then give a formal definition of adjunction, and then proceed to further examples.

This section is inspired by Rydeheard's excellent short article [76].

We begin by showing that the monoid $(List(S), *, [])$ of Example 61, also known as the ***free monoid generated by the set*** S, has a very special property among monoids:

77 Proposition If $f : S \to M$ is any function from the set S to the underlying set M of a monoid $(M, \cdot, e)$, then there is exactly one monoid homomorphism $f^{\#} : (List(S), *, []) \to (M, \cdot, e)$ such that the following diagram commutes (where i is the injection taking an element $s \in S$ to the list $[s]$ of length 1):

$$\begin{tikzcd}
S \arrow[r, "i"] \arrow[dr, "f"'] & List(S) \arrow[d, "f^{\#}"] \\
& M
\end{tikzcd}$$

Proof: Define $f^{\#}$ to be the monoid homomorphism taking each list $[s_1, s_2, \ldots, s_n]$ to the product $f(s_1) \cdot f(s_2) \cdot \cdots \cdot f(s_n)$ in $(M, \cdot, e)$ and taking the empty list $[]$ to e. This definition clearly satisfies the conditions for being a monoid (1 and 2) and makes the diagram commute (3):

1. $f^{\#}([]) = e$,
2. $f^{\#}([s_1, s_2, \ldots, s_n] * [t_1, t_2, \ldots, t_m]) = f^{\#}([s_1, s_2, \ldots, s_n]) \cdot f^{\#}([t_1, t_2, \ldots, t_m])$, and
3. $f^{\#} \circ i = f$.

Now assume that some other $f^{\#'}$ also satisfies these conditions. For any $L \in \mathit{List}(S)$, we show, by induction on the length of L, that $f^{\#'}(L) = f^{\#}(L)$, and thus that $f^{\#'} = f^{\#}$. If $L = []$, the first condition forces $f^{\#'}(L) = e = f^{\#}(L)$. If $L = [s_1, s_2, \ldots, s_n]$ $(n \geq 1)$, then $L = [s_1] * [s_2, \ldots, s_n]$. By the third condition, $f^{\#'}([s_1]) = f^{\#'}(i(s_1)) = f(s_1) = f^{\#}(i(s_1)) = f^{\#}([s_1])$. By the induction hypothesis, $f^{\#'}([s_2, \ldots, s_n]) = f^{\#}([s_2, \ldots, s_n])$. Now by the second condition, $f^{\#'}(L) = f^{\#'}([s_1]) \cdot f^{\#'}([s_2, \ldots, s_n]) = f^{\#}([s_1]) \cdot f^{\#}([s_2, \ldots, s_n]) = f^{\#}(L)$. *(End of Proof)*

The function $f^{\#}$ is called the "extension of f" because it agrees with f on the elements of S, that is, $f^{\#}([s]) = [f(s)]$.

Let us make this construction even more concrete by considering a particular instance.

78 Example (Adapted from Rydeheard [76].) The operation of the *length* function can be described by a set of recursive equations:

$$\begin{aligned} \mathit{length}([]) &= 0 \\ \mathit{length}(L * L') &= \mathit{length}(L) + \mathit{length}(L') \\ \mathit{length}(i(s)) &= 1. \end{aligned}$$

The two monoids involved are $(\mathit{List}(S), *, [])$ and $(\mathbf{Z}^+, +, 0)$. The first two lines of the definition say that length is a homomorphism from the monoid of lists over the set S to the monoid of natural numbers. The third line says that the following triangle commutes (where 1 is the constant function mapping every $s \in S$ to 1):

$$\begin{tikzcd} S \arrow[r, "i"] \arrow[dr, "1"'] & \mathit{List}(S) \arrow[d, "length"] \\ & \mathbf{Z}^+ \end{tikzcd}$$

The properties of monoid homomorphisms together with the commutativity of the triangle correspond exactly to the definition of the length function. Thus, saying that the definition is proper—that it actually defines a function—is the same as asserting that there is a unique arrow in the category **Mon** satisfying the commuting triangle.

This motivates the general definition of adjunction:

79 Definition An *adjunction* consists of

- a pair of categories $\mathbf{C}$ and $\mathbf{D}$;
- a pair of functors $F : \mathbf{C} \to \mathbf{D}$ and $G : \mathbf{D} \to \mathbf{C}$;
- a natural transformation $\eta : I_{\mathbf{C}} \xrightarrow{\cdot} (G \circ F)$;

such that for each **C**-object X and **C**-arrow $f : X \to G(Y)$, there is a unique **D**-arrow $f^{\#} : F(X) \to Y$ such that the following triangle commutes:

$$\begin{tikzcd}
X \arrow[r, "\eta_X"] \arrow[dr, "f"'] & G(F(X)) \arrow[d, dashed, "G(f^{\#})"] \\
& G(Y)
\end{tikzcd}$$

The reader should check that this definition correponds exactly to the example above. The categories **C** and **D** are **Set** and **Mon**, respectively. The functor F is *List* : **Set** $\to$ **Mon**. The functor G is the forgetful functor U : **Mon** $\to$ **Set** that takes each monoid $(M, \cdot, e)$ to its underlying set M and each homomorphism to the corresponding function on the underlying sets. The natural transformation η is the family of functions $i_S : S \to List(S)$ that take each element of S to a singleton list. The *length* function is the $f^{\#}$ corresponding to the f that takes every element of S to the number 1.

In the example the forgetful functor U was left implicit in several places (for example, wherever we considered a monoid homomorphism as a function on sets). Writing U everywhere it belongs, the diagram in the example matches the one in the definition precisely:

$$\begin{tikzcd}
S \arrow[r, "i"] \arrow[dr, "1"'] & U((List(S), *, [])) \arrow[d, "U(length)"] \\
& U((\mathbf{Z}^+, +, 0))
\end{tikzcd}$$

80 Remarks

- We say that $\langle F, G\rangle$ is an ***adjoint pair*** of functors. F is the ***left adjoint*** of G. G is the ***right adjoint*** of F.
- The natural transformation η is called the ***unit*** of the adjunction. For each **D**-object Y there is an arrow $i^{\#}_{G(Y)} : F(G(Y)) \to Y$. This assignment defines a natural transformation $\epsilon : (F \circ G) \dot{\to} I_Y$ called the *co-unit* of the adjunction.
- The functors F and G determine each other and the natural transformations η and ϵ to within an isomorphism.

- A given functor F may or may not have a right or left adjoint. Freyd's Adjoint Functor Theorem gives necessary and sufficient conditions for the existence of left adjoints. (By duality, the same theorem characterizes the conditions for existence of right adjoints.) See Arbib and Manes [1, pp. 131ff], Mac Lane [54, pp. 116ff], Herrlich and Strecker [38, pp. 207ff], or any standard reference on category theory.
- The forgetful functor on a variety of algebras always has a left adjoint, but only occasionally a right adjoint. In the cases most frequently encountered, the left adjoint of a forgetful functor turns out to be something like a free algebra, while the right adjoint (when it exists) usually describes a subset with some closure property. See Cohn [14, p. 314].
- Adjoint functors have many pleasant theoretical properties. For example, functors that are left adjoints preserve colimits (i.e. map colimiting cones in the source category to colimiting cones in the target category) and, dually, right adjoints preserve limits.

There are other, equivalent definitions of adjunction. The one given here seems to be the simplest for gaining an initial grasp of the concept. However, practicing category theorists normally prefer to think in terms of an isomorphism

$$\mathbf{D}(F(X), Y) \cong \mathbf{C}(X, G(Y))$$

that is natural in both X and Y—i.e., a two-variable natural transformation that preserves structure as both X and Y vary and that is a bijection for all X and Y.

Another way of saying the same thing is that adjointness occurs when there is an exact correspondence between $\mathbf{D}$ arrows from $F(X)$ to Y and $\mathbf{C}$-arrows from X to $G(Y)$:

$$\begin{tikzcd} X \arrow[r, "F"] \arrow[d, dashed] & F(X) \arrow[d, dashed] \\ G(Y) & Y \arrow[l, "G"'] \end{tikzcd}$$

(Note that in this "diagram," the two dotted arrows and the horizontal arrows are in three different categories!)

The bijection is often presented schematically:

$$\frac{X \to G(Y)}{F(X) \to Y}$$

The reader who has persevered this far is urged to consult a standard textbook for more details on alternative treatments of adjoints. For the remainder of this section we will stick to the original definition and attempt to deepen the concept with several examples.

Categorical limits and colimits have a simple interpretation in terms of adjunctions, as the next three examples show.

81 Example The initial object *0* in a category **C** arises as a left adjoint to the constant functor $T : \mathbf{C} \rightarrow \mathbf{1}$, where **1** is the one-object category. The unit of the adjunction is the single arrow in **1**. The co-unit picks out the unique **C**-arrow from *0* to each **C**-object.

82 Example If **C** is a category with a product object $A \times B$ for every pair of objects A and B, then the product functor $P : \mathbf{C} \times \mathbf{C} \rightarrow \mathbf{C}$, which takes a pair of objects $\langle A, B\rangle$ to the product object $A \times B$, is the right adjoint of the "diagonal functor" $\Delta : \mathbf{C} \rightarrow \mathbf{C} \times \mathbf{C}$ taking each **C**-object C to $\langle C, C\rangle$:

$$\frac{C \rightarrow A \times B}{\langle C, C\rangle \rightarrow \langle A, B\rangle}$$

The correspondence with Definition 79 is:

$$\begin{array}{lcl} F & \equiv & \Delta \\ G & \equiv & P \\ X & \equiv & C \\ Y & \equiv & \langle A, B\rangle \\ \eta_C & \equiv & \langle \mathrm{id}_C, \mathrm{id}_C\rangle. \end{array}$$

The universal diagram for the unit of the adjuntion is:

$$\begin{tikzcd} C \arrow[r, "\langle \mathrm{id}_C, \mathrm{id}_C\rangle"] \arrow[dr, "\langle f, g\rangle"'] & C \times C \arrow[d, "f \times g"] \\ & A \times B \end{tikzcd}$$

83 Example In the definition of exponentiation (Definition 58), the assignment of a g^* to each g establishes a bijection between the sets $\mathbf{C}(C \times A, B)$ and $\mathbf{C}(C, B^A)$. To see this, suppose $g^* = h^*$. Then $g = eval_{AB} \circ (g^* \times \mathrm{id}_A) = eval_{AB} \circ (h^* \times \mathrm{id}_A) = h$, so the assignment is injective. On the other hand, for any $g^{*'} : C \rightarrow B^A$, define $g = eval_{AB} \circ (g^{*'})$. By the uniqueness of g^*, it must be the case that $g^* = g^{*'}$, so the assignment is also surjective.

This correspondence of sets of arrows signals the presence of an adjunction:

$$\frac{C \rightarrow B^A}{C \times A \rightarrow B}$$

The details are as follows. First, we pick a **C**-object A and hold it fixed during the construction. Now, the "right product" functor $(\text{—} \times A)$ of Example 64 has as its right adjoint the functor $(\text{—})^A$ of Example 71.

In this example, it is the *co*-unit of the adjunction that is most revealing. In general, the co-unit is a natural transform $\epsilon : (F \circ G) \dot{\rightarrow} I_B$, such that for each arrow $g : F(X) \rightarrow Y$

there is a unique arrow $g^* : X \to G(Y)$ for which the following diagram commutes: The universal diagram for the co-unit of the adjunction is:

$$\begin{tikzcd}
F(G(Y)) \arrow[r, "\epsilon_Y"] & Y \\
F(X) \arrow[u, "F(g^*)"] \arrow[ur, "g"'] &
\end{tikzcd}$$

In the present instance, ϵ_Y is just the evaluation arrow $eval_{AB}$ of Definition 58 and, for each $g : C \times A \to B$ the arrow $g^* : C \to B^A$ is the same as the one defined in Definition 58. Filling in the labels on the diagram that defines the co-unit, we see that it exactly matches Definition 58:

$$\begin{tikzcd}
B^A \times A \arrow[r, "eval_{AB}"] & B \\
C \times A \arrow[u, "g^* \times \mathrm{id}_A"] \arrow[ur, "g"'] &
\end{tikzcd}$$

Finally, remember that the construction so far has assumed a fixed **C**-object A. The category **C** "has exponentiation" if the functor $(- \times A)$ has a right adjoint for *every* A.

