Making Discrete Decisions Based on Continuous Values

by

Benjamin Sherman

Submitted to the Department of Electrical Engineering and Computer Science

in partial fulfillment of the requirements for the degree of

Master of Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2017

Author .....

Department of Electrical Engineering and Computer Science

May 19, 2017

Certified by .....

Adam Chlipala

Associate Professor of Electrical Engineering and Computer Science

Thesis Supervisor

Certified by .....

Michael Carbin

Jamieson Career Development Assistant Professor

of Electrical Engineering and Computer Science

Thesis Supervisor

Accepted by .....

Leslie A. Kolodziejski

Professor of Electrical Engineering and Computer Science

Chair, Department Committee on Graduate Theses

Making Discrete Decisions Based on Continuous Values

by

Benjamin Sherman

Submitted to the Department of Electrical Engineering and Computer Science
on May 19, 2017, in partial fulfillment of the
requirements for the degree of
Master of Science

Abstract

Many safety-critical software systems are cyber-physical systems that compute with continuous values; confirming their safety requires guaranteeing the accuracy of their computations. It is impossible for these systems to compute (total and deterministic) discrete computations (e.g., decisions) based on connected input spaces such as $\mathbb{R}$. We propose a programming language based on constructive topology, whose types are spaces and programs are executable continuous maps, that facilitates making formal guarantees of accuracy of computed results. We demonstrate that discrete decisions can be made based on continuous values by permitting nondeterminism. This thesis describes variants of the programming language allowing nondeterminism and/or partiality, and introduces two tools for creating nondeterministic programs on spaces. *Overlapping pattern matching* is a generalization of pattern matching in functional programming, where patterns need not represent decidable predicates and also may overlap, allowing potentially nondeterministic behavior in overlapping regions. *Binary covers*, which are pairs of predicates such that at least one of them holds, yield a formal logic for constructing approximate decision procedures.

Thesis Supervisor: Adam Chlipala
Title: Associate Professor of Electrical Engineering and Computer Science

Thesis Supervisor: Michael Carbin
Title: Jamieson Career Development Assistant Professor
of Electrical Engineering and Computer Science

Acknowledgments

I thank my advisors, Adam Chlipala and Michael Carbin, who contributed to this thesis in many ways, by helping me to focus, justify, and explain my research, and by reviewing drafts of this work and similar ones. I am grateful for their support of my exploration of a subject that is relatively far from their interests.

I was fortunate to work with Luke Sciarappa during the summer of 2016 on related subjects that led us to the subject of this thesis. Luke taught me about categories and toposes, and we worked together to understand constructive topology.

Adam, Michael, Luke, and I submitted a draft on similar material to the Logic in Computer Science (LICS) conference in 2017. We thank the four anonymous reviewers for their helpful feedback. One review was particularly helpful, noting the possibility of using open maps as patterns and noticing Remark 4.17.

Tej Chajed, Gabriel Scherer, and Muralidaran Vijayaraghavan read drafts of this thesis or prior related drafts and provided helpful feedback.

I am grateful for the financial support for my graduate research, which I have received from an MIT SMA fellowship as well as NSF grant CCF-1512611.

I thank my parents for their love and support.

Contents

List of Figures

Chapter 1

Introduction

Many safety-critical software systems are cyber-physical systems that compute with continuous values, such as space, time, magnitude, and probability; confirming their safety requires guaranteeing the accuracy of their computations. We claim that *formal topology*, a constructive theory of topology, provides a suitable theory for building programs that compute with continuous values. We propose a programming language based on formal topology, whose types are spaces and programs are executable continuous maps, that allows execution of programs that manipulate continuous values, while at the same time allowing reasoning in terms of their mathematical descriptions and guaranteeing accuracy of computed results. The interfaces defined in constructive topology give insight into both how to specify such programs and how to compute with continuous spaces [36, 10, 9, 34].

This thesis examines the problem of making discrete decisions based on continuous values from the perspective of constructive topology. By topological arguments, any program $f : \mathbb{R} \to_c \mathbb{B}$ that makes a (total and deterministic) discrete decision based on a real number must be a constant map (i.e., always true or always false). We demonstrate that decisions *can* be made based on continuous values by permitting nondeterminism.

This thesis characterizes two programming language constructs for constructive topology that make it easier to program with continuous values (particularly involving nondeterminism):

- *Overlapping pattern matching* (Chapter 4) generalizes pattern matching on inductive types in functional programming. It differs in that patterns need not correspond to decidable predicates, and patterns are allowed to overlap; an input satisfying multiple patterns may nondeterministically follow any such branch. Because spaces often have few decidable predicates, nondeterminism is often essential for decision-making, and overlapping pattern matching facilitates construction of such programs.

- *Binary covers* (Chapter 5), which are pairs of opens that cover a space, provide an expressive formal logic of "approximately decidable" properties. Most significantly, the logic admits quantification over compact spaces, demonstrating

"approximate decidability" of questions such as whether a real-valued function on a compact space has any roots.

We intend to explain in the remainder of this chapter how this thesis helps to make sense of making decisions based on continuous values, and how the above techniques make it easier to devise such programs. Section 1.1 describes how a rather general approximate-root finding function can be written with the above techniques. Section 1.2 justifies the choice of formal topology as the appropriate formalism for programming with continuous values, and 1.3 explains informally why it is often difficult to make decisions based on continuous values, and why nondeterminism may be necessary.

1.1 Example application: approximate root-finding

Consider the following computational task. Suppose we have an arbitrary continuous function $f : K \to_c \mathbb{R}$, where K is some compact space (such as the unit interval $[0, 1]$), and want to determine if it has any roots. In general, it is not decidable whether f has any roots [36], but an approximation is possible: fix a tolerance $\varepsilon > 0$. Then either, or both, of the following statements must hold:

- There is some $x \in K$ such that $|f(x)| < \varepsilon$.
- For every $x \in K$, $f(x) \neq 0$.

Though each of the above statements is in general undecidable, we can use the techniques devised in this thesis to define a function that nondeterministically computes that one of the above two statements holds, in the former case *computing* such an x that is "almost a root." We will observe in Section 5.4 that the pair of bulleted statements in fact comprise a binary cover, and that an overlapping pattern match can be used to construct the program that solves the task.

1.2 Why formal topology?

This section motivates formal topology as a language theory for programming with continuous values, explaining some desirable properties of such a theory and how these properties are absent in alternative methods of specifying and computing with continuous values.

1.2.1 Reasoning about programs as mathematical expressions

A principal desire is to be able to soundly reason about the behavior of programs that manipulate continuous values as the mathematical expressions that they denote. For instance, we might hope that the real-valued programs $(1 + x) - x$ and 1 would be considered equivalent (since the corresponding real-valued mathematical expressions are equivalent).

Notably, floating point lacks this property: in particular, those two programs are not considered equivalent, and for the value $x \triangleq 10^{50}$, for instance, the expression $(1 + x) - x$ returns 0 rather than 1.

Systems lacking such a connection between the programs and the mathematical expressions they are supposed to represent are difficult to program with: a programmer cannot necessarily use a mathematical identity of real-valued expressions to replace one program with another (which may be faster, simpler, or otherwise more efficient) without worrying that it might change the program's behavior. Similarly, compilers cannot use mathematical identities, either, for the purpose of optimization.

Comparison with floating point

A common approach for computing with real numbers is to use floating-point numbers as a surrogate. Floating point is intended behave like the real numbers: for instance, most floating-point operations return the floating-point value closest to the "ideal" real number result. Unfortunately, this guarantee does not compose well: it is easy to conceive floating-point programs whose results are arbitrarily erroneous compared to the ideal real-valued result. For instance, consider the real-valued function [31, Chapter 9.1]

$$f : \{x : \mathbb{R} \mid x \neq 0\} \to \mathbb{R}$$
$$f(x) \triangleq \frac{1 - \cos(x)}{x^2},$$

and its double-precision floating-point approximation using a cosine operation that returns an answer to within 1 ULP of the ideal answer. Then $f(1.1 \times 10^{-8})$ is very close to 1/2 (easily within 10^{-8}), while the floating-point code returns approximately 0.9175. In practice, it is very difficult to bound the error of floating-point functions.

Issues such as overflow, roundoff error, and catastrophic cancellation make it very difficult to reason about the behavior of floating-point programs. Mathematical properties of real numbers do not necessarily hold for floating-point programs. For instance, addition/multiplication of floating-point values is not associative. Accordingly, replacing one floating-point program with another one which is equivalent when viewed as an expression on real numbers can drastically change behavior. Moreover, there may not be a single such program with the least error for all inputs [27].

The lack of the connection between floating point and real numbers makes programming with floating point a difficult task. A programmer cannot necessarily use a mathematical identity of real-valued expressions to replace a floating-point program with another (which may be faster, simpler, or otherwise more efficient) without worrying that it might negatively impact its accuracy. Nor can programmers (without serious additional verification effort) be reasonably assured that their floating-point program is at all accurate.

Compilers also face difficulty in handling floating-point programs. Since the source language of a compiler will generally reflect all details about the low-level floating point, compilers must preserve *exact* behavior of floating-point programs, even

though often the programmer does not rely on such exactness. Accordingly, compilers are consigned to perform only very limited optimization of floating-point code that doesn't change floating point behavior. For instance, they cannot rewrite a program $(1 + x) - x$ to the constant 1 because floating-point addition is not associative.

So floating point unfortunately lacks a key desideratum for a theory of programming with continuous values.

1.2.2 Computational content of compactness

There are several computational frameworks for computing with real numbers that *do* allow reasoning about programs as mathematical expressions. Most can be labeled a form of *exact real arithmetic*, meaning that terms in their language faithfully represent real numbers themselves. A result from computability theory regarding exact real arithmetic underscores the fundamental importance of topology:

Theorem 1.1. *Any computable function*[1] $f : \mathbb{R} \to \mathbb{R}$ *is necessarily continuous [49].*

So one is not really missing out by restricting attention to continuous maps.

Constructive topology also has an advantage over other frameworks for exact real arithmetic: the computational content in its definition of compactness. *Compactness* is a fundamental topological property of spaces that generalizes the concept of (Kuratowski) finiteness of sets. Compactness is important for many results in analysis. One such result is that a real-valued continuous function on a compact space has a minimum value. Unlike other frameworks for exact real arithmetic, constructive topology allows this value to be computed.

Comparison with Bishop-style constructive analysis

One way to approach programming with real numbers is with Bishop-style constructive analysis, where one develops the theory of metric spaces in a constructive manner. The c-CoRN library in fact implements Bishop-style constructive analysis within Coq.

However, there are known difficulties with Bishop-style analysis. For instance, the notions of continuity, compactness, and subspaces do not interact well, unless one accepts additional axioms[2]. In particular, Waaldijk [48] proves that in Bishop-style analysis, if one desires certain basic properties of continuous maps to hold, then one must accept an axiom called the fan theorem.

1.3 Discrete decision-making on connected spaces

Lamport makes a bold claim with severe implications [18]:

[1] In the theory of type II computability.

[2] It is theoretically possible to give a computational model of type theory admitting such axioms via Type II computability theory [2]. However, to compile systems like Coq to such a computational model would be a difficult task.

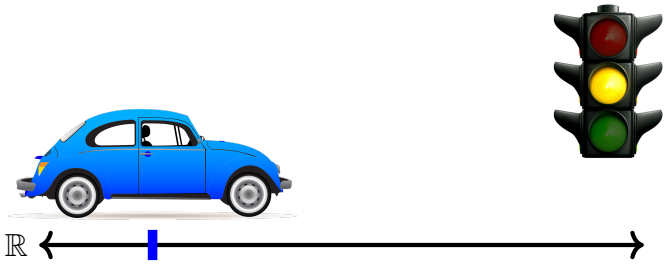


Figure 1-1: An autonomous car approaches a traffic light that has just turned yellow, and must decide whether to stop before the intersection or proceed through it.

> *Buridan's Principle.* A discrete decision based upon an input having a continuous range of values cannot be made in a bounded length of time.

Lamport's name "Buridan's Principle" for the above claim comes from the famous philosophical problem of Buridan's Ass, which has many variations, but in one form finds a donkey unable to decide between two equally distant and equally delicious bails of hay, finally starving to death.

Why must it be so? Lamport argues that ultimately the decision arises by physical mechanisms, and physics is necessarily continuous. The same argument could be made from the perspective of computability, given that any computable function also is necessarily continuous (theorem 1.1). Buridan's principle can be rephrased as the topological theorem:

Theorem 1.2. *Any continuous map $f : A \to_c \mathbb{B}$ from a connected space A to the Booleans is necessarily constant.*

Since the connected spaces include those such as $\mathbb{R}$ and $\mathbb{R}^n$, this theorem precludes the possibility of computing total, deterministic decisions based on real-valued inputs.

Consider an analogous situation, depicted in Figure 1.3, in which an autonomous car approaches a yellow light and must decide whether to stop before the intersection or proceed through it. How can we program the car to safely navigate the intersection, given the impossibility result of Theorem 1.2?

This thesis argues that *nondeterminism* makes it possible to make decisions such as these in bounded time, providing a loophole to work around Buridan's Principle. We claim that in each case there is some (open) region, perhaps small, where both decisions may be made, perhaps from the *exact same* input. In the case of floating point, though a floating-point comparison operation is indeed deterministic, there is effectively nondeterminism in the way measurements are approximated to floating point, and in the rounding in floating-point arithmetic operations.

This thesis explores how to program nondeterministic functions on spaces, developing programming-language constructions for computing approximate decisions that make it easy to reason about what behaviors may occur.

1.4 Terminology and foundations

We intend mathematical statements to be interpreted within a constructive and predicative metatheory, potentially formalizable within, for instance, the predicative fragment of Coq (i.e., without use of the `Prop` universe).

Since all topological notions in this article are pointfree, we will coopt terminology from classical topology without fear of confusion. For instance, we use the word "space" rather than "locale" when describing the pointfree analogue of spaces. Because we want to *compute* with the mathematical structures, we use the propositions-as-types correspondence [30] to encode all propositions as types, such that everything is formalizable within Martin-Löf type theory (as well as the predicative fragment of Coq). Let $\mathcal{U}$ denote the universe of types, where the universe level is left implicit, but we use "small" and "large" to relative sizes of universes.

We use the term "type" to refer to a type, and the term "set" to refer to what is often called a setoid or a Bishop set: a type together with a distinguished equivalence relation on it. We use the symbol $\equiv$ to denote intensional equality on members of any type, reserving $=$ for the equality relation on sets. The reader should assume that if A and B are both sets, then the notation $f : A \to B$ means that f is a morphism in the category of sets (i.e., it maps equivalent elements of A to equivalent elements of B).

Let Ω the (large) set of propositions: $\mathcal{U}$ endowed with the equivalence relation of bi-implication. Ω defines a completely distributive lattice with small meets and joins, and we use lattice notation ($\top$, $\bot$, $\wedge$, $\vee$, $\leq$) accordingly. Propositional functions $P : A \to \Omega$ represent the subsets of A, and for $a : A$ we use the familiar notation $a \in P$ to denote the proposition $P(a) : \Omega$. We use $\cup$ and $\cap$ to denote union and intersection of subsets, respectively.

Chapter 2

Constructive topology

This chapter introduces formal topology as a theory of computation with continuous values. We will develop the idea using the real numbers (or metric spaces, more generally) as an example. We will identify the computational interface that the real numbers provide (that is, what observations we are allowed to make on real numbers) in terms of their "ideal" (implementation-independent) observable properties, and discover that this interface has the logical structure of a *formal topology* and is shared by all formal spaces.

2.1 Introduction via the real numbers

This section introduces formal topology via the example of the real numbers, $\mathbb{R}$. We begin by defining the real numbers by more conventional means (as Cauchy approximations) and attempting to define a program that makes a nondeterministic decision based on a real number in this framework. It is difficult to reason spatially about which real numbers may be mapped to true or false in this way, because one must reason about the many possible implementations of a given real number. We then adjust the definition of the real numbers so that they can be specified in an implementation-independent way, which leads to their definition as a formal space. In this framework, we first *specify* how the nondeterministic behavior should behave spatially, and then derive an *implementation* (formal proof) that computes the behavior. Because the specification completely describes the desired behavior, one need not reason about the form of the implementation beyond the fact that some implementation exists.

The real numbers can be viewed as arbitrarily fine rational-valued approximations to some well-defined quantity. For instance, given a square with sides of length 1, the diagonal has length $\sqrt{2}$, which is a real number but not a rational number. That is, one can produce rational numbers which come ever closer to the length of the diagonal, though no rational number alone suffices to characterize it. The ability to produce arbitrarily fine approximations is a useful and well-behaved one: it is this idea that the real numbers encapsulate.

One standard construction of $\mathbb{R}$ is as the set of Cauchy sequences of rational numbers. A sequence $x : \mathbb{N} \to \mathbb{Q}$ is *Cauchy* if for all $\varepsilon : \mathbb{Q}^+$, there is some $N : \mathbb{N}$

such that for all $m, n : \mathbb{N}$ where $N \leq n$ an $N \leq m$, we have $d(x_m, x_n) < \varepsilon$, where $d : \mathbb{Q} \times \mathbb{Q} \to \mathbb{Q}$ denotes the distance between two rational numbers. Two sequences x and y are equivalent if the distance between their approximations $d(x_n, y_n)$ tends to 0 as n approaches ∞.

This standard definition can be reduced to a simpler one [24]. We can instead define *regular functions* $\mathsf{RegFunc}$, functions $x : \mathbb{Q}^+ \to \mathbb{Q}$ such that for all $\delta, \varepsilon : \mathbb{Q}^+$, $d(x_\delta, x_\varepsilon) \leq \delta + \varepsilon$. Intuitively, we imagine that x_δ is at most a distance δ away from the "true" (but unrepresentable as a rational number) value of x, such that $d(x_\delta, x_\varepsilon) \leq \delta + \varepsilon$ represents that each approximation is consistent with each other in approximating the same limiting value, by imagining a notional application of the triangle inequality,

$$d(x_\delta, x_\varepsilon) \leq d(x_\delta, x) + d(x, x_\varepsilon) \leq \delta + \varepsilon.$$

The relational characterization of consistent approximations of a real number also suffices to characterize when two regular functions implement the same real number: the real numbers $\mathbb{R}$ are the regular functions where two regular functions x and y are considered equivalent if for all $\delta, \varepsilon : \mathbb{Q}^+$, $d(x_\delta, y_\varepsilon) \leq \delta + \varepsilon$. For a regular function $x : \mathsf{RegFunc}$, let $[x] : \mathbb{R}$ be the real number that it implements.

Suppose we want to approximately compute whether a given real number is positive. It is impossible to do so exactly, so suppose we pick some error tolerance $\varepsilon : \mathbb{Q}^+$, and we wish to define a function $f : \mathsf{RegFunc} \to \mathbb{B}$ want the following behavior on real numbers:

1. If $f(x) = \mathsf{true}$ then $[x] > 0$.
2. If $f(x) = \mathsf{false}$, then $[x] \leq \varepsilon$.

We observe that for any $\delta : \mathbb{Q}^+$, $[x] - \delta \leq x_\delta \leq [x] + \delta$, so with a bit of reasoning we find that a valid implementation of f is $f(x) \triangleq x_{\varepsilon/2} > \varepsilon/2$ (recalling that comparison of rational numbers is decidable). It is far from obvious that this definition of f satisfies the expectations defined above, because one must reason about all implementations of a given real number.

This is an issue with the definition of $\mathbb{R}$, as it does not clearly describe what a real number *is* independently of some implementation of that real number as a regular function, so it is unclear what are properties of real numbers rather than of particular regular function. We will tease out those properties that suffice to characterize a real number that

1. are implementation-independent and
2. can be observed by querying *any* implementation (meaning that if the property holds, we can observe it to hold by interacting with the implementation).

We call these properties (subsets) *open*.

Such a characterization would allow us to decouple a real number into a specification that says which of these such properties a real number satisfies, and an implementation which allows computational observation of *only* those properties (and no

additional information), such that we could not care *which* implementation we are given, only that we are given any implementation at all (which also serves as a model to confirm that the specification is consistent).

Examples of open properties of $\mathbb{R}$ are $\cdot > 0$ (i.e., positivity) and $\cdot < \varepsilon$. The property $\cdot \leq \varepsilon$ is not open; its interior $\cdot < \varepsilon$ is the largest open included within it (every subset has an interior). Once we have characterized $\mathbb{R}$ as a formal space, defining the approximate comparison function will reduce to simply finding a formal proof $\top \leq (\cdot > 0) \vee (\cdot < \varepsilon)$ that every number is either positive or less than ε [1].

The regular functions come close to explaining $\mathbb{R}$ in the terms as above, but don't quite work. When a regular function $x : \mathbb{Q}^+ \to \mathbb{Q}$ is queried for some tolerance $\varepsilon : \mathbb{Q}^+$ to produce an answer $x_\varepsilon = q$, we learn that $d([x], q) \leq \varepsilon$, but we may not necessarily learn this same fact from a different representative of the same real number. For instance, we can define the regular functions 0^+ and 0^- by $0^+_\varepsilon \triangleq \varepsilon$ and $0^-_\varepsilon \triangleq -\varepsilon$. Both of these regular functions implement the real number 0, from 0^+ we easily learn that 0 is nonnegative, which we can never learn by observing 0^-. So these types of observations are not necessarily observable given any implementation.

We can ensure that the regular functions only produce observable properties by making a small change. Whenever a regular function $x : \mathsf{RegFunc}$ allows us to observe that $d([x], q) \leq \varepsilon$, we expect that it should then be able to make tighten this estimate, that is, it should be able to produce a $\varepsilon' > 0$ such that $\varepsilon' < \varepsilon$ but still $d([x], q) \leq \varepsilon'$. Therefore, any estimate that says $d([x], q) \leq \varepsilon$ in fact tells us that x is *strictly less than* ε away from q.

We will claim that after making this adjustment, any observation made with one implementation can be made with another. First, let's make the adjustment precise. To allow an implementation x of a real number $[x]$ to make previous approximations more precise, we need to make a datatype representing previous approximations. For each $q : \mathbb{Q}$ and $\varepsilon : \mathbb{Q}^+$, let $[x] \Vdash B_\varepsilon(q)$ denote the type of evidence that $[x]$ lies in the open ball of radius ε around q (or equivalently, $d(x, q) < \varepsilon$). Then any implementation x offers two ways by which we can learn more about $[x]$,

$$\frac{\varepsilon : \mathbb{Q}^+}{\sum_{q:\mathbb{Q}} [x] \Vdash B_\varepsilon(q)}\ \textsc{approx} \qquad \frac{[x] \Vdash B_\varepsilon(q)}{\sum_{\varepsilon':\mathbb{Q}^+ \mid \varepsilon' < \varepsilon} [x] \Vdash B_{\varepsilon'}(q)}\ \textsc{shrink},$$

as well as a rule that allows us to "forget" information,

$$\frac{[x] \Vdash B_\varepsilon(q) \qquad B_\varepsilon(q) \subseteq B_\delta(r)}{[x] \Vdash B_\delta(r)}\ \textsc{weaken}.$$

Proposition 2.1. *If $[x] = [y]$, then if $[x] \Vdash B_\varepsilon(q)$, we can observe the same of y by applications of the* APPROX *and* SHRINK *rules.*

Proof sketch. By SHRINK, there is some $\varepsilon' : \mathbb{Q}^+$ such that $\varepsilon' < \varepsilon$ and $[x] \Vdash B_{\varepsilon'}(q)$.