Many other mathematical constructions are examples of adjoints. Here are some briefer examples illustrating the range of situations where adjunctions can be found:

84 Example Let **Int** $= (\mathbf{Z}, \leq)$ and **Real** $= (\mathbf{R}, \leq)$ be the integers and reals with the usual ordering, both considered as categories. It is easy to see that the inclusion $U : \mathbf{Int} \to \mathbf{Real}$ is a functor. In the other direction, the ceiling function $\lceil r \rceil$, taking each $r \in \mathbf{R}$ to the smallest integer greater or equal to r, is also a functor ($r \leq r'$ implies that $\lceil r \rceil \leq \lceil r' \rceil$, where "$\leq$" in each case stands for an arrow). In fact, the ceiling functor is left adjoint to U. To see this, observe that $r \leq U(\lceil r \rceil)$ for each r. This is the unit of the adjunction. The universal property of $U(\lceil r \rceil)$ in the diagram of Definition 79 corresponds to the word "smallest" in the definition of the ceiling function.

This example appears in Rydeheard and Burstall's book [78], where it is attributed to Pratt. Adjunctions between partial orders are also known as *Galois connections* (see [54,60]).

85 Example (Also from Rydeheard and Burstall [78].) The category **Graph** has directed multi-graphs as its objects. An arrow $f : G \to H$ in **Graph** is a structure-preserving map between graphs, that is, a mapping v from vertices of G to the vertices of H and a mapping e from the edges of G to the edges of H, such that for each edge x of G, the endpoints in

H of the image of x under e are the images under v of the endpoints of x in G. (Note the similarity to the definition of functors!)

Two nodes m and n in a graph G are said to be strongly connected if there is a path in G from m to n and a path from n to m. A subgraph $C \subseteq G$ is strongly connected if every pair of nodes in C is strongly connected. A strong component of a graph is a maximal strongly-connected subgraph. The strong components of a graph themselves form an acyclic graph which is a quotient of the original graph (that is, each node corresponds to an equivalence class of strongly-connected nodes in the original). The mapping taking a graph to the acyclic graph of its strongly-connected components may be expressed as a left adjoint to the inclusion functor from **AcyclicGraph** to **Graph**.

86 Example Goguen [25] defines a category of finite state automata and a category of observable behaviors. The "minimal realization" functor turns out to be left adjoint to the "behavior of" functor. (Also see [29].)

87 Exercises

1. Dualize Example 81
2. The universal diagram for the unit of the product adjunction in Example 82 does not correspond directly to the universal property of products in Definition 36. Draw the co-unit diagram.
3. Show that the categorical coproduct (see the exercise in Section 2.4) arises as a left adjoint to the diagonal functor Δ.
4. What is the *unit* of the adjunction is Example 83? Give an intuitive interpretation to the mapping from f to $f^{\#}$.
5. Show that the floor function from **Real** to **Int** is right adjoint to the inclusion U.

2.13 Cartesian Closed Categories

...But is category theory the long-sought answer? No, no, not at all. Category theory *pure* provides nothing explicitly aside from identity functions... as it stands, category theory has no existential import. (It was not meant to.) Set theory has "too much" existential import. (It was meant to.) What we seek is the middle way—and an argument that the middle way is natural and general.

There is no need to build up unnecessary suspense: the middle way is the theory of the (so called) *cartesian closed categories*....

— Scott [86, p. 408]

Cartesian closed categories, introduced by Lawvere, are essentially "function spaces" in a categorical setting. It has been shown that there is an exact correspondence between CCCs and certain typed lambda calculi, which provides a convenient algebraic treatment of models for lambda calculi. Also, CCCs can be used as the basis of a translation of functional languages into variable-free "combinator expressions," which can be efficiently interpreted by a "categorical abstract machine." (See [15].)

88 Definition A *cartesian closed category* (CCC) is a category with

1. a terminal object 1;
2. products (that is, a product object $A \times B$ for every pair of objects A and B);
3. exponentiation.

89 Example The category **Set** is cartesian closed, with $B^A = \mathbf{Set}(A, B)$.

90 Example The category **Cat** is cartesian closed, with $\mathbf{B}^{\mathbf{A}}$ the functor category (Example 72).

Example 83 motivates an alternative definition of cartesian closedness:

91 Alternate Definition A category **C** is cartesian closed if it has products, and if the functor $(- \times A) : \mathbf{C} \to \mathbf{C}$ has a right adjoint for every **C**-object A.

92 Exercises

1. Given a set S, show that the partial order $(\mathcal{P}(S), \subseteq)$ is a cartesian closed category.
2. Prove that in any cartesian closed category, $B^{A \times A'}$ is naturally isomorphic to $(B^A)^{A'}$ for all objects A, A', and B.
3. Let S be the set of sentences of propositional logic. We can consider S as a preorder $(S, \leq)$, where $p \leq q$ means that from p we can derive q directly from some axiomatization of the logic. Show that S forms a cartesian closed category, with product given by conjunction of propositions and the exponential q^p corresponding to "p implies q".

2.14 Topoi†

The average mathematician, who regards category theory as "generalized abstract nonsense," tends to regard topos theory as generalized abstract category theory. And yet S. Mac Lane regards the rise of topos theory as a symptom of the *decline* of abstraction in category theory, and of abstract algebra in general....

What, then, is the topos-theoretic outlook? Briefly, it consists in the rejection of the idea that there is a fixed universe of "constant" sets within which mathematics can and should be developed, and the recognition that the notion of "variable structure" may be more conveniently handled within a universe of *continuously variable* sets than by the method, traditional since the rise of abstract set theory, of considering separately a domain of variation (i.e. a topological space) and a succession of constant structures attached to the points of this domain....

— Johnstone [42, p. xvi]

A ***topos*** is a cartesian closed category with an object, called a ***subobject classifier***, that represents truth values. This allows the definition of an "internal logic" of the topos. It turns out that there is an exact correspondence between toposes (or ***topoi***) and theories in higher-order intuitionistic logic (see [47]).

The theory of topoi is an extremely rich and sophisticated area of study. We give just the bare definition here, with no explanation of how it works or where it is used. (Thinking of the correspondence between subsets and characteristic functions in set theory may provide some glimmers of intuition.) For more information, see the textbooks of Lambek and Scott [47], Johnstone [42], Goldblatt [31], or Barr and Wells [5], or the introductory papers of Mac Lane [53] or Freyd [24].

93 Definition Let **C** be a category with terminal object 1. A ***subobject classifier*** for **C** is a **C**-object Ω together with a **C**-arrow $t : 1 \to \Omega$ such that for each monic arrow $f : A \to C$, there is a unique $\chi_f : C \to \Omega$ (called the "character of f") that makes the following diagram commute:

$$\begin{tikzcd}
A \arrow[r, "f"] \arrow[d, "!"'] & C \arrow[d, "\chi_f"] \\
1 \arrow[r, "t"'] & \Omega
\end{tikzcd}$$

(Recall that ! stands for the unique arrow from A to 1.)

94 Definition A ***topos*** is a cartesian closed category equipped with a subobject classifier.

95 Example The categories **Set**, **Cat**, **Set**$^{\to}$, and (for any **B** and **C**) $\mathbf{C}^{\mathbf{B}}$ are all topoi.

Chapter 3

Case Studies

This section uses the notation developed in the tutorial to sketch some actual applications of category theory in computer science. The first two subsections cover the essentials of two papers on the *design* of programming languages using category theory. The third outlines some of the applications of category in the semantic *description* of programming languages. All three subjects are treated here just deeply enough to give a feel for how category theory is brought to bear in different situations. The fourth section describes in more detail the use of category theory as a *unifying tool* in domain theory. Enterprising readers should find that the material in the tutorial makes the original papers reasonably accessible.

3.1 Categorical Type Systems

Tatsuya Hagino has developed the idea of using category theory as the basis for the type structure of a programming language. His paper on "A Typed Lambda Calculus with Categorical Type Constructors" [35] describes a uniform category-theoretic mechanism for declaring types and presents a lambda calculus incorporating this mechanism. He shows that the evaluation of expressions in this calculus always terminates even though the calculus can be used to define infinite data structures.

One of the simplest programming languages is the simply-typed lambda calculus. Its type structure is captured by two rules:

1. some predefined set of base types is given;
2. for any two types σ and τ, the expression $\sigma \rightarrow \tau$ is also a type.

The calculus itself is then defined by giving rules for forming typed terms (programs) and rules for reducing (executing) these terms.

It is fairly easy to prove some useful theorems about this language. For example, every well-typed program is strongly normalizing—that is, every program can be reduced to a normal (or canonical) form, and a normal form will always be reached no matter in which order the reduction rules are applied.

But this language is too simple to be of any practical value. Before we can use it to express interesting programs, we need to specify some base types (the natural numbers, for example) and add some more type constructors like products (records) and sums (variant records or disjoint unions). Unfortunately, this complicates the language and makes properties like strong normalization more difficult (or even impossible) to prove. Worse yet, these properties must be proved again each time we think of another type constructor or base type that we've forgotten to add.

The proper approach, of course, is to try to work in a more abstract setting by defining a small set of powerful type constructors from which all of the ones we want can be derived as special cases. Then we can prove the theorems we care about once and for all. One such "high-level constructor" is the least-fixed-point operator $\underline{\mu}$. If σ is a type expression with a free variable ρ, then $\underline{\mu}\rho.\sigma$ denotes the least type satisfying the recursive equation

$$\underline{\mu}\rho.\sigma = \sigma[(\underline{\mu}\rho.\sigma)/\rho],$$

where $\sigma[(\underline{\mu}\rho.\sigma)/\rho]$ denotes the result of replacing all occurrences of ρ in σ with $\underline{\mu}\rho.\sigma$.

The type of natural numbers, for example, no longer needs to be primitive. It can be defined as

$$Nat \equiv \underline{\mu}\rho.1 + \rho,$$

where 1 is a type with one element and + is the sum (disjoint union) constructor. This is good, but of course we still have to define 1 and + as primitives. It would be desirable, if possible, to eliminate even these.

At this point, category theory enters the picture. Hagino describes a general *categorical* type constructor called "F, G-dialgebras." Both the least fixed point operator $\underline{\mu}$ and its dual, the *greatest* fixed point operator $\overline{\mu}$, arise as special objects in the category of F, G-dialgebras. Other type constructors arise, in turn, as combinations of $\underline{\mu}$ and $\overline{\mu}$.

96 Definition Let **C** and **D** be categories and let F and G be functors from **C** to **D**. The category **DAlg**(F, G) of F, G-*dialgebras* is defined as follows:

- its objects are pairs $\langle A, f\rangle$ where A is a **C**-object and $f : F(A) \to G(A)$ is a **D**-arrow;
- its arrows $h : \langle A, f\rangle \to \langle B, g\rangle$ are **C**-arrows $h : A \to B$ for which the following diagram commutes:

$$\begin{tikzcd}
F(A) \arrow[r, "f"] \arrow[d, "F(h)"'] & G(A) \arrow[d, "G(h)"] \\
F(B) \arrow[r, "g"'] & G(B)
\end{tikzcd}$$

To illustrate the definition, let us demonstrate how dialgebras can be used to define products. (A similar construction can be given for any kind of left or right adjoint.)

97 Example Let **C** be a category with products and let A and B be two **C**-objects.

Define the functors $F, G : \mathbf{C} \to \mathbf{C} \times \mathbf{C}$ by

$$\begin{aligned} F(C) &= \langle C, C \rangle \\ F(f) &= \langle f, f \rangle \\ G(C) &= \langle A, B \rangle \\ G(f) &= \langle \mathrm{id}_A, \mathrm{id}_B \rangle. \end{aligned}$$

It can be shown that $\mathbf{DAlg}(F, G)$ has a terminal object *1*. In this case, let $1 = \langle A \times B, \langle \pi_1, \pi_2 \rangle \rangle$, that is, assign the *names* $A \times B$, π_1, and π_2 to the components of the terminal F, G-dialgebra. (The reader may want to pause and check the details of what we have done.)