[1] It will satisfy an even tighter specification, that if $f(x) = \mathsf{false}$, then $[x]$ is *strictly* less than ε.

Define $\delta : \mathbb{Q}^+$ by $\delta \triangleq \varepsilon - \varepsilon'$. Then by APPROX there is some $r : \mathbb{Q}$ such that $y \Vdash B_\delta(r)$. Since $[x] = [y]$, $B_{\varepsilon'}(q)$ and $B_\delta(r)$ must overlap, and therefore by WEAKEN, $y \Vdash B_\varepsilon(q)$. □

Since any open subset is a union of open balls, we can define for any open subset U and any implementation x of a real number, the type of evidence $[x] \Vdash U$ that $[x]$ lies in U as

$$[x] \Vdash U \triangleq \bigcup \{[x] \Vdash B_\varepsilon(q) \mid B_\varepsilon(q) \subseteq U\}.$$

We can now specify the task of making the nondeterministic decision as finding a function that for all implementations x of real numbers computes

$$([x] \Vdash \cdot > 0) + ([x] \Vdash \cdot < \varepsilon)\,,$$

and returns true in the case of inl and false in the case of inr. Because of the proof terms that such a function returns, it would be guaranteed to behave as was originally desired. The derivation of the above function would mirror the definition using regular functions: we would first use APPROX$(\varepsilon/2)$ to determine $\sum_{q:\mathbb{Q}} [x] \Vdash B_\varepsilon(q)$, and check to see of the result whether $q > \varepsilon/2$, and then use the WEAKEN rule to weaken to $[x] \Vdash \cdot > 0$ if $q > \varepsilon/2$ and $[x] \Vdash \cdot < \varepsilon$ if $q \leq \varepsilon/2$.

This shift succeeds in isolating specifications of real numbers from their implementations. To specify a real number, it suffices to say what balls $B_\varepsilon(q)$ it lies in. The computational rules of an implementation, e.g. APPROX and SHRINK, explicitly indicate what the results of the computation mean in terms of the observable properties of the real number that the implementation represents.

The above characterization is informal and incomplete; the remainder of this chapter will give a rigorous description of formal topology, and in particular Section 2.3.6 will characterize metric spaces. The basic idea in constructing $\mathbb{R}$ as a formal space is to first describe a logic of the balls $B_\varepsilon(q)$ that completely describes which balls are covered by unions of other balls. A point is specified by the formal balls that it lies in, and an implementation of a real number allows one to *compute* which formal balls that it lies in, which also serves to guarantee the soundness of the specification of that real number.

2.2 Formal topology

In this section, we formally describe formal spaces, and then the inductively generated formal spaces, which will form a category **FSpc**.

Definition 2.2. *A* formal space A *is a preorder* $\mathcal{O}_{\mathsf{B}}(A)$ *together with a relation* $\lhd : \mathcal{O}_{\mathsf{B}}(A) \to (\mathcal{O}_{\mathsf{B}}(A) \to \Omega) \to \Omega$ *that satisfies the rules in Figure 2-1.*

The intuitive meaning of this structure is that $\mathcal{O}_{\mathsf{B}}(A)$ is a set of basic opens on A, where $a \leq_{\mathcal{O}_{\mathsf{B}}(A)} b$ if the basic open a is included in the basic open b. A subset $U : \mathcal{O}_{\mathsf{B}}(A) \to \Omega$ represents an arbitrary union of the basic opens in U. This means that every open of A can be represented as a subset $U : \mathcal{O}_{\mathsf{B}}(A) \to \Omega$. Then $a \lhd U$

$$\textsc{refl}\ \frac{a \in U}{a \lhd U} \qquad \textsc{trans}\ \frac{a \lhd U \qquad U \lhd V}{a \lhd V} \qquad \leq\textsc{-left}\ \frac{a \leq b \qquad b \lhd U}{a \lhd U}$$

$$\leq\textsc{-right}\ \frac{a \lhd U \qquad a \lhd V}{a \lhd U \downarrow V}$$

Figure 2-1: Rules that a formal cover relation must satisfy.

represents the proposition that the basic open a is covered by the open U (that is, the extent of a is included in the extent of U, though it need not be the case that directly $a \in U$).

This can be extended to specify the entire covering relation, that is, when an open U is covered by another open V. We define (overloading the notation $\lhd$ to have either basic opens or opens on the left side)

$$U \lhd V \triangleq \forall a \in U.\ a \lhd V,$$

which says that U is covered by V if all of the basic opens comprising U are covered by V. With this definition, we can define the (large) set of opens on A, $\mathcal{O}(A)$, as the type $\mathcal{O}_{\mathsf{B}}(A) \to \Omega$ where two opens $U, V : \mathcal{O}_{\mathsf{B}}(A) \to \Omega$ are equivalent if they each cover each other, i.e., $U \lhd V$ and $V \lhd U$.

The intersection of two basic opens a and b need not necessarily be a basic open itself; however, it must necessarily be the union of basic opens c which are included in both a and b. Therefore, we define the operator[2] [3]

$$\downarrow : \mathcal{O}_{\mathsf{B}}(A) \to \mathcal{O}_{\mathsf{B}}(A) \to (\mathcal{O}_{\mathsf{B}}(A) \to \Omega)$$
$$a \downarrow b \triangleq \bigcup \{c : A \mid c \leq a \text{ and } c \leq b\}.$$

The $\downarrow$ operator can be extended to apply to opens just by taking unions, i.e.,

$$U \downarrow V \triangleq \bigcup_{a \in U} \bigcup_{b \in V} a \downarrow b.$$

Having defined $\downarrow$, we can return to understand the rules on a formal cover described in Figure 2-1. REFL says that a basic open is covered by an open that includes that basic open. $\leq$-LEFT is similar, saying that if U covers a basic open b, then it covers a basic open a that is included in b. TRANS describes transitivity of the cover relation: if a is covered by U, and U is covered by V, then a is covered by V. $\leq$-RIGHT says

[2] For any type A and $x : A$, the notation $\{x\}$ denotes the singleton subset, $\lambda y.\ x \equiv y$.

[3] There are many analogies between the construction of the topos of sheaves on a site and the construction of the lattice of opens of a formal space. The definition of $\downarrow$ operator is just a special case of the construction of the product of representable presheaves, where the presheaves are over the category $\mathcal{O}_{\mathsf{B}}(A)$ (considering a preorder as a category). We will refrain from pointing out the other various correspondences.

$$\frac{x \Vdash a \qquad a \lhd U}{x \Vdash U}\ \textsc{split} \qquad \frac{}{x \Vdash \top}\ \textsc{meet-0} \qquad \frac{x \Vdash a \qquad x \Vdash b}{x \Vdash a \downarrow b}\ \textsc{meet-2}$$

Figure 2-2: Rules that a point of a formal space must satisfy.

that if a basic open a is covered by two opens U and V, it must be covered by their intersection.

We should emphasize that these definitions and rules are purely formal: basic opens are just "symbols" in some sense, rather than subsets as they are in classical topology. The rules should match spatial intuition but are not justified by it. The rules can be understood from a *computational* perspective as well, which is most apparent when considering the definition of a point in a formal space.

Definition 2.3. *A* point *x of a formal space A is a subset $(x \Vdash \cdot) : \mathcal{O}_{\mathsf{B}}(A) \to \Omega$ (read "x lies in") satisfying the rules in Figure 2-2. The points of A form a set $\mathsf{Pt}(A)$ where two points x and y are considered equivalent if for all $a : \mathcal{O}_{\mathsf{B}}(A)$, $x \Vdash a$ if and only if $y \Vdash a$.*

As with the $\downarrow$ operator, we extend $x \Vdash \cdot$ to operate on opens, rather than just basic opens, by taking a union, [4]

$$x \Vdash U \triangleq \exists a \in U.\ x \Vdash a.$$

The symbol $\top : \mathcal{O}(A)$ denotes the trivially true propositional function (i.e., subset that is the whole set), and denotes the open that represents the entire space.

Intuitively, $x \Vdash a$ means that the point x lies in the basic open a. Points are described by which opens they lie in, but not every collection of basic opens actually specifies a point. The SPLIT rule says that if x lies in a and if a is covered by U, the x lies in U. This can also be read computationally. If $x \Vdash a$, and we provide a cover U of a (imagine U is very fine), then x *computes* some basic open $b \in U$ such that $x \Vdash b$. We call this computation *splitting* a point with an open cover (following Palmgren [26]). MEET-0 says that a point must lie in the entire space, and MEET-2 says that if x lies in both U and V, then it must lie in their intersection. Another intuitive interpretation is that SPLIT allows one to refine information about a point, MEET-0 gives us *some* information about the point (so there is at least something to refine), and MEET-2 allows us to combine two pieces of information into one.

Definition 2.4. *The one-point space $*$ is defined by taking $\mathcal{O}_{\mathsf{B}}(*) \triangleq *$, where $*$ also denotes the preorder with one element tt, and defining the cover relation*

$$\mathsf{tt} \lhd U \triangleq \mathsf{tt} \in U.$$

[4] The existential quantifier in this definition should be viewed as a dependent pair, where it is possible to *compute* the basic open that exists. This definition agrees with formal topology as formalized in Martin-Löf type theory but differs from Maietti and Sambin [21], who use an existential quantifier that does not guarantee a computational interpretation.

$$\frac{a \lhd U}{f^*(a) \lhd f^*(U)}\ \textsc{split} \qquad \frac{}{\top \lhd f^*(\top)}\ \textsc{meet-0} \qquad \frac{}{f^*(a) \downarrow f^*(b) \lhd f^*(a \downarrow b)}\ \textsc{meet-2}\ .$$

Figure 2-3: Rules that a continuous map of formal spaces must satisfy.

One can confirm that this covering relation satisfies the necessary rules. Then $\mathcal{O}(*) \cong \Omega$, and therefore we can interpret the subset $(x \Vdash \cdot) : \mathcal{O}_{\mathsf{B}}(A) \to \Omega$ corresponding to a point x of a space A also as an inverse image map $x^* : \mathcal{O}_{\mathsf{B}}(A) \to \mathcal{O}(*)$. In $\mathcal{O}(*)$, the $\downarrow$ operator corresponds to conjunction of propositions, and $\lhd$ to implication. We can use this correspondence to generalize the definition of points of a space to continuous maps from one space to another:

Definition 2.5. *A* continuous map *f from a formal space Γ to a formal space A, written $f : \Gamma \to_c A$, is a map $f^* : \mathcal{O}_{\mathsf{B}}(A) \to \mathcal{O}(\Gamma)$ satisfying the rules in Figure 2-3. The continuous maps from Γ to A form a set $\Gamma \to_c A$ where two maps f and g are considered equivalent if for all $a : \mathcal{O}_{\mathsf{B}}(A)$, $f^*(a) = g^*(a)$.*

2.2.1 Inductively generated formal spaces

Unfortunately, in the general case, it seems impossible to form product spaces for those formal spaces that have no structure beyond their satisfaction of the laws in Figure 2-1. This motivates the definition of *inductively generated formal spaces* by Coquand et al. [7]. This will be the main category that we work with in this thesis. As a category, the inductively generated formal spaces, **FSpc**, have products and form a full subcategory of the more general class of formal spaces.

For inductively generated formal spaces, the cover relation takes a particular form: it is generated by an indexed family of axioms of the form $a \lhd U$ for concrete as and Us.