Now, because $\langle A \times B, \langle \pi_1, \pi_2 \rangle \rangle$ is a terminal object in $\mathbf{DAlg}(F, G)$, there must be a unique $\mathbf{DAlg}(F, G)$-arrow h from any $\mathbf{DAlg}(F, G)$-object $\langle C, \langle f, g \rangle \rangle$ to $\langle A \times B, \langle \pi_1, \pi_2 \rangle \rangle$. Furthermore, from the definition of F, G-dialgebras, this h will be such that the following diagram commutes:

$$\begin{tikzcd} \langle C, C \rangle \arrow[r, "\langle f, g \rangle"] \arrow[d, "\langle h, h \rangle"'] & \langle A, B \rangle \arrow[d, "\langle \mathrm{id}_A, \mathrm{id}_B \rangle"] \\ \langle A \times B, A \times B \rangle \arrow[r, "\langle \pi_1, \pi_2 \rangle"'] & \langle A, B \rangle \end{tikzcd}$$

But this is just a different way of drawing the ordinary universal diagram for products:

$$\begin{tikzcd} & C \arrow[dl, "f"'] \arrow[d, dashed, "\langle f, g \rangle"] \arrow[dr, "g"] & \\ A & A \times B \arrow[l, "\pi_1"] \arrow[r, "\pi_2"'] & B \end{tikzcd}$$

Terminal objects in the category of F, G-dialgebras turn out to correspond exactly to the types defined by equations of the form $\overline{\mu}\rho.\sigma$. Initial objects correspond to types defined by equations of the form $\underline{\mu}\rho.\sigma$. This means that F, G-dialgebras van be used as the *semantics* of a lambda calculus whose type constructors are $\underline{\mu}$ and $\overline{\mu}$ (and $\to$). Hagino presents such a calculus, gives its reduction rules, and shows the the calculus is strongly normalizing by arguing that the class of functions that can be computed in it are just the primitive recursive functions.

The datatypes that can be defined in the calculus include not only natural numbers, products, sums, and recursive data types like lists, but also, via the $\overline{\mu}$ constructor, *infinite* structures like infinite (sometimes called "lazy") lists and trees. This suggests an elegant way of adding facilities for "lazy functional programming" to languages like Standard ML [36]. In fact, Hagino's language CPL (described in his Ph.D. thesis [34]) goes one step further, eliminating even the basic $\to$ type constructor by giving up lambda calculus itself and working with left and right adjoints in a completely categorical setting.

3.2 Implicit Conversions and Generic Operators

John Reynolds, in his paper on "Using Category Theory to Design Implicit Conversions and Generic Operators," [69] has applied category theory to a thorny problem in the design of programming languages.

Most languages support at least a limited form of "generic operators." For example, it would be a shame to have to distinguish two separate addition operators:

$$\begin{array}{lll} +_{Int} & : & Int \times Int \rightarrow Int \\ +_{Real} & : & Real \times Real \rightarrow Real. \end{array}$$

Instead, the single operator + is considered to have two different signatures:

$$\begin{array}{lll} + & : & Int \times Int \rightarrow Int \\ + & : & Real \times Real \rightarrow Real. \end{array}$$

The compiler must now decide which of these is intended in a given situation.

Another convenience provided by many languages is the ability to write an *Int* in a context where a *Real* is expected to appear, relying on the compiler to insert an "implicit conversion" from *Int* to *Real* as required by the context. For example, if x is a *Real* variable and i is an *Int* variable, we could simply write

$$x{:=}i$$

instead of

$$x{:=}\mathit{Int\text{-}to\text{-}Real}(i).$$

The designers of some languages (notably PL/I and Algol 68) have actually combined these two mechanisms, making it possible to write

$$x{:=}i+j$$

instead of

$$x{:=}\mathit{Int\text{-}to\text{-}Real}(i) +_{Real} \mathit{Int\text{-}to\text{-}Real}(j)$$

or

$$x{:=}\mathit{Int\text{-}to\text{-}Real}(i +_{Int} j).$$

But a question immediately arises: which of these did we mean? Unfortunately, the usual approach—trying to specify exactly where implicit conversions will be inserted in expressions involving generic operators—has led in practice to complex, confusing, and even inconsistent language definitions. Reynolds suggests an alternative approach, based on the observation that, mathematically speaking, it *doesn't matter* which of the interpretations of $x{:=}i+j$ is chosen by the compiler. He suggests, in fact, that this observation be enshrined as a requirement for the design of the conversions and generic operators themselves: their specification should not be considered well-formed unless they can be inserted in any order by the compiler without affecting the meaning of any program.

The mathematical tools Reynolds chooses to carry out this program are a generalization of Higgins' Algebras with Schemes of Operations [39], which in turn can be thought of as a generalization of the conventional Many-Sorted Algebras used by Goguen, Thatcher, Wagner, and Wright [30]. (The roots of this approach to the semantics of programs go back to the work of Burstall and Landin [12].) Reynolds calls his formalism Category-Sorted Algebra.

The essence of the approach lies in viewing the types of a given programming language as forming a partial order Ω, for example:

$$\begin{tikzcd}
\mathit{NS}\ \text{(the universal type)} \arrow[d, "\leq"'] \arrow[dr, "\leq"'] & \\
\mathit{Real} \arrow[d, "\leq"'] & \mathit{Bool} \\
\mathit{Int} &
\end{tikzcd}$$

This partial order can then be considered as a category, where the unique Ω-arrow $\sigma \leq \tau$ represents the fact that values of type σ may be implicitly converted to values of type τ. The actual conversion functions are the images of the $\leq$ arrows under a functor $B : \Omega \rightarrow$ Set. (The object part of this functor maps each type σ to the set of values of type σ.)

Finally, the fact that generic operators and implicit conversions may be inserted in either order by the compiler corresponds to the commutativity of diagrams like the following:

$$\begin{tikzcd}
B(\mathit{Int}) \times B(\mathit{Int}) \arrow[r, "\gamma_+(\mathit{Int},\mathit{Int})"] \arrow[d, "\begin{array}{r}B(\mathit{Int} \leq \mathit{Real})\\ \times B(\mathit{Int} \leq \mathit{Real})\end{array}"'] & B(\mathit{Int}) \arrow[d, "B(\mathit{Int} \leq \mathit{Real})"] \\
B(\mathit{Real}) \times B(\mathit{Real}) \arrow[r, "\gamma_+(\mathit{Real},\mathit{Real})"'] & B(\mathit{Real})
\end{tikzcd}$$

(Here γ_+ is a *family* of Set-functions for performing addition, indexed by the types of the two values being added.)

Such diagrams, Reynolds shows, may elegantly be viewed as natural transformations.

This view of data types leads to a general notion of algebraic semantics, which Reynolds uses in the second half of the paper to analyze a simple expression language derived from Algol 60. (Also see his paper on "The Essence of Algol" [70].) More recently, he has used similar techniques in designing the type system of the language Forsythe [74] and, with Frank Oles, in describing the semantics of imperative languages with nested block structure [62,63]. The notes for his course on Semantics as a Design Tool [68] cover the details of category-sorted algebras and related constructions in great depth. The techniques of Reynolds and Oles are also discussed in an introductory article by Tennent [90].

3.3 Semantics

One area of computer science where the relevance of category theory is practically undisputed is semantic models of programming languages. Peter Dybjer has surveyed the work in this area in his article, "Category Theory and Programming Language Semantics: an Overview" [16].

Dybjer distinguishes ***mathematical semantics***, which concerns methods for interpreting programming languages in mathematical structures from set theory, algebra, or topology, from ***operational semantics***. The applicability of category theory to the latter seems to be minimal. Within mathematical semantics, Dybjer distinguishes topological methods (denotational semantics and domain theory) from algebraic (universal algebra). Both were influenced early on by category theory. In domain theory, Scott [83] showed that the continuous lattices with continuous functions form a cartesian closed category. In algebraic semantics, "algebraic theories" were developed by authors such as Elgot [20,21], Burstall and Thatcher [13], and the ADJ group [95]. [The idea of giving a categorical description of theories is originally due to Lawvere [49].]

Later, important papers by Wand [94], Smyth and Plotkin [88], and Lehmann and Smith [51] used category theory to unify different methods of solving domain equations and to connect denotational semantics with initial algebra semantics. Plotkin [67] developed lecture notes for a whole course on domain theory in categorical terms. Barendregt [3] found a natural unifying description of various models of the lambda calculus in terms of cartesian closed categories.

Dybjer goes on to discuss three specific approaches in more depth:

- The connection between category theory and ***type theory*** is based on Lambek's observation [44,45,46] that the cartesian closed categories are in perfect correspondence with a certain class of typed lambda calculi (specifically, the $\lambda\beta\eta$-calculi with surjective pairing). This has led a number of theorists to the conclusion that cartesian closure provides the appropriate notion of a model of the typed lambda calculus [86].

- The connection between category theory and ***domain theory*** is very rich, but quite complicated. Domain theory incorporates the notion of "partial elements" in order to assign meaning to programs whose computation may not terminate. This requirement has given rise to a number of formulations of domains as cartesian closed categories [7,33,66,67,83,84,85,87].

- The connection between category theory and ***algebraic semantics*** arises from the notion of "initial" or "free" algebra [12,21,27,95]. The abstract syntax of a programming language may be thought of as the initial object in a category of Σ-Algebras, where Σ is intuitively the grammar defining the language. A "meaning map" is then the unique homomorphism from the initial algebra to some semantic algebra.

One controversial point in any discussion of the applicability of category theory to computer science is ***how much*** of category theory people are interested in using. Some authors (e.g.

Reynolds) use category theory simply as a powerful and uniform notational framework for masses of complicated but relatively shallow detail. On the other side, Dybjer cites papers by Lehmann [52] and Goguen and Burstall [26] where what he feels are deep theorems of category theory are applied to computational situations.

3.4 Recursive Domain Equations

One of the great successes of category theory in computer science has been the development of a "unified theory" of the constructions underlying denotational semantics, specifically the solution of recursive domain equations. Equations like

$$D \cong At + (D \to D)$$

(where At is a fixed domain) can be solved by finding least fixed points of functions mapping domains to domains—in this case, of the function

$$f(X) = At + (X \to X).$$

(The solution to this equation, in particular, provides a good domain for the semantics of an untyped λ-calculus computing over some collection of atoms.)

Scott's inverse limit construction [83] provided the basic insight that the key to finding such fixed points is taking "$A \to B$" to mean not *all* functions from A to B, but only the continuous ones. (This construction is also discussed by Reynolds [72] and Stoy [89], and in an excellent expository chapter in Schmidt's book [81].) But there are many possible definitions of "domain" and "continuous" for which the construction can be carried out; the details in each case are similar, but not identical. To cope with a proliferation of special cases, it was important to find a general characterization of the conditions under which a given equation would have a solution over a given class of domains.

Smyth and Plotkin's paper, "The Category-Theoretic Solution of Recursive Domain Equations" [88] builds on earlier work of Wand [92,93] to give a definitive treatment of these matters. This section develops a simplified version of their results.

Our central example throughout the section will be the equation $F(X) = At + (X \to X)$. The key shift of perspective from the domain-theoretic to the more general category-theoretic approach lies in considering F not as a function on domains, but as a *functor* on a category of domains. Instead of a least fixed point of the function, we will be looking for an *initial* fixed point of the functor.

We turn now to the task of developing enough of the general theory that we can state all of this precisely.

98 Definition Let $\mathbf{K}$ be a category and $F : \mathbf{K} \to \mathbf{K}$ a functor. A *fixed point* of F is a pair $\langle A, a\rangle$, where A is a $\mathbf{K}$-object and $a : FA \to A$ is an isomorphism. A *prefixed point* of F is a pair $\langle A, a\rangle$, where A is a $\mathbf{K}$-object and a is any arrow from FA to A.

Prefixed points of a functor F are also called F-algebras. They form the objects of a category:

99 Definition Let $\mathbf{K}$ be a category and $F : \mathbf{K} \to \mathbf{K}$ a functor. The category $\mathbf{F}$-$\mathbf{Alg}$ has as objects the prefixed points of F. Given objects $\langle A, a\rangle$ and $\langle A', a'\rangle$ of $\mathbf{F}$-$\mathbf{Alg}$, an arrow $f : \langle A, a\rangle \to \langle A', a'\rangle$ (called an *F-homomorphism*) is a $\mathbf{K}$-arrow $f : A \to A'$ such that the following diagram commutes:

$$\begin{tikzcd}
FA \arrow[r, "a"] \arrow[d, "Ff"'] & A \arrow[d, "f"] \\
FA' \arrow[r, "a'"'] & A'
\end{tikzcd}$$

100 Fact (Lemma 1 of [88].) The initial F-algebra, if it exists, is also the initial fixed point of F in $\mathbf{K}$.

This allows us to work in the more structured setting of F-algebras. Next, we need to define some basic conditions on $\mathbf{K}$ and F that ensure the existence of an initial F-algebra.

101 Definition An *ω-chain* in a category $\mathbf{K}$ is a diagram of the following form:

$$\Delta = D_0 \xrightarrow{f_0} D_1 \xrightarrow{f_1} D_2 \xrightarrow{f_2} \cdots$$

An *ω^{op}-chain* is a diagram of the following form:

$$\Delta = D_0 \xleftarrow{f_0} D_1 \xleftarrow{f_1} D_2 \xleftarrow{f_2} \cdots$$

Recall that a *cocone* $\mu : \Delta \to X$ of of an ω-chain Δ is a $\mathbf{K}$-object X and a collection of $\mathbf{K}$-arrows $\{\mu_i : D_i \to X \mid i \geq 0\}$ such that for all $i \geq 0$, $\mu_i = \mu_{i+1} \circ f_i$.

Dually, a *cone* $\mu : X \to \Delta$ of an ω^{op}-chain Δ is a $\mathbf{K}$-object X and a collection of $\mathbf{K}$-arrows $\{\mu_i : X \to D_i \mid i \geq 0\}$ such that for all $i \geq 0$, $\mu_i = f_i \circ \mu_{i+1}$.

A *colimit* $\mu : \Delta \to X$ is a cocone with the property that if $\nu : \Delta \to X'$ is also a cocone, then there exists a unique mediating arrow $k : X \to X'$ (we will often say "from μ to ν") such that for all $i \geq 0$, $\nu_i = k \circ \mu_i$. Colimits of ω-chains are sometimes referred to as *ω-colimits*.

An *ω^{op}-limit* of an ω^{op}-chain Δ is a cone $\mu : X \to \Delta$ with the property that if $\nu : X' \to \Delta$ is also a cone, then there exists a unique mediating arrow $k : X' \to X$ such that for all $i \geq 0$, $\mu_i \circ k = \nu_i$.

We write $\bot_{\mathbf{K}}$ (or just $\bot$) for the initial object of $\mathbf{K}$ (if it has one), and $!_{\bot\to A}$ for the unique arrow from $\bot$ to each $\mathbf{K}$-object A. It is also convenient to write

$$\Delta^- = D_1 \xrightarrow{f_1} D_2 \xrightarrow{f_2} \cdots$$

for all of Δ except D_0 and f_0. By analogy, μ^- is $\{\mu_i \mid i \geq 1\}$. For the images of Δ and μ under F we write

$$F\Delta = FD_0 \xrightarrow{Ff_0} FD_1 \xrightarrow{Ff_1} FD_2 \xrightarrow{Ff_2} \cdots$$

and $F\mu = \{F\mu_i \mid i \geq 0\}$.

With these definitions in hand, we can state Smyth and Plotkin's "Basic Lemma":

102 Lemma Let $\mathbf{K}$ be a category with initial object $\perp$ and let $F : \mathbf{K} \to \mathbf{K}$ be a functor. Define the ω-chain Δ by

$$\Delta = \perp \xrightarrow{!_{\perp\to F(\perp)}} F(\perp) \xrightarrow{F(!_{\perp\to F(\perp)})} F^2(\perp) \xrightarrow{F^2(!_{\perp\to F(\perp)})} \cdots$$

where by F^i we mean the iterated composition (i times) of F: $F^0(f) = f$, $F^1(f) = F(f)$, $F^2(f) = F(F(f))$, etc. If both $\mu : \Delta \to D$ and $F\mu : F\Delta \to FD$ are colimits, then the initial F-algebra exists and is $\langle D, d\rangle$, where $d : FD \to D$ is the mediating arrow from $F\mu$ to μ^-.

Proof: Let $\langle D', d'\rangle$ be any F-algebra. Define $\nu : \Delta \to D'$ by

$$\begin{aligned} \nu_0 &= !_{\perp\to D'}, \\ \nu_{n+1} &= d' \circ F(\nu_n). \end{aligned}$$

To show that ν is a cocone, we prove by induction that the following diagram commutes for all n:

$$\begin{tikzcd} F^n(\perp) \arrow[r, "F^n(!_{\perp\to F(\perp)})"] \arrow[dr, "\nu_n"'] & F^{n+1}(\perp) \arrow[d, "\nu_{n+1}"] \\ & D' \end{tikzcd}$$

For $n = 0$ this is clear, since $\perp$ is initial. For $n + 1$ we have:

$$\begin{aligned} \nu_{n+2} \circ F^{n+1}(!_{\perp\to F(\perp)}) &= d' \circ F(\nu_{n+1}) \circ F^{n+1}(!_{\perp\to F(\perp)}) && \text{(by the definition of } \nu) \\ &= d' \circ F(\nu_{n+1} \circ F^n(!_{\perp\to F(\perp)})) && \text{(since } F \text{ is a functor)} \\ &= d' \circ F(\nu_n) && \text{(by ind. hyp.)} \\ &= \nu_{n+1} && \text{(by definition).} \end{aligned}$$

We need to show that there is a unique F-homomorphism $f : \langle D, d\rangle \to \langle D', d'\rangle$.

First, suppose f is such a homomorphism. The uniqueness of f follows from the fact that it is the mediating arrow from μ to ν; to see this, we use induction again to show that $\nu_n = f \circ \mu_n$ for each n. Again, the case for $n = 0$ is clear. For $n + 1$, we have:

$$\begin{aligned} f \circ \mu_{n+1} &= f \circ d \circ F(\mu_n) && \text{(by the definition of } d) \\ &= d' \circ F(f) \circ F(\mu_n) && \text{(since } f \text{ is an } F\text{-homomorphism)} \\ &= d' \circ F(f \circ \mu_n) && \text{(since } F \text{ is a functor)} \\ &= d' \circ F(\nu_n) && \text{(by the induction hypothesis)} \\ &= \nu_{n+1} && \text{(by definition).} \end{aligned}$$

Second, to show that f exists, we *define* it as the mediating arrow from μ to ν (so that $\nu_n = f \circ \mu_n$ for all $n \geq 0$). We will show that $f \circ d$ and $d' \circ Ff$ are both mediating arrows from $F\mu$ to ν^-, which implies that they are equal, and that f is an F-homomorphism as required.

In the first case,

$$\begin{array}{rcll} (f \circ d) \circ F\mu_n & = & f \circ \mu_{n+1} & \text{(by the definition of } d\text{)} \\ & = & \nu_{n+1} & \text{(by the definition of } f\text{).} \end{array}$$

In the second case,

$$\begin{array}{rcll} (d' \circ Ff) \circ F\mu_n & = & d' \circ F(f \circ \mu_n) & \text{(since } F \text{ is a functor)} \\ & = & d' \circ F(\nu_n) & \text{(by the definition of } f\text{)} \\ & = & \nu_{n+1} & \text{(by the definition of } \nu\text{).} \end{array}$$

(*End of Proof*)

The Basic Lemma is a fundamental tool for finding initial fixed points of functors. But applying it directly in each situation where a fixed point is needed would be quite tedious (see Exercise 114). To make it more useful, Smyth and Plotkin go on to develop a theory of "O-categories" that captures the essentials of what is required to apply the lemma (mainly, the use of embeddings). This allows them to state simpler conditions guaranteeing the applicability of the lemma. The remainder of this section summarizes a part of this theory, using the example from the beginning of the section as motivation.

To begin an attack on the example, we need to be more precise about where we want to solution to live.

103 Definition Recall that a ***partial order*** is a set P equipped with a reflexive, transitive, antisymmetric relation $\sqsubseteq_P$. (We usually omit the subscript.) An ***ω-sequence*** is a sequence $\{p_i \mid i \geq 0\} \subseteq P$ in which $\forall I \geq 0$, $p_i \sqsubseteq p_{i+1}$. An ***upper bound*** of an ω-sequence $\{p_i \mid i \geq 0\}$ is an element p such that $\forall i \geq 0$, $p_i \sqsubseteq p$. The ***least upper bound*** (or ***lub***) of $\{p_i \mid i \geq 0\}$ is an upper bound that is less than or equal to every other upper bound—that is, an element p such that p is an upper bound of of $\{p_i \mid i \geq 0\}$ and, whenever p' is also an upper bound of $\{p_i \mid i \geq 0\}$, $p \sqsubseteq p'$.

A given ω-sequence may or may not have a lub. If an ω-sequence $\{p_i \mid i \geq 0\}$ does have a lub, it is written $\bigsqcup^P_{n\geq 0} p_n$ (or just $\bigsqcup p_n$).

A partial order P in which every ω-sequence has a lub is called ***ω-complete***. If it also has a least element (written $\bot_P$ or just $\bot$), P is called an ***ω-complete pointed partial order***.

A function $f : P \to Q$ (where P and Q are ω-complete pointed partial orders) is ***monotonic*** iff for all $p_1, p_2 \in P$, $p_1 \sqsubseteq p_2$ implies $f(p_1) \sqsubseteq f(p_2)$. It is ***continuous*** iff it is monotonic and for each ω-sequence $\{p_i \mid i \geq 0\}$ it is the case that $f(\bigsqcup^P_{n\geq 0} p_n) = \bigsqcup^Q_{n\geq 0}(f(p_n))$.

104 Definition The category **CPO** has ω-complete pointed partial orders as objects and ω-continuous functions as arrows.

105 Exercise Check that **CPO** satisfies the category laws of Definition 1.

This is almost a category where we can build a solution to the example equation, but we need one further refinement: the notion of a category of embeddings.

106 Definition A category **K** is an **O**-*category* iff

1. for every pair of **K**-objects A and B, the hom-set $\mathbf{K}(A,B)$ is a partial order in which every ω-sequence has a lub;
2. composition of **K**-arrows is an ω-continuous operation with respect to this ordering—that is, if $f \sqsubseteq_{\mathbf{K}(A,B)} f'$ and $g \sqsubseteq_{\mathbf{K}(B,C)} g'$, then $g \circ f \sqsubseteq_{\mathbf{K}(A,C)} g' \circ f'$, and if $\{f_i \mid i \geq 0\}$ is an ω-sequence in $\mathbf{K}(A,B)$ and $\{g_i \mid i \geq 0\}$ is an ω-sequence in $\mathbf{K}(B,C)$, then $\bigsqcup(g_n \circ f_n) = (\bigsqcup g_n) \circ (\bigsqcup f_n)$.

For example, **CPO** is an **O**-category when its hom-sets are ordered "pointwise":

$$f \sqsubseteq_{\mathrm{CPO}(A,B)} f' \quad \text{iff} \quad \forall a \in A.\, f(a) \sqsubseteq_B f'(a).$$

107 Definition Let **K** be an **O**-category and let $f : A \to B$ be a **K**-arrow such that for some arrow $f^R : B \to A$,

$$f^R \circ f = \mathrm{id}_A \quad \text{and} \quad f \circ f^R \sqsubseteq \mathrm{id}_B.$$

Then f is called an *embedding* and f^R is called a *projection*.

108 Fact Each embedding f determines a unique projection f^R and vice versa.

109 Definition If Δ is an ω-chain in an **O**-category **K**,

$$\Delta = D_0 \xrightarrow{f_0} D_1 \xrightarrow{f_1} D_2 \xrightarrow{f_2} \cdots$$

where each f_i is an embedding, then we write Δ^R for the ω^{op}-chain obtained by replacing each embedding f_i with the corresponding projection f_i^R:

$$\Delta^R = D_0 \xleftarrow{f_0^R} D_1 \xleftarrow{f_1^R} D_2 \xleftarrow{f_2^R} \cdots$$

110 Definition Let **K** be an **O**-category. The *category of embeddings of* **K**, written $\mathbf{K}^E$, has as objects the objects of K and as arrows the **K**-arrows that are embeddings.

It is in $\mathbf{CPO}^E$ that we can construct an initial fixed point of the functor $F(X) = At + (X \to X)$. In order to express F as a functor from $\mathbf{CPO}^E$ to $\mathbf{CPO}^E$, we need to define the more primitive functors At, $+$, and $\to$. Primarily because of the contravariance of $\to$ in its first argument, we must take a somewhat roundabout route. First, we define At, $+$, and $\to$ over **CPO** and $\mathbf{CPO}^{\mathrm{op}}$ rather than $\mathbf{CPO}^E$. We then use them to build a functor $G : \mathbf{CPO}^{\mathrm{op}} \times \mathbf{CPO} \to \mathbf{CPO}$, from which we derive a functor $G^E : \mathbf{CPO}^E \times \mathbf{CPO}^E \to \mathbf{CPO}^E$. Finally, from G^E we derive the functor $F^E : \mathbf{CPO}^E \to \mathbf{CPO}^E$.