Definition 2.6. *A formal space A is* inductively generated *if for each $a : \mathcal{O}_{\mathsf{B}}(A)$ there is an index type $I_a : \mathcal{U}$ indexing a family of opens $C_a : I_a \to (\mathcal{O}_{\mathsf{B}}(A) \to \Omega)$ such that the covering relation is equivalent to the one inductively generated by the following constructors (which always satisfy the rules in Figure 2-1):*

$$\frac{a \in U}{a \lhd U}\ \textsc{refl} \qquad \frac{a \le b \qquad b \lhd U}{a \lhd U}\ \le\textsc{-left}$$

$$\frac{i : I_b \qquad a \le b \qquad a \downarrow C_b(i) \lhd U}{a \lhd U}\ \textsc{cov-axiom}\ .$$

The covering relation generated by an axiom set (I, C) is the *least* covering relation satisfying $a \lhd C_a(i)$ for all $a : \mathcal{O}_{\mathsf{B}}(A)$ and $i : I_a$ [7]. The COV-AXIOM rule at once ensures that the required covers are present and that TRANS and $\le$-RIGHT hold.

Most importantly, inductively generated formal spaces have products [7] and pullbacks [16] (whereas formal spaces in general may not). Coquand et al. and Vickers demonstrate inductive generation of all spaces used in this thesis [7, 41, 40, 43, 45]. These constructions are critical in enabling computation over these spaces.

2.2.2 Locales

Formal topology describes spaces with a particular base of opens, but often it is easier to express certain constructions instead with *locale theory*, where a space is described without reference to a particular base. Every formal space determines a locale, but (predicatively) one cannot in general construct a formal space from a locale.

There are really two key reasons (which are related) why we deal with formal topology at all, rather than exclusively using locales:

- Formal topology is more concrete and, in practice, is useful for giving concrete methods of constructing spaces (similar to the use of generators and relations for presenting locales).
- Predicatively, a general construction of product spaces and pullbacks does not appear to be possible for locales [7]. These can only be constructed for (inductively generated) formal spaces.

Once a space has been constructed using formal topology, it is sometimes convenient to shift to the language of locale theory, where no fundamental distinction is drawn between the basic opens and opens in general.

Definition 2.7. *A* locale *A is a distributive lattice $\mathcal{O}(A)$ that has top and bottom elements, $\top$ and $\bot$, respectively, and that has small joins such that meets distribute over joins:* [5]

$$a \wedge \bigvee_{i:I} b_i = \bigvee_{i:I} a \wedge b_i.$$

Theorem 2.8. *For any formal space A, the preorder $\mathcal{O}(A)$ of opens (defined in Section 2.2) yields a locale.*

Proof sketch. We need to show $\mathcal{O}(A)$ has the requisite operations. We define

$$\begin{aligned} \top &\triangleq \lambda\_.\ \top \\ \bot &\triangleq \lambda\_.\ \bot \\ U \wedge V &\triangleq U \downarrow V \\ \bigvee_{i:I} U_i &\triangleq \bigcup_{i:I} U_i. \end{aligned}$$

[5] Having small joins means having I-indexed joins for any small type $I : \mathcal{U}$. Any lattice of opens $\mathcal{O}(A)$ where $\top \neq \bot$ is necessarily large. Since locale theory is generally impredicative, care must be taken with defining the predicative analogue presented here. Palmgren [25] offers a more careful treatment of predicativity and universes in formal topology.

One can confirm that these operations are well-defined (in fact, monotone) with respect to the preorder on $\mathcal{O}(A)$ and that they satisfy the requisite algebraic laws. $\square$

We call the lattice $\mathcal{O}(A)$ the *opens* of A, which describes the observable or "affirmable" properties of A [39]. If $U \leq \bigvee_{i:I} V_i$, we call the family $(V_i)_{i:I}$ an *open cover* of U.

We next define the points and continuous maps, which are closely analogous to the version for formal spaces.

Definition 2.9. *A* point *x of a space A is a subset $x \Vdash \cdot : \mathcal{O}(A) \to \mathcal{U}$ (read "x lies in") such that*

$$\frac{x \Vdash U \qquad U \leq \bigvee_{i:I} V_i}{\exists i : I.\ x \Vdash V_i}\ \textsc{split} \qquad \frac{}{x \Vdash \top}\ \textsc{meet-0} \qquad \frac{x \Vdash U \qquad x \Vdash V}{x \Vdash U \wedge V}\ \textsc{meet-2}\ .$$

By the same reasoning as with formal spaces, we can rephrase these rules so that they suggest the rules for continuous maps:

$$\frac{U \leq \bigvee_{i:I} V_i}{x^*(U) \leq x^*\left(\bigvee_{i:I} V_i\right)}\ \textsc{split}$$

$$\frac{}{\top \leq x^*(\top)}\ \textsc{meet-0} \qquad \frac{}{x^*(U) \wedge x^*(V) \leq x^*(U \wedge V)}\ \textsc{meet-2}\ .$$

However, since SPLIT implies that x^* preserves small joins and is monotone, this means that the reversed inequalities in the MEET rules must hold, and so we can simplify these rules to a simpler and more algebraic definition:

Definition 2.10. *A* continuous map *$f : A \to_c B$ between locales is a map $f^* : \mathcal{O}(B) \to \mathcal{O}(A)$ that preserves $\top$, binary meets, and small joins. The map f^* is known as the* inverse image map*.*

A continuous map $f : A \to_c B$ transforms covers on B into covers on A. Locales and continuous maps form a category.

Definition 2.11. *Two spaces A and B are* homeomorphic*, written $A \cong B$, if they are isomorphic in the category of continuous maps, that is, if there are continuous maps $f : A \to_c B$ and $g : B \to_c A$ such that $g \circ f = \mathsf{id}_A$ as well as $f \circ g = \mathsf{id}_B$.*

Definition 2.12 (Specialization order)**.** *For two maps $f, g : A \to_c B$, define $f \leq g$ if for every $U : \mathcal{O}(B)$, $f^*(U) \leq g^*(U)$.*

Whenever it is the case that $f \leq g$,g can always behave as f. For instance, if f and g are points, then given an open cover $\top \leq \bigvee_{i:I} U_i$, there is some $i^* : I$ such that $f \Vdash U_{i^*}$. If $f \leq g$, then $g \Vdash U_{i^*}$ as well.

Definition 2.13. *An open $U : \mathcal{O}(A)$ is* positive*, written $\mathsf{Pos}(U)$ if every open cover of U is nonempty, i.e., if for every small family $(V_i)_{i:I}$ such that $U \leq \bigvee_{i:I} V_i$, I is inhabited.*

Positivity encodes the notion of being "strictly bigger than $\bot$," and assuming classical logic, $\mathsf{Pos}(U)$ is equivalent to $\neg(U \leq \bot)$.

Proposition 2.14. *If $f : \mathcal{O}(A) \to \mathcal{O}(B)$ preserves joins, then for $U : \mathcal{O}(B)$, $\mathsf{Pos}(f(U))$ implies $\mathsf{Pos}(U)$.*

Proof. Suppose $U \leq \bigvee_{i:I} V_i$. Since f preserves joins, it is monotone, so

$$f(U) \leq f\left(\bigvee_{i:I} V_i\right) = \bigvee_{i:I} f(V_i).$$

Since $f(U)$ is positive, I is inhabited by the above cover, and thus U is positive. □

2.2.3 The computational content of formal topology

A continuous map $f : \Gamma \to_c A$ (with a point of a space a special case, where $\Gamma \cong *$) is defined by two pieces of data: its inverse image map $f^* : \mathcal{O}(A) \to \mathcal{O}(\Gamma)$ and a formal proof that f^* preserves small joins, $\top$, and binary meets. It is reasonable to consider the inverse image map f^* a *specification*, as it describes the observable behavior of the output in terms of observable properties of the input. This accords with the fact that any two continuous maps with the same inverse image maps are considered equal; the formal proofs of structure preservation can be ignored when reasoning, except for the requirement that those formal proofs must exist. However, the formal proofs are necessary for computing concrete results, and so we can consider the formal proof that f^* preserves joins and finitary meets as the *implementation* of the behavior specified by f^*. It is remarkable, then, that the specification is "complete," uniquely specifying a continuous map (by definition) if it exists.

Suppose we define a function $f : \Gamma \to_c A$. The map f can be applied to an input $x : \mathsf{Pt}(\Gamma)$ to produce an outpoint point $f(x) : \mathsf{Pt}(A)$. How then does one inspect $f(x)$ to get concrete results about where the output lies within A?

One computes concrete results in formal topology by splitting a point with an open cover. This open cover may be arbitrarily fine. By combining the SPLIT and MEET-0 rules, we observe that if we present a point y in a space A with an open cover of the whole space $\top \leq \bigvee_{i:I} U_i$, then y can *compute* an $i : I$ such that y lies in U_i. If a point lies in several opens in an open cover, it may return any index corresponding to an open it lies in. In the case of the point $f(x)$ where $f : \Gamma \to_c A$, f translates a cover of A into a cover of Γ, from which x can then compute some open from that cover of Γ that it lies in.

For instance, a point $x : \mathsf{Pt}(\mathbb{R})$, when given a formal proof of the open cover

$$\top \leq \bigvee_{b:\mathbb{B}} \text{if } b \text{ then } (\cdot < 1) \text{ else } (\cdot > -1),$$

must either return a concrete Boolean true together with a proof $x \Vdash \cdot < 1$ or return false with a proof $x \Vdash \cdot > -1$. If a point lies in both opens, whether it returns true or false depends on both the "implementation" of the point (the formal proofs that x satisfies SPLIT and MEET-0) as well as the formal proof of the cover.

In implementing a real number, one must provide a function which, given any tolerance $\varepsilon > 0$, produces a rational approximation within that tolerance. Imagine two implementations of the real number 0, where the first always returns 0, whereas the second returns $\varepsilon/2$. Consider a formal proof of the above cover of $\mathbb{R}$ that proceeds in the following manner: first, approximate to a tolerance of 1 with the cover $\top \leq \bigvee_{q:\mathbb{Q}} B_1(q)$. Then, for a given $q : \mathbb{Q}$, if the lower endpoint $q - 1$ of the resulting approximating interval $B_1(q)$ is at most 0, then prove that $B_1(q) \leq (\cdot < 1)$, and if not, prove $B_1(q) \leq (\cdot > -1)$. With this particular formal covering proof, the first implementation of 0 will return true, $(\cdot < 1)$, but the second will return false, $(\cdot > -1)$. However, if this cover is applied to the real number 2, the observation will be false, no matter the implementation of 2 or the particular proof of the covering.

Note that the MEET-2 rule does not figure into this notion of computation. We can think of maps $x \Vdash \cdot : \mathcal{O}(A) \to \mathcal{O}(*)$ that satisfy SPLIT and MEET-0 but not MEET-2 as nondeterministic values (see Chapter 3). But there is a notion of incremental computation where MEET-2 has a computational meaning.

Suppose we probe a point $x : \mathsf{Pt}(A)$ with a cover $\top \leq \bigvee_{i:I} U_i$ and learn that $x \Vdash U_{i_*}$ for some particular $i_* : I$. Then we may decide to further refine our knowledge of x by probing with another open cover $U_{i_*} \leq \bigvee_{j:J} V_j$, or equivalently, $U_{i_*} \leq \bigvee_{j:J} U_{i_*} \wedge V_j$, to learn that $x \Vdash U_{i_*} \wedge V_{j_*}$ for some $j_* : J$. This gives a sort of "sequential composition" of splitting with covers. The MEET-2 rule allows two independent threads of computation to be joined. For instance, we might probe x with an open cover to learn $x \Vdash U$, and independently with another open cover (of the whole space, not just U) to learn that $x \Vdash V$. By MEET-2, we can combine what we've learned to determine $x \Vdash U \wedge V$, and can continue to refine where x is with covers of $U \wedge V$. For nondeterministic values, where MEET-2 may not hold, this isn't satisfied, since (informally) each independent thread of refinement may get a different nondeterministic realization.