111 Definition

1. For any **CPO**-object A, the constant functor A is defined by:

$$\begin{array}{rcl} A(B) & = & A, \\ A(f) & = & \mathrm{id}_A. \end{array}$$

2. The functor $+ : \mathbf{CPO} \times \mathbf{CPO} \to \mathbf{CPO}$ is defined on objects by

$$\begin{array}{rcl} A+B & = & \{\langle 0,a\rangle \mid a \in A\} \\ & \cup & \{\langle 1,b\rangle \mid b \in B\} \\ & \cup & \{\bot_{A+B}\}, \end{array}$$

where the partial order on $A+B$ is given by

$$\begin{array}{rcl} c \sqsubseteq_{A+B} c' & \text{iff} & (c = \bot_{A+B}) \\ & \vee & (\exists a, a' \in A.\, c = \langle 0,a\rangle \ \wedge\ c' = \langle 0,a'\rangle \ \wedge\ a \sqsubseteq_A a') \\ & \vee & (\exists b, b' \in B.\, c = \langle 1,b\rangle \ \wedge\ c' = \langle 1,b'\rangle \ \wedge\ b \sqsubseteq_B b'). \end{array}$$

The action of $+$ on arrows $f : A \to A'$ and $g : B \to B'$ is:

$$(f+g)(c) = \begin{cases} \langle 0, f(a)\rangle & \text{if } \exists a \in A.\, c = \langle 0,a\rangle \\ \langle 1, g(b)\rangle & \text{if } \exists b \in B.\, c = \langle 1,b\rangle \\ \bot_{A'+B'} & \text{otherwise.} \end{cases}$$

3. The functor $\to : \mathbf{CPO}^{\mathrm{op}} \times \mathbf{CPO} \to \mathbf{CPO}$ is defined on objects by

$$A \to B = \mathbf{CPO}(A,B),$$

the ω-complete pointed partial order whose elements are the continuous functions from A to B under the pointwise ordering:

$$f \sqsubseteq_{A \to B} f' \quad \text{iff} \quad \forall a \in A.\, f(a) \sqsubseteq_B f'(a).$$

The action of $\to$ on a $\mathbf{CPO}^{\mathrm{op}}$-arrow $f : A \to A'$ (that is, an ω-continuous function from A' to A) and a **CPO**-arrow $g : B \to B'$ is:

$$(f \to g)(h) = g \circ h \circ f.$$

We can't quite define F from these components. In fact, F is not a functor on **CPO** because it only takes one parameter and uses it on both sides of the $\to$ functor, which is contravariant in one argument and covariant in the other. For our F acting on objects, this is no problem. But if the argument to F is an arrow $f : A \to B$ we get stuck: there is no way to get from this arrow to one running the opposite direction, because we cannot always derive an ω-continuous function $f' : B \to A$ from an ω-continuous function $f : A \to B$. However, we *can* define a functor $G : \mathbf{CPO}^{\mathrm{op}} \times \mathbf{CPO} \to \mathbf{CPO}$ that is only a small step away from F:

$$\begin{array}{rcl} G\langle X,Y\rangle & = & At + (X \to Y), \\ G\langle f,g\rangle & = & At + (f \to g). \end{array}$$

The next step is to find a functor $G^E : \mathbf{CPO}^E \times \mathbf{CPO}^E \to \mathbf{CPO}^E$.

112 Definition Let $\mathbf{K}$ be an $\mathbf{O}$-category. A functor $T : \mathbf{K}^{\mathrm{op}} \times \mathbf{K} \to \mathbf{K}$ is *locally monotonic* iff it is monotonic on the hom-sets of $\mathbf{K}$—that is, if for $f, f' : A \to B$ in $\mathbf{K}^{\mathrm{op}}$ and $g, g' : C \to D$ in $\mathbf{K}$, $f \sqsubseteq f'$ and $g \sqsubseteq g'$ imply that $T\langle f, g\rangle \sqsubseteq T\langle f', g'\rangle$.

113 Fact (Special case of Smyth and Plotkin's Lemma 4.) If $T : \mathbf{K}^{\mathrm{op}} \times \mathbf{K} \to \mathbf{K}$ is locally monotonic, it can be used to define a covariant functor $T^E : \mathbf{K}^E \times \mathbf{K}^E \to \mathbf{K}^E$ by putting

$$\begin{aligned} T^E\langle X,Y\rangle &= T\langle X,Y\rangle, \\ T^E\langle f,g\rangle &= T\langle f^R,g\rangle. \end{aligned}$$

For our example, this functor is:

$$\begin{aligned} G^E\langle X,Y\rangle &= At + (X \to Y), \\ G^E\langle f,g\rangle &= \mathrm{id}_{At} + (f^R \to g). \end{aligned}$$

Now, since G^E is covariant in both its arguments, we can define F^E simply by:

$$\begin{aligned} F^E(X) &= G^E\langle X,X\rangle, \\ F^E(f) &= G^E\langle f,f\rangle, \end{aligned}$$

or more explicitly:

$$\begin{aligned} F^E(X) &= At + (X \to X), \\ F^E(f) &= \mathrm{id}_{At} + (f^R \to f). \end{aligned}$$

At this point, it would be possible to check the conditions of the basic lemma directly. The diagram Δ in $\mathbf{CPO}^E$ is:

$$\Delta = \bot \xrightarrow{!_{\bot \to F^E(\bot)}} F^E(\bot) \xrightarrow{F^E(!_{\bot \to F^E(\bot)})} (F^E)^2(\bot) \xrightarrow{(F^E)^2(!_{\bot \to F^E(\bot)})} \cdots$$

To be more succinct in what follows, we can abbreviate

$$\begin{aligned} \Delta_i &= (F^E)^i(\bot) \\ \delta_i &= (F^E)^i(!_{\bot \to F^E(\bot)}), \end{aligned}$$

and write:

$$\Delta = \Delta_0 \xrightarrow{\delta_0} \Delta_1 \xrightarrow{\delta_1} \Delta_2 \xrightarrow{\delta_2} \cdots$$

The colimit object D is the ω-complete pointed partial order whose elements are infinite tuples of "compatible" elements of the Δ_i's,

$$D = \{\langle x_0, x_1, x_2, \ldots\rangle \mid \forall i \geq 0.\ x_i \in \Delta_i \ \wedge\ x_i = \delta_i^R(x_{i+1})\},$$

with the "componentwise" ordering:

$$\langle x_0, x_1, x_2, \ldots \rangle \sqsubseteq \langle x'_0, x'_1, x'_2, \ldots \rangle \quad \text{iff} \quad \forall i \geq 0.\ x_i \sqsubseteq x'_i.$$

The elements of the colimit $\mu : \Delta \rightarrow D$ are given by:

$$\mu_i(x_i) = \langle \ldots, \delta^R_{i-2}(\delta^R_{i-1}(x_i)), \delta^R_{i-1}(x_i), x_i, \delta_i(x_i), \delta_{i+1}(\delta_i(x_i)), \ldots \rangle$$

114 Exercise (Difficult.) The reader can gain a better appreciation for the importance of the more general theory to follow by working through the details of applying the basic lemma. The main steps are checking that:

1. D is an ω-complete pointed partial order;
2. μ is a colimit:
   (a) each μ_i is an embedding;
   (b) μ is a cocone;
   (c) if $\nu : \Delta \rightarrow D'$ is a cocone, then $k : D \rightarrow D'$, defined by $k = \bigsqcup(\nu_n \circ \mu^R_n)$, satisfies:
      i. k is an embedding;
      ii. $\forall i \geq 0.\ k \circ \mu_i = \nu_i$;
      iii. k is the unique arrow satisfying $\forall i \geq 0.\ k \circ \mu_i = \nu_i$;
3. $F\mu : F\Delta \rightarrow FD$ is a colimit.

Readers not familiar with the details of the domain-theoretic version of the inverse limit construction should consult a textbook for guidance. (Schmidt [81] is a good choice: he works with domains that are similar to these, and provides about the right level of detail.)

We now summarize the steps in the more general approach. In brief, it consists of defining "global" conditions on a category $\mathbf{K}^E$ and a functor T^E that ensure the applicability of the basic lemma, and then finding easily checkable "local" conditions on $\mathbf{K}$ and T that imply the global conditions.

115 Definition A category $\mathbf{K}$ is an *ω-complete pointed category* (or just *ω-category*) iff it has an initial element and every ω-chain has a colimit.

116 Definition A functor $F : \mathbf{K} \rightarrow \mathbf{K}$ is *ω-continuous* iff it preserves ω-colimits—that is, if whenever Δ is an ω-chain and $\mu : \Delta \rightarrow A$ is a colimit, $F\mu : F\Delta \rightarrow FA$ is also a colimit.

117 Fact If $\mathbf{K}$ is an ω-category and $F : \mathbf{K} \rightarrow \mathbf{K}$ is ω-continuous, the conditions of the basic lemma are satisfied.

These are the global conditions. The local condition for a category $\mathbf{K}^E$ to be an ω-category is the existence in $\mathbf{K}$ of *ω-limits*. (This is an instance of the well-known "limit/colimit" coincidence noticed by Scott [83].)

118 Fact (See Smyth and Plotkin's Theorem 2.) Let **K** be an **O**-category and Δ be an ω-chain in $\mathbf{K}^E$. If Δ^R has a limit in **K**, then Δ has a colimit in $\mathbf{K}^E$.

119 Fact **CPO** has all ω^{op}-limits.

Proof Sketch: Let Δ be the ω^{op}-chain:

$$\Delta = D_0 \xleftarrow{f_0} D_1 \xleftarrow{f_1} D_2 \xleftarrow{f_2} \cdots$$

Then the limit D (an ω-complete pointed partial order) is

$$D = \{\langle d_0, d_1, d_2, \ldots\rangle \mid \forall i \geq 0.\ d_i \in D_i \ \wedge\ d_i = f_i(d_{i+1})\}$$

under the componentwise ordering. The elements of the limit $\nu : D \to \Delta$ are the projections:

$$\nu_i(\langle d_0, d_1, d_2, \ldots\rangle) = d_i.$$

(*End of Proof*)

From these facts, it follows that $\mathbf{CPO}^E$ is an ω-category. All that remains is to show that F^E is ω-continuous.

120 Definition Let **K** be an **O**-category and $\mu : \Delta \to A$ a cocone in $\mathbf{K}^E$. Then μ is an **O**-*colimit* of Δ provided that $\{\mu_n \circ \mu_n^R \mid n \geq 0\}$ is an ω-sequence in the ordering on $\mathbf{K}(A, A)$, and that $\bigsqcup(\mu_n \circ \mu_n^R) = \mathrm{id}_A$.

The motivation for the definition of **O**-colimits is purely technical: these are exactly the conditions that are needed to make Smyth and Plotkin's Theorem 2 go through. (But they are not arbitrary: they should be familiar to anyone who has been carefully through the details of a domain-theoretic inverse limit construction.)

121 Definition An **O**-category **K** is said to have *locally determined ω-colimits of embeddings* provided that whenever Δ is an ω-chain in $\mathbf{K}^E$, μ is a colimit of Δ in $\mathbf{K}^E$ iff μ is an **O**-colimit of Δ.

122 Fact (Corollary to Smyth and Plotkin's Theorem 2.) Suppose that **K** is an **O**-category in which every ω^{op}-chain has a limit. Then **K** has locally determined colimits of embeddings.

123 Definition A functor $T : \mathbf{K}^{op} \times \mathbf{K} \to \mathbf{K}$ is *locally continuous* iff it is continuous on the hom-sets of **K**—that is, if whenever $\{f_n : A \to B \mid n \geq 0\}$ is an ω-sequence in $\mathbf{K}^{op}(A, B)$ and $\{g_n : C \to D \mid n \geq 0\}$ is an ω-sequence in $\mathbf{K}(C, D)$, then $T\langle\bigsqcup f_n, \bigsqcup g_n\rangle = \bigsqcup(T\langle f_n, g_n\rangle)$.

124 Fact (Smyth and Plotkin's Theorem 3.) If $T : \mathbf{K}^{op} \times \mathbf{K} \to \mathbf{K}$ is locally continuous and **K** has locally determined colimits of embeddings, then T^E is ω-continuous.

Our functor G is easily shown to be locally continuous (from the definitions of $+$, $\rightarrow$, and constant functors, and the fact that composition of functors preserves local continuity). Thus, G^E is ω-continuous, from which it is easy to see that F^E is ω-continuous. By Fact 117, the conditions of the basic lemma are therefore satisfied by $\mathbf{CPO}^E$ and F^E. This gives us an initial F^E-algebra, which by Lemma 100 is also the initial fixed point of F^E in $\mathbf{CPO}^E$. Because the objects of $\mathbf{CPO}^E$ are the same as those of **CPO** and an isomorphism in $\mathbf{CPO}^E$ is also an isomorphism in **CPO**, this gives us a solution in **CPO** to the equation $D \cong F^E(D)$.

Chapter 4

Literature Survey

4.1 Textbooks

Categories for the Working Mathematician (Mac Lane [54]) is the standard, heavyweight reference for category theory. It cannot easily be used as an introduction to the subject, since it assumes considerable mathematical maturity, and (especially for the examples) expertise in areas such as algebraic topology that the computer science reader almost certainly does not posess. Nevertheless, Mac Lane's writing is sufficiently lucid that following along at 10% comprehension can be as valuable as checking every detail of another book. His volume belongs on the bookshelf of every serious student of the field.

Topoi: The Categorial Analysis of Logic (Goldblatt [31]) may be the best book for the beginner. It is criticized by category theorists for being misleading on some aspects of the subject, and for presenting long and difficult proofs where simpler ones are available. On the other hand, it makes liberal use of simple, set-theoretic examples and motivating intuitions—much more so than any other introduction. Although Goldblatt's main topic is topoi, the first 75 pages are devoted to standard category theoretic fundamentals and the later chapters on functors and adjoints can be read without the intervening material on topos theory. (Other standard works on topoi and categorical logic include books by Johnstone [42] Barr and Wells [5], and Lambek and Scott [47], and articles by Freyd [24] and Mac Lane [53].)

Computational Category Theory (Rydeheard and Burstall [78]) extends the authors' work on "programming up category theory," described in several earlier articles [10,11,80] and (in greater depth) in Rydeheard's thesis [79]. Starting from the observation that "categories themselves are the models of an essentially algebraic theory and nearly all the derived concepts are finitary and algorithmic in nature," it presents a self contained introduction to essentially the same parts of category theory as Section 2 of the present paper, in almost entirely computational terms. Each concept is defined as a datatype in ML and each construction as a working algorithm.

Arrows, Structures, and Functors: The Categorical Imperative (Arbib and Manes [1]) was, for several years, the only reasonable introduction to category theory for nonspecialists. Its treatment of the important basic concepts is fairly complete, and it provides a number of clear examples from areas of college algebra and automata theory that readers are likely to be familiar with. However, some of the examples—especially those in automata theory—may be more involved than their intrinsic interest justifies. The exposition is extremely clear on the easier material but less so in difficult sections.

Algebraic Approaches to Program Semantics (Manes and Arbib [58]) is a later book by the same authors. It presents a self-contained exposition of basic category theory and two different approaches to categorical denotational semantics—the *order semantics* of Scott and Strachey, and the authors' own *partially additive semantics*. Except for "mathematically mature" readers, it is a bit too dense to be a good first book on either category theory or semantics.

Categories (Blyth [9]) is a short introduction to the basic categorical notions. The writing is terse and the examples are drawn solely from mathematics. The computer scientist whose purpose is simply to gain enough grounding in category theory to read research papers in computer science, as opposed to delving into category theory for its own sake, may find it too much work. However, it includes numerous exercises with solutions, making it a good choice for self-study if the reader has some background in algebra.

Category Theory for Computer Science (Barr and Wells [4]) is still in draft form at the time of this writing, but promises to be an excellent addition to the literature. Despite alluring claims in the preface that the only prerequisites are "some familiarity with abstract mathematical thinking, and some specific knowledge of the basic language of computer science of the sort taught in an introductory discrete mathematics course," this book is not for the faint-hearted. The exposition is terse, abstract, and mostly lacking in explicit connections to previously-known subjects. Nevertheless, the coverage of aspects of category theory relevant to computer science is very complete, including several topics not covered in most introductions (including the Grothendiek construction, the unifying notion of "sketches as a systematic way to turn finite descriptions into mathematical objects," cartesian closed categories, and toposes).

Universal Algebra (Cohn [14]) is the standard reference on universal algebra, which has numerous applications in the study of semantics. (Familiar structures studied in this field include lattices, free algebras, boolean algebras, and formal grammars.) Much of the presentation is couched in categorical terminology.

4.2 Introductory Articles

"A Junction Between Computer Science and Category Theory: I, Basic Definitions and Concepts (part 1)" (Goguen, Thatcher, Wagner, and Wright [29]) is the first of a well-known series of articles by the "ADJ group." It begins with a nice discussion of the relevance of category theory to computer science, introduces some background definitions and notation for sets and algebras, and develops the concepts of categories and functors. Copious examples are provided, mostly from algebra and automata theory.

"A Junction Between Computer Science and Category Theory: I, Basic Definitions and Concepts (part 2)" (Goguen, Thatcher, Wagner and Wright [28]) continues the previous report. It covers graphs and diagrams and their relation to categories, as well as natural transformations. Again, the discussion is supplemented with numerous examples. One significant example—a categorical technique for proving correctness and termination of flow-diagram programs—is developed at length. The presentation of category theory in these two reports is far from complete, but pedagogically excellent: all definitions and examples are presented carefully and in good detail. Unfortunately, the notation they use to accomplish this tends to be somewhat heavy (and occasionally nonstandard).

"An Introduction to Categories, Algebraic Theories and Algebras" (Goguen, Thatcher, Wagner and Wright [27]) presents the ADJ group's categorical approach to universal algebra, based on Lawvere's concept of an algebraic theory [50]. It consists of a fairly self-contained introduction to basic category theory, developed in parallel with an exposition of algebraic theories and their applications to universal algebra.

"Notes on Algebraic Fundamentals for Theoretical Computer Science" (Thatcher and Wright [91]) is a broad, terse summary of what the authors feel is the necessary groundwork for mathematically sound work in theoretical computer science. In addition to a section on categories (which emphasises adjoints), the notes include sections on set theory, partial orders, many-sorted algebras, ordered algebras, continuous algebras, algebraic theories, and the solution of equations within theories.

"Cartesian Closed Categories and Typed λ-Calculi" (Lambek [44]) is the most accessible (but still quite technical!) introduction to the isomorphism between cartesian closed categories and typed lambda calculi, written by the developer of the idea. (See Lambek and Scott's book [47] for a more complete development, and a paper by Huet [40] for an alternative formulation of the theory. Curien's categorical combinators [15] are based on a similar idea.)

"Relating Theories of the Lambda Calculus" (Scott [86]) gives a different development of the relation between cartesian closed categories and lambda calculi. The first section motivates CCCs as a general "theory of types," from the perspective of the philosophy of logic. It then reveals that theories in typed lambda calculus are just cartesian closed categories. Later sections discuss the relation between typed and untyped lambda calculus, intuitionistic type theories and CCCs, and combinatory algebras. The early sections of the paper are not too technical for beginners.

4.3 Reference Books

Category Theory (Herrlich and Strecker [38]) is an excellent comprehensive reference on all aspects of pure category theory. Unlike most category theory texts, its level of pedagogical care is high enough (and its prerequisites modest enough) that it can profitably be read by any computer scientist who wants to understand a categorical concept in maximum depth and generality. It is currently out of print.

Categories (Schubert [82]) is another good, heavyweight reference on pure category theory.

Abelian Categories (Freyd [23]) is another useful reference, particularly for the notion of representability.

Theory of Categories (Mitchell [61]) was the first comprehensive exposition of category theory.

Introduction to Higher Order Categorical Logic (Lambek and Scott [47]) is an attempt to reconcile mathematical logic with category theory as approaches to the foundations of mathematics. The first section is a terse overview of category theory from perspective of categorical logic. The second section shows that typed lambda calculi are equivalent to cartesian closed categories, and that untyped lambda calculi are similarly related to certain algebras. The second section explores the relationship between intuitionistic type theory and toposes. The final section discusses the representation of numerical functions (recursion theory) in various categories.

Categorical Combinators, Sequential Algorithms and Functional Programming (Curien [15]) describes a "Categorical Abstract Machine" based on the connection between cartesian closed categories and typed lambda calculi and intended as a practical implementation technique for functional languages.

Categories for Denotational Semantics (Asperti and Longo [2]) is a streamlined introduction to the applications of parts of category theory in denotational semantics. Topics include some of the standard ones—cartesian closed categories and lambda calculi (presented in semantic terms, however, rather than the usual syntactic style), universal arrows and adjunctions, cones and limits—as well as newer work by the authors and their colleagues on partial-morphisms, internal category theory, and internal CCCs as models of the polymorphic lambda calculus. It is still in draft form.

Algebraic Theories (Manes [59]) studies "equationally-definable classes" both set-theoretically and as a category. For the computer scientist, it is an abstract view of the mathematical structures from universal algebra and category theory that form the basis of many algebraic approaches to semantics. The book assumes some knowledge of algebra and toplogy, but includes a self-contained presentation of "enough category theory for our needs and at least as much as every pure mathematician should know."

Algebra (Mac Lane and Birkhoff [56]) is a comprehensive treatment of abstract algebra at the undergraduate level, organized according to the unifying "categorical insights" that have emerged in that field in the latter half of this century. Category theory *per se* is introduced at the end of the book, generalizing the special cases that have appeared throughout (i.e. showing the step from concrete categories to arbitrary categories, and from universal constructions to adjoints).

The Lambda Calculus (Barendregt [3]) is well known as the standard reference on lambda calculus. It includes a good discussion of models of the lambda calculus in arbitrary cartesian closed categories (see pp. 107, 477, etc.).

Category Theory Applied to Computation and Control (E.G. Manes, editor [57]) is the proceedings of one of the first important conferences on category theory in computer science. It includes a number of seminal papers.

Category Theory and Computer Programming (Pitt, Abramsky, Poigné, and Rydeheard, editors [64]) is a large conference proceedings with several tutorials on various aspects of basic category theory (some particularly good ones were mentioned above), as well as a number of important research papers.

Category Theory and Computer Science (Pitt, Poigné, and Rydeheard, editors [65]) is the proceedings of the most recent meeting of the same conference.

4.4 A Sampling of Research Papers

"Type Algebras, Functor Categories, and Block Structure" (Oles [63]) presents a condensed version of some of the results in the author's Ph.D. thesis [62]. It shows how category-theoretic notions, in particular functor categories, can be used to explain the semantics of languages with updatable stores, procedures, and block-structure. In such languages, not only is the store itself modifiable, but even the "shape" of the store can change as the program enters and exits blocks. Functors and categories of functors provide an elegant way of describing stores, semantics of expressions and programs, and algebras of types.

"Preliminary Design of the Programming Language Forsythe" (Reynolds [74]) describes the design of a language that attempts to capture the "essence" of Algol 60 [70] in as general and uniform a framework as possible. Category theory plays a central organizing role in the design and description of Forsythe's type system.

"Continuous Lattices" (Scott [83]) establishes the existence of semantic domains satisfying isomorphism equations like $D \cong D \rightarrow D$, thus providing the first known models of the untyped lambda calculus. Although this paper is couched in topological rather than categorical terms, the influence of categorical intuitions is apparent.

"Profinite Solutions for Recursive Domain Equations" (Gunter [33]) studies the category of "profinite" semantic domains, "an especially natural and, in a sense, *inevitable* class of spaces," and addresses some difficulties with solving recursive domain equations over the profinite semantic domains.

"On Functors Expressible in the Polymorphic Typed Lambda Calculus" (Reynolds and Plotkin [75]) develops a categorical proof of the nonexistence of a model of the polymorphic typed lambda calculus [71] in which types denote sets and $S \rightarrow S'$ denotes the set of all functions from S to S'. The paper is a joint exposition of Plotkin's generalization (to an abstract categorical setting) of an earlier result by Reynolds [73].

"Doctrines in Categorical Logic" (Kock and Reyes [43]) is a survey of category-theoretic methods in logic, organized by "doctrines," that is, "...categorical analogues of fragments of logical theories which have sufficient category-theoretic structure for their models to be described as functors." Equational, cartesian, finitary coherent, and infinitary coherent logic are covered, as well as (briefly) higher order logic and set theory.

"A Categorical Unification Algorithm" (Rydeheard and Burstall [80]) is an interesting example of using categorical reasoning to derive an algorithm. It is based on the observation that unification can be considered as a coequalizer in an appropriate category. Adding some basic theorems about the construction of coequalizers provides a correctness proof of a recursive construction of the unification function. Finally, the whole construction is encoded in Standard ML. (The same derivation also appears as a chapter in Rydeheard and Burstall's book [78].)