2.3 Spaces

2.3.1 Discrete spaces

The simplest kinds of spaces are those that just represent sets. Let **Set** denote the category of sets. The objects of **Set** are types A together with an equivalence relation $=_A : A \to A \to \Omega$ on A, and the arrows $f : A \to B$ are those equivalence-preserving functions, that is, functions $f : A \to B$ on the underlying types such that for all $a, a' : A$, if $a =_A a'$, then $f(a) =_B f(a')$.

We will show that it is possible to work with sets within the framework of spaces. Precisely, we can define a full and faithful functor $\mathsf{Discrete} : \mathbf{Set} \to \mathbf{FSpc}$ that exhibits **Set** as the discrete spaces, which form a full subcategory of **FSpc**. Given a set A

(with equivalence relation $=_A$), we construct a formal space whose type of basic opens is A, with the inclusion preorder given by $=_A$. We think of a basic open $a : A$ as representing the subset $(\cdot =_A a)$ of A, which is open in the discrete topology (and hence the preorder defining inclusion of basic opens is discrete). Since every element $a : A$ represents both an open and a point of the the discrete space, for $a : A$ we will use the notation $\cdot = a$ to refer to the open, reserving a for the point of the space.

In $\mathsf{Discrete}(A)$, we have $a \triangleleft U$ if and only if there is some $a' : A$ such that $a =_A a'$ and $a' \in U$. We can use this to simplify the rules that a point x of a discrete space or a continuous map $f : \Gamma \to_c A$ between discrete spaces must satisfy:

$$\frac{x \Vdash a \qquad a =_A b}{x \Vdash b}\ \textsc{split} \qquad \frac{}{\exists a : A.\ x \Vdash a}\ \textsc{meet-0} \qquad \frac{x \Vdash a \qquad x \Vdash b}{a =_A b}\ \textsc{meet-2}$$

$$\frac{a =_A b}{f^*(a) =_{\Gamma \to \Omega} f^*(b)}\ \textsc{split} \qquad \frac{}{\exists a : A.\ \gamma \in f^*(a)}\ \textsc{meet-0}$$

$$\frac{\gamma \in f^*(a) \qquad \gamma' \in f^*(b) \qquad \gamma =_\Gamma \gamma'}{a =_A b}\ \textsc{meet-2}\ .$$

We observe that f^* identifies a relation between Γ and A. We read $\gamma \in f^*(a)$ as "γ is in the preimage of a under f" (which is equivalent to saying that f maps γ to a). The SPLIT rule says that this relation respects equality on A. MEET-0 says that the relation is total (from Γ to A), and MEET-2 says that the relation respects equality on Γ and also that each input maps to at most one output (up to $=_A$). Accordingly, f^* is a functional relation from the set Γ to the set A, so it is in bijective correspondence with $\Gamma \to_{\mathbf{Set}} A$, the collection of functions from the set Γ to the set A, meaning that the functor $\mathsf{Discrete}$ is full and faithful: continuous maps between discrete spaces are just functions on their underlying sets.

Recall the functor $\mathsf{Pt} : \mathbf{FSpc} \to \mathbf{Set}$ which takes a space A to its (large) set of points $\mathsf{Pt}(A)$, where two points are considered equal if they lie in the same basic opens. This is right adjoint to $\mathsf{Discrete}$, i.e., $\mathsf{Discrete} \dashv \mathsf{Pt}$, giving a correspondence for a set A and a space B

$$\frac{A \to_{\mathbf{Set}} \mathsf{Pt}(B)}{\mathsf{Discrete}(A) \to_c B}\ .$$

2.3.2 Open subspaces

Given a space A and an open $U : \mathcal{O}(A)$, we can form the open subspace $\{A \mid U\}$ of A by making $\mathcal{O}(\{A \mid U\})$ a quotient of $\mathcal{O}(A)$, identifying opens $P, Q : \mathcal{O}(A)$ in $\{A \mid U\}$ when $P \wedge U = Q \wedge U$. The quotient still defines a space, since the operation $\cdot \wedge U : \mathcal{O}(A) \to \mathcal{O}(A)$ preserves binary meets and small joins.

2.3.3 The Sierpínski space

The Sierpínski space Σ is fundamental in topology, defining the space of possible "truth values." Just as the subsets of a set S are in correspondence with functions $S \to \Omega$, the opens of a space A are in correspondence with the continuous maps $A \to_c \Sigma$ [39],

$$\mathcal{O}(A) \cong (A \to_c \Sigma).$$

The basic opens of Σ are $\mathcal{O}_{\mathsf{B}}(\Sigma) \triangleq \mathbb{B}_{\leq}$, where $\mathbb{B}_{\leq}$ is $\mathbb{B}$ with the "truth order," i.e., where false is strictly less than true. The Sierpínski space has no covering axioms. We have thus defined Σ.

In particular, we have

$$(* \to_c \Sigma) \cong \mathcal{O}(*) \cong \Omega,$$

so the points of Σ are just the propositions. Given any point x of Σ, the corresponding proposition is $x \Vdash \mathsf{false}$. In particular, there are points $\top_\Sigma$ and $\bot_\Sigma$, where $\top_\Sigma \Vdash \mathsf{false}$ but not $\top_\Sigma \Vdash \mathsf{false}$.

2.3.4 Disjoint unions (sums)

We have not come across any characterization of sum spaces in **FSpc**, so we describe them here.

From a family of spaces $(A_i)_{i:I}$ parameterized over some index type I[6], we can form their disjoint union space $\sum_{i:I} A_i$, which is the coproduct of the A_is in **FSpc**. Intuitively, $\sum_{i:I} A_i$ pastes all the A_is together, where points from different spaces are not considered near each other. When $I \equiv \mathbb{B}$ (as types), this specializes to binary sums, which we denote with $+$. For instance, we have the homeomorphism in **FSpc**

$$\mathbb{B} \cong * + *,$$

and more generally, for any type I (considered as a set with $\equiv$ as its equivalence relation),

$$\mathsf{Discrete}(I) \cong \sum_{i:I} *.$$

The preorder of basic opens of $\sum_{i:I} A_i$ is the coproduct of the basic opens of the constituent spaces,

$$\mathcal{O}_{\mathsf{B}}\left(\sum_{i:I} A_i\right) \triangleq \sum_{i:I} \mathcal{O}_{\mathsf{B}}(A_i).$$

The preorder relation for the coproduct preorder $\sum_{i:I} \mathcal{O}_{\mathsf{B}}(A_i)$ is generated as the

[6] In this section, we require the index type I to satisfy uniqueness of identity proofs (UIP), which states that

$$\forall a, b : I.\ \forall p, q : a \equiv b.\ p \equiv q.$$

inductive type with the single constructor

$$\frac{a \leq_{A_i} b}{(i,a) \leq_{\sum_{j:I} A_j} (i,b)}.$$

The axioms for $\sum_{i:I} A_i$ are then just a sort of "coproduct" of the axioms of the constituent spaces. For a basic open $(j,a) : \mathcal{O}_{\mathsf{B}}(\sum_{i:I} A_i)$, for each axiom $a \lhd_{A_j} U$, we add the axiom

$$(j,a) \lhd \mathsf{InDisjunct}_j(U),$$

where $\mathsf{InDisjunct}_j : \mathcal{O}(A_j) \to \mathcal{O}(\sum_{i:I} A_i)$ is inductively generated by the constructor

$$\frac{a \in U}{(j,a) \in \mathsf{InDisjunct}_j(U)}.$$

Theorem 2.15.

$$A_j \cong \left\{ \sum_{i:I} A_i \mid \mathsf{InDisjunct}_j(\top) \right\}$$

Proof sketch. Define $f : A_j \to_c \left\{ \sum_{i:I} A_i \mid \mathsf{InDisjunct}_j(\top) \right\}$ to have its inverse image map $f^* : \mathcal{O}\left(\left\{ \sum_{i:I} A_i \mid \mathsf{InDisjunct}_j(\top) \right\}\right) \to \mathcal{O}(A_j)$ as the inductive family generated by the constructor

$$\frac{a =_{A_j} a'}{a' \in f^*(j,a)}\ .$$

Note that with this definition, $f^*(i,b) = \bot$ whenever $i \not\equiv j$.

Define $g : \left\{ \sum_{i:I} A_i \mid \mathsf{InDisjunct}_j(\top) \right\} \to_c A_j$ by

$$g^*(a) \triangleq (j,a).$$

It should be clear that f^* and g^* are inverses of each other, so we must just confirm that f^* and g^* indeed define continuous maps, which is mostly a calculational matter. For f, SPLIT is straightforward. We only note that proving that g satisfies MEET-0 reduces to

$$\top \lhd \mathsf{InDisjunct}_j(\top),$$

which is exactly the additional covering rule given by the open subspace of f's output. □

Once open embeddings have been defined (Section 3.4.2), this theorem will establish the open embedding $\mathsf{inj}_j : A_j \hookrightarrow \sum_{i:I} A_i$.

2.3.5 Products

We next introduce notation and relevant properties for the construction of product spaces in formal topology[7].

For a family of spaces $(A_i)_{i:I}$ parameterized over some index type I[8], we denote their product in **FSpc** by $\prod_{i:I} A_i$. The key idea characterizing the structure of the product space is that we can make an observation on the product space by choosing a component and making an observation on that component. Since we can only make finitely many observations, any open will make non-trivial observations on only finitely many components.

More precisely, the sub-basic opens of $\prod_{i:I} A_i$ are also given by the coproduct preorder $\sum_{i:I} \mathcal{O}_{\mathsf{B}}(A_i)$. However, we will use the notation $[i \mapsto a]$ for $i : I$ and $a : \mathcal{O}_{\mathsf{B}}(A_i)$ to refer to such a sub-basic open, because it represents the idea that the ith component lies in a. Similarly, we can define the open $[i \mapsto U] : \mathcal{O}\left(\prod_{i:I} A_i\right)$ for an arbitrary open $U : \mathcal{O}(A_i)$ as inductively generated by the constructor

$$\frac{a \in U}{[i \mapsto a] \in [i \mapsto U]} \ .$$

The universal property of product spaces says that for any space Γ and maps $f_i : \Gamma \to_c A_i$, we can construct a continuous map $g : \Gamma \to_c \prod_{i:I} A_i$. The inverse image map acts according to [9]

$$g^* : \mathcal{O}_{\mathsf{SB}}\left(\prod_{i:I} A_i\right) \to \mathcal{O}(\Gamma)$$
$$g^*([i_k \mapsto a_k]) \triangleq f_{i_k}^*(a_k).$$

If we consider the index type $I \equiv \mathbb{B}$, we get the binary products, which we denote $A \times B$ for spaces A and B. We notice that every basic open of $A \times B$ is equivalent to one of the form

$$[\mathsf{true} \mapsto U] \wedge [\mathsf{false} \mapsto V]$$

for $U : \mathcal{O}(A)$ and $V : \mathcal{O}(B)$, and so we may use the notation $U \times V : \mathcal{O}_{\mathsf{B}}(A \times B)$ to represent a basic open of $A \times B$ (also known as an *open rectangle*). Accordingly, every open of $A \times B$ can be represented as a union of open rectangles.

Given $f : \Gamma \to_c A$ and $g : \Gamma \to_c B$, we denote by $\langle f, g \rangle : \Gamma \to_c A \times B$ the "pair" given by the universal property of products.

[7] Vickers [42] provides a full definition and characterization.

[8] Again we require that I satisfy UIP.

[9] It suffices to only define g^* on its sub-basic opens ($\mathcal{O}_{\mathsf{B}}\left(\prod_{i:I} A_i\right)$) since g^* will be required to preserve finitary meets, and the basic opens are just finitary meets of sub-basic opens.

2.3.6 Metric spaces, including $\mathbb{R}$

It is possible to extend a set with a metric defined on it (such as $\mathbb{Q}$) to a metrically complete (i.e., Cauchy complete) formal space (such as $\mathbb{R}$) [41]. This section describes this construction.

Suppose we are given a *metric set*, a set X with a distance metric relation $d : \mathbb{Q}^+ \times X \times X \to \Omega$, where $d(\varepsilon, x, y)$ indicates the proposition that the distance between x and y is at most ε. The predicate d must in fact define the closed-ball relation for a metric, meaning it must satisfy the following rules:[10]

$$\frac{}{d(\varepsilon, x, x)}\ \text{REFL} \qquad \frac{d(\varepsilon, x, y)}{d(\varepsilon, y, x)}\ \text{SYM} \qquad \frac{d(\delta, x, y) \qquad d(\varepsilon, y, z)}{d(\delta + \varepsilon, x, z)}\ \text{TRIANGLE}$$

$$\frac{\forall \delta : \mathbb{Q}^+.\ d(\varepsilon + \delta, x, y)}{d(\varepsilon, x, y)}\ \text{CLOSED}\ .$$

The basic opens of the metric completion of X (if $X \equiv \mathbb{Q}$, this would be $\mathbb{R}$) will be the formal balls, the type $\mathsf{Ball}(X)$ defined by the single constructor

$$\frac{\varepsilon : \mathbb{Q}^+ \qquad x : X}{B_\varepsilon(x) : \mathsf{Ball}(X)}\ .$$

We define a relation $<$ on balls that indicates when one ball contains another with "room to spare" all around, and then a relation $\leq$ on balls indicating containment of one formal ball within another:

$$B_\delta(x) < B_\varepsilon(y) \triangleq \exists \gamma : \mathbb{Q}^+.\ d(\gamma, x, y) \text{ and } \gamma + \delta < \varepsilon$$
$$B_\delta(x) \leq B_\varepsilon(y) \triangleq \forall \gamma : \mathbb{Q}^+.\ B_\delta(x) < B_{\varepsilon+\gamma}(y).$$

We then define the metric completion space $\mathcal{M}(X)$ of X as having the basic opens $\mathsf{Ball}(X)$, and the following axioms:

$$\frac{\varepsilon : \mathbb{Q}^+}{\top \lhd \{B_\varepsilon(q) \mid q : \mathbb{Q}\}}\ \text{APPROX} \qquad \frac{}{B_q(\varepsilon) \lhd \{B_{\varepsilon'}(q') \mid B_{\varepsilon'}(q') < B_\varepsilon(q)\}}\ \text{SHRINK}$$

Definition 2.16. *A function $f : X \to Y$ on metric sets X and Y is* k-Lipschitz *(for $k : \mathbb{Q}^+$) if for all $x, y : X$ and $\varepsilon : \mathbb{Q}^+$, if $d(\varepsilon, x, y)$ then $d(k\varepsilon, f(x), f(y))$.*

We proved in Coq that f can be extended to a continuous map $g : \mathcal{M}(X) \to_c$

[10] This definition of a closed-ball relation is due to O'Connor [24]. We use it so that our Coq library is compatible with the Coq Repository at Nijmegen (CoRN), which has many constructive results regarding metric spaces.

$\mathcal{M}(Y)$ defined by the inverse image map[11]

$$g^* : \mathsf{Ball}(Y) \to \mathcal{O}\left(\mathcal{M}(X)\right)$$
$$g^*(b) \triangleq \{B_\delta(x) \mid B_{k\delta}(f(x)) < b\}.$$

The fact that f is k-Lipschitz implies that for balls $a, a' : \mathsf{Ball}(X)$, if $a \leq a'$ and $a' \in g^*(b)$, then $a \in g^*(b)$.

Theorem 2.17. *g^* preserves joins, $\top$, and binary meets.*

Proof sketch. **MEET-0** For every formal ball $B_\delta(x)$ of X, there is a ball of Y, $B_{\delta+k\delta}(f(x))$, such that $B_\delta(x) \leq g^*(B_{\delta+k\delta}(f(x)))$. Thus $\top \leq g^*(\top)$.

MEET-2 We are given a ball of Y, $B_{k\delta}(f(x))$, that lies strictly within two other balls of Y, $B_{k\delta}(f(x)) < B_\varepsilon(y)$ and $B_{k\delta}(f(x)) < B_{\varepsilon'}(y')$. We must prove in X that

$$B_\delta(x) \lhd g^*(B_\varepsilon(y) \downarrow B_{\varepsilon'}(y')).$$

By the properties of $<$, we can shrink each of the larger balls by just a bit, $\gamma : \mathbb{Q}^+$ while maintaining strict containment, so we get $B_{k\delta}(f(x)) < B_{\varepsilon-\gamma}(y)$ and $B_{k\delta}(f(x)) < B_{\varepsilon'-\gamma}(y')$. Then for some even smaller tolerance $\gamma' < \gamma$, we use the covering axiom APPROX(γ'/k) to derive

$$B_\delta(x) \lhd \{B_{\gamma'/k}(x') \mid x' : X\} \downarrow B_\delta(x).$$

This reduces our problem to showing that any ball of X with radius at most γ'/k that is contained within $B_\delta(x)$ is covered by $g^*(B_\varepsilon(y) \downarrow B_{\varepsilon'}(y'))$, which in fact follows by reflexivity, which we will now show. We must only consider the "worst case," where we have a ball $B_{\gamma'/k}(z)$ such that $B_{\gamma'/k}(z) \leq B_\delta(x)$. We will prove $B_{\gamma'/k}(z) \in g^*(B_\varepsilon(y) \downarrow B_{\varepsilon'}(y'))$. We do so in two steps, showing that $B_{\gamma'/k}(z) \in g^*(B_\gamma(f(z)))$ and that $B_\gamma(f(z)) \in B_\varepsilon(y) \downarrow B_{\varepsilon'}(y')$. The former is almost immediate. The latter is surprisingly intricate, depending on the fact that f is k-Lipschitz as well as using the triangle inequality.

SPLIT We first prove that g^* preserves unary covers, and then proceed by induction on covering axioms. Preserving unary covers reduces to proving that, given balls $b \leq b'$ in Y that $g^*(b) \leq g^*(b')$, which is follows from the fact that if $a < b$ and $b \leq b'$, then $a < b'$.

We now proceed by induction on the covering axioms.

In the case of APPROX(ε), given balls $B_\delta(x) : \mathsf{Ball}(X)$ and $B_\gamma(y) : \mathsf{Ball}(Y)$ such that $B_\delta(x) \in g^*(B_\gamma(y))$ show that

$$B_\delta(x) \lhd g^*(B_\gamma(y) \downarrow \{B_\varepsilon(y') \mid y' : Y\}).$$

[11] This definition of the inverse image map is based on a theorem of Vickers (Theorem 4.9 and Remark 4.10 in [41]) extending 1-Lipschitz functions to their metric completions. Vickers omits the proof that the inverse image map defines a continuous map, saying it is "routine to check."

To do so, we can find some $\alpha : \mathbb{Q}^+$ such that $\alpha < \varepsilon/k$, and cover $B_\delta(x)$ with balls of this radius,

$$B_\delta(x) \lhd B_\delta(x) \downarrow \{B_\alpha(x') \mid x' : X\}.$$

Then

$$B_\delta(x) \downarrow \{B_\alpha(x') \mid x' : X\} \lhd g^*(B_\gamma(y) \downarrow \{B_\varepsilon(y') \mid y' : Y\})$$

follows by reflexivity.

In the case of SHRINK, the covering in fact follows from reflexivity.

□

The extension of Lipschitz functions from metric sets to their metric completions can be used to define addition ($+ : \mathbb{R} \times \mathbb{R} \to_c \mathbb{R}$), for instance, in conjunction with facts relating products of spaces and of metric sets, such as [41]

$$\mathcal{M}(X \times Y) \cong \mathcal{M}(X) \times \mathcal{M}(Y).$$

Since $\times : \mathbb{R} \times \mathbb{R} \to_c \mathbb{R}$ is not Lipschitz, it cannot be defined using the above construction. However, since it is locally Lipschitz, it can be defined as a "gluing" of many Lipschitz maps defined on open subspaces using overlapping pattern matching, as described in Section 4.1.4.

In $\mathbb{R}$, open intervals and open "rays" (e.g., $(0 < \cdot)$) are evidently open, as they can be described as unions of open balls. For instance, we define

$$\begin{aligned}&(0 < \cdot) : \mathcal{O}(\mathbb{R})\\ &(0 < \cdot) \triangleq \{B_\varepsilon(x) \mid 0 \leq x - \varepsilon\}.\end{aligned}$$

In $\mathbb{R} \times \mathbb{R}$, the relations $(<), (\neq) : \mathcal{O}(\mathbb{R} \times \mathbb{R})$ are open. We can define $<$ by

$$(<) \triangleq \{(B_\varepsilon(x), B_\delta(y) \mid x + \varepsilon \leq y - \delta\}$$

and then $\neq$ in terms of that.

2.4 As a programming language

Since **FSpc** is a cartesian monoidal category (i.e., it has well-behaved products), it admits a restricted λ-calculus syntax that compiles to continuous maps, which this thesis uses freely.

Following Escardó [9], we describe the syntax as an inductive family $\vdash$: $\mathsf{list}(\mathbf{FSpc}) \to \mathcal{U}$ with constructors

$$\frac{(x : X) \in \Gamma}{\Gamma \vdash x : X}\ \textsc{var}$$

$$\frac{f : A_1 \times \cdots \times A_n \to_c B \qquad \Gamma \vdash x_1 : A_1 \qquad \cdots \qquad \Gamma \vdash x_n : A_n}{\Gamma \vdash f(x_1, \ldots, x_n) : B}\ \textsc{app},$$

where in the VAR rule $(x : X) \in \Gamma$ denotes a witness that X is a member of the list Γ. Let $\mathsf{Prod} : \mathsf{list}(\mathbf{FSpc}) \to \mathbf{FSpc}$ compute the product of a list of types.

Theorem 2.18. *Given any expression $\Gamma \vdash e : A$, we can construct a term $[\![e]\!] : \mathsf{Prod}(\Gamma) \to_c A$, and conversely, given a function $f : \mathsf{Prod}(\Gamma) \to_c B$, there is an expression $\Gamma \vdash e : B$.*

Proof. First, we construct a continuous map from syntax by induction on its structure. In the VAR case, we compute from the witness $(x : X) \in \Gamma$ a projection $[\![x]\!] : \mathsf{Prod}(\Gamma) \to_c X$. In the APP case, we are given maps $[\![x_i]\!] : \Gamma \to_c A_i$ for $i \in 1, \ldots n$. We use these maps to build a map $x : \Gamma \to_c A_1 \times \cdots \times A_n$ by the universal property for products, i.e., $x \triangleq \langle [\![x_1]\!], \ldots, [\![x_n]\!] \rangle$. Then $f \circ x : \Gamma \to_c B$.