Universal Theory of Automata: A Categorical Approach (Ehrig et al [17]) presents a unified description of a theory of automata, encompassing deterministic, partial, linear, topological, nondeterministic, relational, and stochastic automata in a common (categorical) framework.

Appendix A

Proof of the Limit Theorem

Theorem Let $\mathbf{D}$ be a diagram in a category $\mathbf{C}$, with sets V of vertices and E of edges. If every V-indexed and every E-indexed family of objects in $\mathbf{C}$ has a product, and if every pair of arrows in $\mathbf{C}$ has an equalizer, then $\mathbf{D}$ has a limit.

Proof: (Adapted from Arbib and Manes [1, p. 45]. Also see [54, p. 109].) Begin by forming the following V-indexed and E-indexed products and projections:

$$\begin{tikzcd}
 & & D_J \\
\prod_{I\in V} D_I \arrow[urr, "\pi_J"] \arrow[d, "\pi_I"] & & \prod_{I\xrightarrow{e}J\in E} D_J \arrow[u, "\pi_e"'] \arrow[d, "\pi_e"] \\
D_I \arrow[rr, "D_e"] & & D_J
\end{tikzcd}$$

For each D_J at the top of the diagram there is an arrow $\pi_J : (\prod_{I\in V} D_I) \to D_J$. By the universal property of indexed products, this implies the existence of a unique arrow $p : (\prod_{I\in V} D_I) \to (\prod_{I\xrightarrow{e}J\in E} D_J)$ such that $\pi_e \circ p = \pi_J$ for each edge $e : I \to J$. Similarly, for each D_J at the bottom right there is an arrow $(D_e \circ \pi_I) : (\prod_{I\in V} D_I) \to D_J$, which implies the existence of a unique arrow $q : (\prod_{I\in V} D_I) \to (\prod_{I\xrightarrow{e}J\in E} D_J)$ such that $\pi_e \circ q = D_e \circ \pi_I$ for each edge $e : I \to J$. Let h be the equalizer of p and q. Set $f_I = \pi_I \circ h$ for each $I \in V$.

$$\begin{tikzcd}
 & & & D_J \\
X \arrow[r, "h"] \arrow[dr, "f_I"'] & \prod_{I\in V} D_I \arrow[urr, "\pi_J"] \arrow[d, "\pi_I"] \arrow[rr, dashed, shift left, "p"] \arrow[rr, dashed, shift right, "q"'] & & \prod_{I\xrightarrow{e}J\in E} D_J \arrow[u, "\pi_e"'] \arrow[d, "\pi_e"] \\
 & D_I \arrow[rr, "D_e"] & & D_J
\end{tikzcd}$$

We claim that $\{f_I : X \to D_I\}$ is a limit for $\mathbf{D}$. We must show first that it is a cone for $\mathbf{D}$, and furthermore that it is universal among cones for $\mathbf{D}$ (that if $\{f'_I : X' \to D_I\}$ is also a cone for **D**, then there exists a unique arrow $k : X' \to X$ such that $f_I \circ k = f'_I$ for every vertex I).

For each edge $e : I \to J$ in E, the commutativity of the diagram

$$\begin{tikzcd}
D_I \arrow[rr, "D_e"] & & D_J \\
& X \arrow[ul, "f_I"] \arrow[ur, "f_J"'] &
\end{tikzcd}$$

is established as follows (referring to the diagram above):

$$\begin{aligned}
D_e \circ f_I &= D_e \circ \pi_I \circ h && \text{(by the definition of } f_I\text{)} \\
&= \pi_e \circ q \circ h && \text{(by the universality of } q\text{)} \\
&= \pi_e \circ p \circ h && \text{(since } h \text{ equalizes } p \text{ and } q\text{)} \\
&= \pi_J \circ h && \text{(by the universality of } p\text{)} \\
&= f_J && \text{(by the definition of } f_J\text{).}
\end{aligned}$$

This shows that $\{f_I : X \to D_I\}$ is a cone. We must now show that it is universal among cones.

Assume that $\{f'_I : X' \to D_I\}$ is a cone for **D**. By the universal property of products, there is a unique arrow $h' : X' \to (\prod_{I \in V} D_I)$ such that $\pi_I \circ h' = f'_I$ for each $I \in V$. For any edge $e : I \to J$ in E,

$$\begin{aligned}
\pi_e \circ p \circ h' &= \pi_J \circ h' && \text{(by the definition of } p\text{)} \\
&= f'_J && \text{(by the definition of } h'\text{)} \\
&= D_e \circ f'_I && \text{(since } \{f'_I : X' \to D_I\} \text{ is a cone)} \\
&= D_e \circ \pi_I \circ h' && \text{(by the definition of } h'\text{)} \\
&= \pi_e \circ q \circ h' && \text{(by the definition of } q\text{).}
\end{aligned}$$

This establishes the commutativity of the diagram

$$\begin{tikzcd}
& X' \arrow[dl, "D_e \circ p \circ h' = D_e \circ q \circ h'"'] \arrow[d, shift right, "p \circ h'"'] \arrow[d, shift left, "q \circ h'"] \\
D_J & \prod_{I \xrightarrow{e} J \in E} D_J \arrow[l, "\pi_e"]
\end{tikzcd}$$

which, by the universal property of the product, implies that $p \circ h' = q \circ h'$.

Since h is an equalizer of p and q, the universal property of equalizers guarantees the existence of a unique $k : X' \to X$ such that $h \circ k = h'$. It is easy to see from the diagram

$$\begin{tikzcd}
X \arrow[rd, "h"] \arrow[rrd, "f_I"] & & \\
& \prod_{I \in V} D_I \arrow[r, "\pi_I"] & D_I \\
X' \arrow[uu, dashed, "k"] \arrow[ru, "h'"] \arrow[rru, "f'_I"'] & &
\end{tikzcd}$$

that

$$f_I \circ k = \pi_I \circ h \circ k = \pi_I \circ h' = f'_I.$$

Finally, we must show that the arrow k is unique. But if k' also satisfies $f_I \circ k' = f'_I$, then as $\pi_I \circ h \circ k' = \pi_I \circ h'$ for all $I \in V$, the universal property of the product guarantees (by the same argument as above) that $h \circ k' = h'$. The unique arrow with this property is k, so $k = k'$. *(End of Proof)*

125 Exercises

1. Specialize the proof of Theorem 57 to show how to construct limits of diagrams in **Set**.
2. Show that essentially the same construction gives limits of diagrams in **Poset**.
3. Apply the dual of Theorem 57 to show how to construct colimits of diagrams in **Set**

Appendix B

Summary of Notation

NOTATION	CONCEPT	SEE
$\mathbf{C}$	category	p. 7
A	object	p. 7
$f : A \to B$	arrow	p. 7
$f \circ g$	composition	p. 7
id_A	identity arrow	p. 7
$\mathrm{dom}\ f$, $\mathrm{cod}\ f$	domain and codomain	p. 7
$\mathbf{C}^{\mathrm{op}}$	dual category	p. 13
$\mathbf{C}^{\to}$	arrow category	p. 13
$\mathbf{C} \times \mathbf{D}$	product category	p. 13
$\langle A, B\rangle$	object of product category	p. 13
$\langle f, g\rangle$	arrow of product category	p. 13
$\langle A, B\rangle$	product object	p. 18
π_1, π_2	projections	p. 18
$(f, g) : C \to A \times B$	pair of arrows	p. 18
$f \times g : A \times B \to C \times D$	product arrows	p. 18
$\prod_{i \in S} A_i$	indexed product	p. 19
π_j	projections	p. 19
B^A	exponential object	p. 27
$eval : B^A \times A \to C$	evaluation arrow	p. 27
g^*	currying	p. 27
$F : \mathbf{C} \to \mathbf{D}$	functor	p. 28
$I_{\mathbf{C}}$	identity functor	p. 29
$\tau : F \dot{\to} G$	natural transformation	p. 31
$\tau_A : F(A) \to G(A)$	component of a natural transform	p. 31
$\mathit{0}, \mathit{1}$	initial and terminal objects	p. 16
$\mathbf{D}^{\mathbf{C}}$	functor category	p. 32
$f^{\#}$	extension of f	p. 34

Bibliography

[1] Michael Arbib and Ernest Manes. *Arrows, Structures, and Functors: The Categorical Imperative.* Academic Press, 1975.

[2] Andrea Asperti and Giuseppe Longo. Categories for denotational semantics. May 1988. Draft book.

[3] H. P. Barendregt. *The Lambda Calculus.* North Holland, Revised edition, 1984.

[4] Michael Barr and Charles Frederick Wells. Category theory for computer scientists. October 1987. Draft book.

[5] Michael Barr and Charles Frederick Wells. *Toposes, Triples, and Theories.* Springer-Verlag, 1984.

[6] Jean Bénabou. Fibered categories and the foundations of naive category theory. *Journal of Symbolic Logic*, 50(1):10–37, March 1985.

[7] G. Berry and P.L. Curien. Sequential algorithms on concrete data structures. *Theoretical Computer Science*, 20:265–321, 1982.

[8] A. Blass. The interaction between category theory and set theory. In J.W. Gray, editor, *Proc. of the Special Session on the Mathematical Applications of Category Theory, 89th meeting of the American Mathematical Society*, American Mathematical Society, 1984. Contemporary Mathematics, 30.

[9] T. S. Blyth. *Categories.* Longman, 1986.

[10] R. Burstall and D. Rydeheard. Computing with categories. In David Pitt, Samson Abramsky, Axel Poigné, and David Rydeheard, editors, *Category Theory and Computer Programming*, pages 506–519, Springer-Verlag, September 1985. LNCS 240.

[11] R.M. Burstall. Electronic category theory. In *Mathematical Foundations of Computer Science*, Rydzyna, Poland, 1980. Invited paper.

[12] R.M. Burstall and P.J. Landin. Programs and their proofs: an algebraic approach. *Machine Intelligence*, 4:17–43, 1969.

[13] R.M. Burstall and J.W. Thatcher. An algebraic theory of recursive program schemes. In E.G. Manes, editor, *Proceedings of the AAAS Symposium on Category Theory Applied to Computation and Control, San Francisco, California*, Springer-Verlag, 1975. LNCS 25.

[14] Paul M. Cohn. ***Universal Algebra.*** D. Reidel, revised edition, 1981. Originally published in 1965 by Harper and Row.

[15] P-L Curien. ***Categorical Combinators, Sequential Algorithms and Functional Programming.*** Pittman, 1986. Available from John Wiley and Sons.

[16] Peter Dybjer. Category theory and programming language semantics: an overview. In David Pitt, Samson Abramsky, Axel Poigné, and David Rydeheard, editors, *Category Theory and Computer Programming*, pages 165–181, Springer-Verlag, September 1985. LNCS 240.

[17] H. Ehrig, K.-D. Kiermeier, H.-J. Kreowski, and W. Kuehnel. ***Universal Theory of Automata: A Categorical Approach.*** B.G. Teubner, Stuttgart, 1974.

[18] S. Eilenberg and S. Mac Lane. General theory of natural equivalences. *Trans. Am. Math. Soc.*, 58:231–294, 1945.

[19] S. Eilenberg and S. Mac Lane. Group extensions and homology. *Ann. Math.*, 43:757–831, 1942.

[20] C.C. Elgot. Algebraic theories and program schemes. In E. Engeler, editor, *Symposium on Semantics of Algorithmic Languages*, pages 71–88, Springer-Verlag, 1971. Lecture Notes in Math 188.

[21] C.C. Elgot. Monadic computation and iterative algebraic theories. In *Logic Colloquium '73*, pages 175–230, North Holland, Bristol, England, 1975.

[22] Solomon Feferman. Set-theoretical foundations of category theory. In S. Mac Lane, editor, ***Reports of the Midwest Category Seminar III***, pages 201–247, Springer-Verlag, 1969. Lecture Notes in Mathematics, No. 106.

[23] Peter Freyd. ***Abelian Categories: An Introduction to the Theory of Functors.*** Harper and Row, 1964.

[24] Peter Freyd. Aspects of topoi. *Bull. Austral. Math. Soc.*, 7:1–76, 1972.

[25] J.A. Goguen. Realization is universal. *Math. Sys. Th.*, 6:359–374, 1973.

[26] J.A. Goguen and R.M. Burstall. Some fundamental tools for the semantics of computation; part 1: comma categories, colimits, signatures, and theories. *Theoretical Computer Science*, 31:175–209, 1984.