Conversely, given $f : \mathsf{Prod}(\Gamma) \to_c B$, we can build the term

$$\Gamma \vdash f(\pi_1, \ldots, \pi_n) : B,$$

where each π_i is a variable $\Gamma \vdash \pi_i : \Gamma[i]$. □

This allows us to define a continuous map by introducing variables and applying continuous maps to them. For instance, we can define a map like

$$f : \mathbb{B} \times \mathbb{R} \times \mathbb{R} \to_c \mathbb{R}$$
$$f(x, y, z) \triangleq \mathsf{if}(x, y, -y) + z \times z$$

rather than having to manually arrange it as a "linear" composition of continuous maps. If we don't want to give a name to a function such as the one above, we may choose to write it with an "anonymous" lambda,

$$\lambda(x, y, z).\ \mathsf{if}(x, y, -y) + z \times z.$$

In later chapters, we will expand this syntax. In chapter 3, we will show that similar languages can be define for representing partial or nondeterministic functions. In chapter 4, we will add syntax for pattern matching, and in chapter 5, we will allow quantification over compact spaces (in certain cases).

2.4.1 Opens and Σ-valued maps

For any space A, there is a correspondence $\mathcal{O}(A) \cong A \to_c \Sigma$ between opens of A and Σ-valued continuous maps on A. We can use this to describe opens via Σ-valued

continuous maps; this is particularly convenient for describing opens of spaces which are finite products. We will use the notation $\{x : A \mid e\}$ where e is a Σ-valued term that may mention x, $x : A \vdash e : \Sigma$, to denote the open subspace $\{A \mid [\![e]\!]\}$. This thesis will readily conflate opens and Σ-valued continuous maps, implicitly converting between the two.

Chapter 3

Partiality and nondeterminism

The real line $\mathbb{R}$ is connected, meaning that any continuous map $f : \mathbb{R} \to_c A$ to a discrete set A must be a constant map. In particular, every map $f : \mathbb{R} \to_c \mathbb{B}$ is constant. The practical implications of connectedness are severe: it is impossible to (continuously) make (non-trivial) discrete decisions over variables that come from connected spaces such as $\mathbb{R}$.

In order to make decisions, we must give something up. While it is impossible to make discrete decisions on $\mathbb{R}$ that are total and deterministic, we *can* make decisions that are either partial (only defined on some open subspace of the input space) or nondeterministic (could potentially give different answers even when given the exact same input).

This chapter defines notions of partiality and nondeterminism related to continuous maps and characterizes the open maps and open embeddings as those continuous maps having partial and/or nondeterministic inverses. While each of these subjects has been studied in the context of constructive topology, we contribute the integrated characterization relating them, summarized by Figures 3-1 and 3-4.

Recall that points (or continuous maps) must satisfy SPLIT (preserving joins), MEET-0 (preserving $\top$), and MEET-2 (preserving binary meets). Each rule corresponds to a computational property. The SPLIT rule says that we can refine our knowledge of a point by "splitting" it with an open cover. The MEET-0 rule can be viewed as a rule enforcing totality: viewing $\top$ as the predicate representing the entire space, MEET-0 says that a point must lie in the entire space. If we were not to require the MEET-0 rule, it would be legal to define a relaxed "point" x that does not satisfy *any* properties. Accordingly, we can view values x that do not necessarily satisfy MEET-0 as *partial* points, and continuous maps that fail to satisfy MEET-0 as partial continuous maps.

The MEET-2 rule can be viewed as enforcing determinism. Spatially, the rule says that if a point lies in two opens, it must lie in their intersection. Computationally, it says that it should be possible to reconcile answers given from two independent lines of "questioning" from the SPLIT rule. Relaxed "points" which are not required to satisfy either MEET-0 or MEET-2 can be viewed as both partial and nondeterministic.

We can consider categories with spaces as objects but whose maps are like continuous maps, but the inverse image maps need not necessarily satisfy either MEET-0

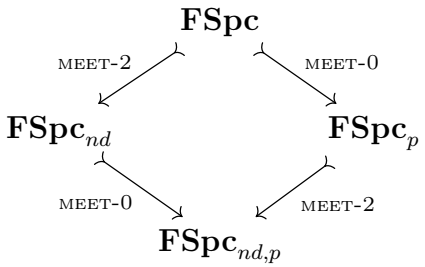


Figure 3-1: The lattice of categories representing potentially nondeterministic or partial maps on spaces.

or MEET-2. This gives us a lattice of categories that represent nondeterministic and partial maps, depicted in Figure 3-1[1], where each arrow denotes a faithful ("forgetful") functor where a particular rule is no longer required for inverse image maps. What is remarkable about these forgetful functors is that they have right adjoints, such that they induce a family of (strong) monads on **FSpc** for representing partiality and nondeterminism. The remainder of this chapter will characterize these monads.

To understand how partiality and nondeterminism arise by forgetting MEET-0 and MEET-2, respectively, it is instructive to observe how the categories of partial and nondeterministic values restrict to the discrete spaces. For discrete spaces, we have that $a \lhd U$ if and only if there is some a' such that $a = a'$ and $a' \in U$. The SPLIT rule, then, says that an inverse image map must respect the equivalence relations on the relevant sets. For discrete spaces $a \downarrow b$ is inhabited if and only if $a = b$, so MEET-2 reduces to the requirement

$$\frac{x \Vdash a \qquad x \Vdash b}{a = b} \quad .$$

Accordingly, a point x of the space given by a discrete set S is a subset $(x \Vdash \cdot)$ of S that has exactly one member: MEET-0 says it has at least one member and MEET-2 says it has at most one member. Of course, such subsets are in bijective correspondence with elements of S, so that points of the discrete space S correspond to elements of S. A continuous map $f : S \to_c T$ of discrete spaces is a relation $f^* : T \to S \to \Omega$ which is functional from S to T, in that for all $s : S$, there is some $t : T$ such that $s \in f^*(t)$ (by MEET-0), and for all $s : S$ and $t, t' : T$, if $s \in f^*(t)$ and $s \in f^*(t')$, then $t = t'$ (by MEET-2). As functional relations are in bijective correspondence with functions, so are continuous maps of discrete spaces $S \to_c T$ in correspondence with functions $S \to T$ on their underlying sets.

If we do not require MEET-2, the relaxed continuous maps are the multivalued functions from S to T; this justifies the sense in which we consider $\mathbf{FSpc}_{nd}$ to be the category of nondeterministic maps. If we instead do not require MEET-0, the relaxed continuous maps are the partial functions from S to T. If we require neither MEET-0 nor MEET-2, these relaxed continuous maps are just those (equality-respecting)

[1] nd stands for nondeterministic, p for partial.

relations between S and T.

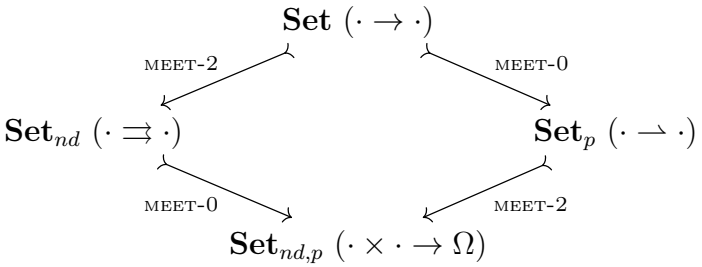


Figure 3-2: The restriction of the categories of nondeterministic and partial maps of spaces to the discrete spaces.

3.1 Partiality

Partiality allows definition of a continuous map that is only defined on an open subspace of the domain.

Definition 3.1. *A* partial map *f from A to B, written $f : A \xrightarrow{p}_c B$, is a map $f^* : \mathcal{O}(B) \to \mathcal{O}(A)$ that preserves joins and binary meets, but not necessarily $\top$.*

Example. *Reconsider the task of approximately comparing a real number with 0. We can define a partial comparison*

$$\mathsf{cmp} : \mathbb{R} \xrightarrow{p}_c \mathbb{B}$$

by only defining a continuous map on the open subspace $\{\mathbb{R} \mid \cdot \neq 0\}$ of $\mathbb{R}$. We specify its observable behavior with its inverse image map

$$\mathsf{cmp}^*(\cdot = \mathsf{true}) \triangleq \cdot > 0$$
$$\mathsf{cmp}^*(\cdot = \mathsf{false}) \triangleq \cdot < 0.$$

The inverse image map cmp^* in fact defines a partial map, as cmp^* preserves joins and binary meets, but it is not total, since it fails to preserve $\top$.

Proof. To confirm cmp^* preserves binary meets: it suffices to check binary meets of basic opens, which is only interesting when they differ, so we confirm

$$\begin{aligned}\mathsf{cmp}^*((\cdot = \mathsf{true}) \wedge (\cdot = \mathsf{false})) &= \mathsf{cmp}^*(\bot) \\ &= \bot \\ &= (\cdot > 0) \wedge (\cdot < 0) \\ &= \mathsf{cmp}^*(\cdot = \mathsf{true}) \wedge \mathsf{cmp}^*(\cdot = \mathsf{false}).\end{aligned}$$

However, cmp^* doesn't preserve $\top$, since

$$\top \not\leq f^*(\top) = (\cdot < 0) \vee (\cdot > 0).$$

□

Theorem 3.2. *There is a bijective correspondence*

$$\frac{A \xrightarrow{p}_c B}{\sum_{U:\mathcal{O}(A)} (\{A \mid U\} \to_c B)}$$

between partial maps and continuous maps defined on some open subspace of the domain.

Proof. We will show that any $f : A \xrightarrow{p}_c B$ can be considered as a continuous map $f : \{A \mid f^*(\top)\} \to_c B$. That is, if we consider opens of A equivalent whenever they are equal when met with $f^*(\top)$, then f^* preserves joins and finitary meets up to this weaker equivalence. Since f^* already preserves joins and binary meets, it certainly preserves them with this weaker equivalence on A. It only remains to prove that f^* preserves $\top$ up to this weaker equivalence, which follows the fact that

$$f^*(\top) =_{\{A|f^*(\top)\}} \top$$

since

$$f^*(\top) \wedge f^*(\top) =_A \top \wedge f^*(\top).$$

Now, we will show the opposite direction: Given some $U : \mathcal{O}(A)$ and $f : \{A \mid U\} \to_c B$, then we can also consider f as a partial map $f : A \xrightarrow{p}_c B$. That is, we know that f^* preserves joins and finitary meets up to equivalence on $\{A \mid U\}$, and must show that f^* preserves joins and binary meets on A. This follows from the fact that the operation $\cdot \wedge U$ of intersection with U preserves joins and binary meets. □

Corollary 3.3. *There is a bijective correspondence*

$$\frac{A \xrightarrow{p}_c *}{\mathcal{O}(A)}$$

between partial maps from A to the one-point space and opens of A.

Proof. This follows from the above theorem and the fact that $*$ is the terminal object in **FSpc**, so there is exactly one continuous map from any space to $*$. □

Corollary 3.4. *There is a bijective correspondence*

$$\frac{* \xrightarrow{p}_c A}{\sum_{P:\Omega} (P \to \mathsf{Pt}(A))}$$

between partial values of A and points of A "guarded" by some arbitrary proposition P.

Proof. This follows from the correspondence between opens of $*$ and propositions. □

This tells us that to prove that a partial point is actually total could require proving an arbitrary proposition. Assuming classical logic, the proposition P is either true, in which case we indeed have a point of A, or it is false, in which case no point of A was defined.

We can extend specialization order to partial maps in the obvious way. Just as in the continuous case, $f \leq g$ means that g is "at least as defined" as f is.

3.1.1 Lifted spaces

As mentioned previously, the "forgetful" functor from **FSpc** to $\mathbf{FSpc}_p$ has a right adjoint, such that there is a (strong) monad $\cdot_\perp : \mathbf{FSpc} \to \mathbf{FSpc}$ giving a correspondence

$$\frac{A \xrightarrow{p}_c B}{A \to_c B_\perp}$$

between continuous maps and their partial counterparts. By this correspondence, we can think of the space $B_\perp$ as the space of partial values. We will call spaces of the form $B_\perp$ *lifted spaces*. Vickers describes them briefly in *Topology via Logic* [39].