[27] J.A. Goguen, J. W. Thatcher, E.G. Wagner, and J.B. Wright. ***An Introduction to Categories, Algebraic Theories and Algebras.*** Technical Report RC-5369, IBM Research, April 1975.

[28] J.A. Goguen, J. W. Thatcher, E.G. Wagner, and J.B. Wright. *A junction between computer science and category theory: I, Basic definitions and concepts.* Technical Report RC-5908, IBM Research, March 1976. (part 2).

[29] J.A. Goguen, J. W. Thatcher, E.G. Wagner, and J.B. Wright. *A junction between computer science and category theory: I, Basic definitions and concepts.* Technical Report RC-4526, IBM Research, September 1973. (part 1).

[30] J.A. Goguen, J.W. Thatcher, E.G. Wagner, and J.B. Wright. Initial algebra semantics and continuous algebras. *J. ACM*, 24(1):68–95, 1977.

[31] Robert Goldblatt. *Topoi: The Categorial Analysis of Logic.* North Holland, 1984.

[32] A. Grothendeick. Catégories fibrées et descente. Revêtements étales et group fondamental. In *Séminaire de Gémétrie Algébrique du Bois-Marie 1960/61 (SGA 1), exposé VI*, Institut des Hautes Études Scientifiques, Paris, 1963. Reprinted in Lecture Notes in Math No. 224 (Springer-Verlag, 1971).

[33] Carl Gunter. *Profinite Solutions for Recursive Domain Equations.* PhD thesis, Carnegie-Mellon University, 1985. CMU-CS-85-107.

[34] Tatsuya Hagino. *A Category Theoretic Approach to Data Types.* PhD thesis, University of Edinburgh, 1987.

[35] Tatsuya Hagino. A typed lambda calculus with categorical type constructors. In D.H. Pitt, A. Poigne, and D.E. Rydeheard, editors, *Category Theory and Computer Science*, Springer-Verlag, September 1987. LNCS 283.

[36] Robert Harper, Robin Milner, and Mads Tofte. *The Semantics of Standard ML: Version 1.* Technical Report ECS-LFCS-87-36, Computer Science Department, University of Edinburgh, 1987.

[37] William S. Hatcher. *Foundations of Mathematics.* W. B. Saunders Co., 1968.

[38] H. Herrlich and G.E. Strecker. *Category Theory.* Allyn and Bacon, 1973.

[39] P.J. Higgins. Algebras with a schema of operators. *Math. Nachr.*, 27:115–132, 1963.

[40] Gerard Huet. Cartesian closed categories and lambda calculus. In Guy Cousineau, Pierre-Louis Curien, and Bernard Robinet, editors, *Combinators and Functional Programming Languages*, Springer-Verlag, May 1985. LNCS 242.

[41] W. Hurewicz. On duality theorems. *Bull. Am. Math. Soc.*, 47:562–563, 41.

[42] P.T. Johnstone. *Topos Theory.* Academic Press, 1977.

[43] A. Kock and G.E. Reyes. Doctrines in categorical logic. In J. Barwise, editor, *Handbook of Mathematical Logic*, pages 283–313, North Holland, 1977.

[44] J. Lambek. Cartesian closed categories and typed lambda-calculi. In Guy Cousineau, Pierre-Louis Curien, and Bernard Robinet, editors, *Combinators and Functional Programming Languages*, Springer-Verlag, May 1985. LNCS 242.

[45] J. Lambek. *Deductive Systems and Categories II.* Springer-Verlag, 1969. Lecture Notes in Math 86.

[46] J. Lambek. From λ-calculus to cartesian closed categories. In J.P. Seldin and J.R. Hindley, editors, *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, Academic Press, 1980.

[47] J. Lambek and P.J. Scott. *Introduction to higher order categorical logic.* Cambridge University Press, 1986.

[48] F. W. Lawvere. The category of categories as a foundation for mathematics. In *Proceedings of the Conference on Categorical Algebra (La Jolla, 1965)*, pages 1–20, Springer-Verlag, 1966.

[49] F. W. Lawvere. *Functorial Semantics of Algebraic Theories.* PhD thesis, Columbia University, 1963. Announcement in Proc. Nat. Acad. Sci. 50 (1963), pp. 869-873.

[50] F.W. Lawvere. Functional semantics of algebraic theories. *Proceedings of the National Academy of Science*, 50:869–872, 1963.

[51] D. J. Lehmann and M.B. Smyth. Data types (extended abstract). In *Proceedings 18th IEEE Symposium on Foundations of Computer Science*, pages 7–12, 1977.

[52] D.J. Lehmann. On the algebra of order. *Journal of Computer and System Sciences*, 21, 1980.

[53] S. Mac Lane. Sets, topoi, and internal logic in categories. In *Proc. Logic Coll.*, North Holland, Bristol, 1973.

[54] Saunders Mac Lane. *Categories for the Working Mathematician.* Springer-Verlag, 1971.

[55] Saunders Mac Lane. One universe as a foundation for category theory. In S. Mac Lane, editor, *Reports of the Midwest Category Seminar III*, pages 192–200, Springer-Verlag, 1969. Lecture Notes in Mathematics, No. 106.

[56] Saunders Mac Lane and Garrett Birkhoff. *Algebra.* MacMillan, 1967.

[57] E.G. Manes, editor. *Proceedings of the AAAS Symposium on Category Theory Applied to Computation and Control, San Francisco, California*, Springer-Verlag, 1975. LNCS 25.

[58] Ernest Manes and Michael Arbib. *Algebraic Approaches to Program Semantics.* Springer-Verlag, 1986.

[59] Ernest G. Manes. *Algebraic Theories.* Springer-Verlag, 1976. Graduate Texts in Math, volume 26.

[60] A. Melton, D.A. Schmidt, and G.E. Strecker. Galois connections and computer science applications. In David Pitt, Samson Abramsky, Axel Poigné, and David Rydeheard, editors, *Category Theory and Computer Programming*, pages 299–312, Springer-Verlag, September 1985. LNCS 240.

[61] B. Mitchell. *Theory of Categories.* Academic Press, 1965.

[62] Frank J. Oles. *A Category-Theoretic Approach to the Semantics of Programming Languages.* PhD thesis, Syracuse University, 1982.

[63] Frank J. Oles. Type algebras, functor categories, and block structure. In Maurice Nivat and John C. Reynolds, editors, *Algebraic Methods in Semantics*, Cambrige University Press, 1985.

[64] David Pitt, Samson Abramsky, Axel Poigné, and David Rydeheard, editors. *Category Theory and Computer Programming*, Springer-Verlag, September 1985. LNCS 240.

[65] D.H. Pitt, A. Poigne, and D.E. Rydeheard, editors. *Category Theory and Computer Science*, Springer-Verlag, September 1987. LNCS 283.

[66] G.D. Plotkin. A powerdomain construction. *SIAM Journal of Computing*, 5:452–487, 1976.

[67] Gordon Plotkin. Domains. 1980. Lecture notes, Department of Computer Science, University of Edinburgh.

[68] John Reynolds. Semantics as a design tool. Fall 1988. CMU course notes (a previous version was distributed in Spring, 1987).

[69] John Reynolds. Using category theory to design implicit conversions and generic operators. In N.D. Jones, editor, *Proceedings of the Aarhus Workshop on Semantics-Directed Compiler Generation*, Springer-Verlag, January 1980. LNCS 94.

[70] John C. Reynolds. The essence of algol. In de Bakker and van Vliet, editors, *Algorithmic Languages*, pages 345–372, IFIP, North Holland, 1981.

[71] John C. Reynolds. An introduction to the polymorphic lambda calculus. 1988. Introduction to the section on "Polymorphic Lambda Calculus" in "Logical Foundations of Functional Programming, Proceedings of the Year of Programming Institute", edited by Gérard Huet, to be published by Addsion Wesley.

[72] John C. Reynolds. *Notes on a Lattice-Theoretic Approach to the Theory of Computation.* Technical Report, Syracuse University, School of Computer and Information Science, October 1972. Revised March, 1979.

[73] John C. Reynolds. Polymorphism is not set-theoretic. In G. Kahn, D. B. MacQueen, and G. D. Plotkin, editors, *Semantics of Data Types*, pages 145–156, Springer-Verlag, 1984. LNCS 173.

[74] John C. Reynolds. ***Preliminary Design of the Programming Language Forsythe.*** Technical Report CMU-CS-88-159, Carnegie Mellon University, June 1988.

[75] John C. Reynolds and Gordon D. Plotkin. ***On Functors Expressible in the Polymorphic Typed Lambda Calculus.*** Technical Report CMU-CS-88-125, Computer Science Department, Carnegie Mellon University, 1988. Submitted to Information and Computation. This version will also appear in "Logical Foundations of Functional Programming, Proceedings of the Year of Programming Institute," edited by Gérard Huet, to be published by Addsion Wesley.

[76] David Rydeheard. Adjunctions. In David Pitt, Samson Abramsky, Axel Poigné, and David Rydeheard, editors, ***Category Theory and Computer Programming***, pages 53–57, Springer-Verlag, September 1985. LNCS 240.

[77] David Rydeheard. Functors and natural transformations. In David Pitt, Samson Abramsky, Axel Poigné, and David Rydeheard, editors, ***Category Theory and Computer Programming***, pages 43–52, Springer-Verlag, September 1985. LNCS 240.

[78] David E. Rydeheard and Rod M. Burstall. ***Computational Category Theory.*** Prentice Hall, 1988.

[79] David Eric Rydeheard. ***Applications of Category Theory to Programming and Program Specification.*** PhD thesis, University of Edinburgh, 1981. CST-14-81.

[80] D.E. Rydeheard and R.M. Burstall. A categorical unification algorithm. In David Pitt, Samson Abramsky, Axel Poigné, and David Rydeheard, editors, ***Category Theory and Computer Programming***, pages 493–505, Springer-Verlag, September 1985. LNCS 240.

[81] David A. Schmidt. ***Denotational Semantics: A Methodology for Language Development.*** Allyn and Bacon, 1986.

[82] Horst Schubert. ***Categories.*** Springer-Verlag, 1972.

[83] Dana Scott. Continuous lattices. In F.W. Lawvere, editor, ***Toposes, Algebraic Geometry, and Logic***, pages 97–136, Springer-Verlag, 1972. Lecture Notes in Math 274.

[84] Dana Scott. Domains for denotational semantics. In M. Nielson and E.M. Schmidt, editors, ***Automata, Languages, and Programming, 9th Colloquium***, pages 577–613, Springer-Verlag, 1982. LNCS 140.

[85] Dana Scott. ***Lectures on a Mathematical Theory of Computation.*** Technical Report PRG-19, Oxford University, Programming Research Group, May 1981.

[86] Dana Scott. Relating theories of the $\lambda-$calculus. In J.P. Seldin and J.R. Hindley, editors, ***To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism***, Academic Press, 1980.

[87] M.B. Smyth. Effectively given domains. *Theoretical Computer Science*, 5:257–274, 1977.

[88] M.B. Smyth and G.D. Plotkin. The category-theoretic solution of recursive domain equations. *SIAM Journal of Computing*, 11(4):761–783, 1982.

[89] Joseph E. Stoy. *Denotational Semantics: The Scott-Strachey Approach to Programming Language Theory.* MIT Press, 1977.

[90] R.D. Tennent. Functor-category semantics of programming languages and logics. In David Pitt, Samson Abramsky, Axel Poigné, and David Rydeheard, editors, *Category Theory and Computer Programming*, pages 206–224, Springer-Verlag, September 1985. LNCS 240.

[91] James W. Thatcher, Eric G. Wagner, and Jesse B. Wright. Notes on algebraic fundamentals for theoretical computer science. June 1978. Lecture notes from summer on Foundations of Artificial Intelligence and Computer Science, Pisa.

[92] M. Wand. *Fixed-point Constructions in Order-Enriched Categories.* Technical Report 23, Computer Science Department, Indiana University, Bloomington, Indiana, 1977.

[93] M. Wand. Fixed-point constructions in order-enriched categories. *Theoretical Conputer Science*, 8:13–30, 1979.

[94] M. Wand. On recursive specification of data types. In E.G. Manes, editor, *Proceedings of the AAAS Symposium on Category Theory Applied to Computation and Control, San Francisco, California*, Springer-Verlag, 1975. LNCS 25.

[95] J.B. Wright, J.A. Goguen, J.W. Thatcher, and E.G. Wagner. Rational algebraic theories and fixed point solutions. In *Proceedings 17th IEEE Symposium on Foundations of Computer Science*, Houston, Texas, 1976.