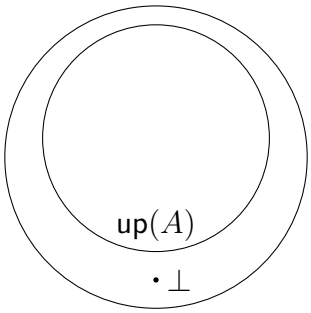


Figure 3-3: A schematic diagram of the lifted space $A_\perp$.

Figure 3-3 depicts a picture that in some sense represents what $A_\perp$ looks like: it has an open subspace $\mathsf{up}(A)$ (defined in Section 3.4.2) that looks exactly like A, and a single point $\perp : \mathsf{Pt}(A_\perp)$ outside of that open subspace, which represents an undefined or nonterminating value.

We will now describe the construction of lifted spaces and their properties. Algebraically, we can think of the construction of lifted spaces as a free construction on the lattice of opens. Suppose we want to construct from a space A its lifted space $A_\perp$. Then we want to have an operator $\Downarrow : \mathcal{O}(A) \to \mathcal{O}(A_\perp)$ which should preserve the structure from A that we want to keep: joins and binary meets. We will build such an operator using the structure of A as a formal space.

Since we want $\Downarrow$ to preserve joins, it suffices to define it for basic opens. We will add in a new top element T as well. Accordingly, the lifted space $A_\perp$ has basic opens generated by the constructors

$$\frac{a : \mathcal{O}_{\mathsf{B}}(A)}{\Downarrow a : \mathcal{O}_{\mathsf{B}}(A_\perp)} \qquad\qquad \frac{}{\mathsf{T} : \mathcal{O}_{\mathsf{B}}(A_\perp)}$$

and the preorder on the basic opens is inductively generated by the constructors

$$\frac{a \leq_A b}{\Downarrow a \leq_{A_\perp} \Downarrow b} \qquad\qquad \frac{}{a \leq_{A_\perp} \mathsf{T}} \ .$$

We extend $\Downarrow$ to operate on opens of A so that it preserves joins: for $U : \mathcal{O}(A)$, $\Downarrow U$ is generated by the constructor

$$\frac{a \in U}{\Downarrow a \in \Downarrow U} \ .$$

Now we describe the covering axioms in $A_\perp$. We simply copy over the covering axioms from A. For each axiom $a \lhd U$ in A, we add the axiom $\Downarrow a \lhd \Downarrow U$ in $A_\perp$.

Proposition 3.5. *Indeed* T *is the top element of* $A_\perp$, $\top \leq \mathsf{T}$.

Proof. It suffices to show that for any basic open $a : \mathcal{O}_{\mathsf{B}}(A_\perp)$, $a \leq \mathsf{T}$, which follows directly from the definition of $\leq_{A_\perp}$. □

Lifted spaces satisfy a property much stronger than compactness[2]:

Proposition 3.6. *If* $\top \lhd U$, *then* $\mathsf{T} \in U$.

Proof. It is equivalent to prove that if $\mathsf{T} \lhd U$ then $\mathsf{T} \in U$. We can proceed by induction on the proof of covering. The root node of the proof could not have been the use of an axiom, because there are no axioms for covering T. Therefore, there must be some basic open $u \in U$ such that $\mathsf{T} \leq u$. Since T is the largest basic open, it must be that $u \equiv \mathsf{T}$, so $\mathsf{T} \in U$. □

This means that any covering of $A_\perp$ is necessarily trivial. What this means is that one cannot get (nontrivial) information about a point of $A_\perp$ by splitting the point with a cover. This makes sense, because we require refinement by splitting to occur in finite time, and we want to allow $A_\perp$ to represent points that may be partial. To get useful information from a point in a lifted space, one should *prove* that the point actually lies in $\Downarrow \top$.

Every space $A_\perp$ has a point $\perp$ which has no interesting information: it lies only in the entire space. This corresponds to the undefined or nonterminating value. We

[2] A space is compact if whenever $\top \lhd U$, then there is a (Kuratowski-) finite subset K of U such that $\top \lhd K$.

describe the basic opens that $\bot$ lies in by the inverse image map

$$\begin{aligned} \bot \Vdash \cdot : \mathcal{O}_{\mathsf{B}}(A) &\to \mathcal{O}(1) \\ \bot \Vdash \mathsf{T} &\triangleq \top \\ \bot \Vdash \Downarrow a &\triangleq \bot. \end{aligned}$$

Proposition 3.7. *$\bot$ indeed describes a point of $A_\bot$.*

Proof. **MEET-0** Follows from $\bot \Vdash \mathsf{T}$.

MEET-2 Trivially satisfied, since there are no nontrivial intersections of basic opens that $\bot$ lies in.

SPLIT Follows from Proposition 3.6.

□

The idea that $\bot$ is the "least defined" point can be made formal, in that it is minimal in terms of specialization order.

Proposition 3.8. *For any (generalized) point $x : \Gamma \to_c A_\bot$, $\bot \leq x$ (in terms of specialization order).*

Proof. Since $\bot$ lies only in T, it suffices to show that $\top \leq x^*(\mathsf{T})$. However, since $\mathsf{T} = \top$, this is equivalent to $\top \leq x^*(\top)$, which is true of every continuous map. □

Having established some intuition about what lifted spaces represent, we can confirm they fulfill their original purpose:

Theorem 3.9. *There is the bijective correspondence*

$$\frac{A \xrightarrow{p}_c B}{A \to_c B_\bot}.$$

Proof. Given $f : A \xrightarrow{p}_c B$, we define $g : A \to_c B_\bot$ by

$$\begin{aligned} g^* : \mathcal{O}_{\mathsf{B}}(B_\bot) &\to \mathcal{O}(A) \\ g^*(\mathsf{T}) &\triangleq \top \\ g^*(\Downarrow b) &\triangleq f^*(b). \end{aligned}$$

The map g^* preserves $\top$ since $g^*(\mathsf{T}) = \top$ and preserves joins and binary meets since f^* does.

Conversely, given $g : A \to_c B_\bot$, we define $f : A \xrightarrow{p}_c B$ by

$$\begin{aligned} f^* : \mathcal{O}(B) &\to \mathcal{O}(A) \\ f^*(U) &\triangleq g^*(\Downarrow U). \end{aligned}$$

The map f^* preserves joins and binary meets since g^* and $\Downarrow$ each do. □

This correspondence in natural in A and B, giving an adjunction and thus a monad. It remains to show that $\cdot_\perp$ in fact defines a *strong* monad. Its tensorial strength $s : A \times B_\perp \xrightarrow{p}_c A \times B$ is defined by the inverse image map

$$\begin{aligned} s^* &: \mathcal{O}_{\mathsf{B}}(A \times B) \to \mathcal{O}\,(A \times B_\perp) \\ s^*(a \times b) &\triangleq a \times \Downarrow b. \end{aligned}$$

The inverse image map s^* preserves joins and binary meets since $\Downarrow$ does. We claim (but do not prove) that s satisfies the strong monad laws.

3.2 Nondeterminism

Nondeterminism allows the definitions of programs whose observable behavior might depend on the *exact implementation* of their inputs (specifically, the formal proofs that their inputs preserve joins and finitary meets). Rather than viewing such behavior as breaking the abstraction provided by the equivalence relation on points (since points that lie in the same opens may be treated differently), we can instead choose to maintain this abstraction and view such behavior as fundamentally nondeterministic.

For instance, we can perform a nondeterministic approximate comparison of a real number with 0.

Example. *Fix some error tolerance parameter $\varepsilon > 0$. We may define a total but nondeterministic approximate comparison with 0*

$$\mathsf{cmp} : \mathbb{R} \xrightarrow{nd}_c \mathbb{B},$$

allowing error up to ε, by specifying its observable behavior with the inverse image map

$$\begin{aligned} \mathsf{cmp}^*(\cdot = \mathsf{true}) &\triangleq \cdot > -\varepsilon \\ \mathsf{cmp}^*(\cdot = \mathsf{false}) &\triangleq \cdot < \varepsilon. \end{aligned}$$

We can confirm that cmp^* *in fact defines a nondeterministic map, as it preserves joins and $\top$, but fails to preserve binary meets.*

Proof. Since cmp's codomain is discrete, it trivially satisfies SPLIT. First, we confirm it preserves $\top$:

$$\begin{aligned} \mathsf{cmp}^*(\top) &= \mathsf{cmp}^*(\cdot = \mathsf{true}) \vee \mathsf{cmp}^*(\cdot = \mathsf{false}) \\ &= (\cdot > -\varepsilon) \vee (\cdot < \varepsilon) \\ &= \top. \end{aligned}$$

However, it fails to preserve binary meets, since

$$\mathsf{cmp}^*((\cdot = \mathsf{true}) \wedge (\cdot = \mathsf{false})) = \mathsf{cmp}^*(\bot) = \bot$$

but

$$\begin{aligned}\mathsf{cmp}^*(\cdot = \mathsf{true}) \wedge \mathsf{cmp}^*((\cdot = \mathsf{false}) &= (\cdot > -\varepsilon) \wedge (\cdot < \varepsilon) \\ &= -\varepsilon < \cdot < \varepsilon,\end{aligned}$$

which is a positive open (thus, not $\bot$). □

As we might expect, we can perform nondeterministic joins of (generalized) points in $\mathbf{FSpc}_{nd}$.

Proposition 3.10. *For any inhabited index type I and space A, there is the nondeterministic map $\sqcup : \prod_{i:I} A \overset{nd}{\to}_c A$ defined by*

$$\begin{aligned}(\sqcup)^* &: \mathcal{O}(A) \to \mathcal{O}\left(\prod_{i:I} A\right) \\ (\sqcup)^*(a) &\triangleq \bigvee_{i:I} [i \mapsto a].\end{aligned}$$

Proof. Let $i^* : I$ witness that I is inhabited. Then this map satisfies MEET-0 since

$$\top \leq [i^* \mapsto \top] \leq \bigvee_{i:I} [i \mapsto \top] \leq (\sqcup)^*(\top).$$

Given any $a \lhd_A U$, we have

$$(\sqcup)^*(a) = \bigvee_{i:I} [i \mapsto a] \leq \bigvee_{i:I} [i \mapsto U] = (\sqcup)^*(U),$$

so SPLIT is satisfied as well. □

Computationally, we notice that $\bigsqcup_{i:I} x_i$ (for $x_i : \Gamma \overset{nd}{\to}_c A$) will behave as x_{i^*}.

We can extend specialization order to nondeterministic maps too, in the obvious way, and in this case $f \leq g$ represents that g can potentially behave as f (though it may also behave differently, as well).

3.2.1 Nondeterministic powerspaces

As mentioned previously, the "forgetful" functor from **FSpc** to $\mathbf{FSpc}_{nd}$ has a right adjoint, such that there is a monad $\mathcal{P}^+_\Diamond : \mathbf{FSpc} \to \mathbf{FSpc}$ giving a correspondence

$$\frac{A \overset{nd}{\to}_c B}{A \to_c \mathcal{P}^+_\Diamond(B)}$$

between continuous maps and their nondeterministic counterparts [40]. By this correspondence, we can think of the space $\mathcal{P}^+_\Diamond(B)$ as the space of nondeterministic values.

Spaces of the form $\mathcal{P}^+_\Diamond(B)$ are known as *positive lower powerspaces* (also known as positive or inhabited Hoare powerlocales).

There is a map $\Diamond : \mathcal{O}(A) \to \mathcal{O}\left(\mathcal{P}^+_\Diamond(A)\right)$ (read "possibly") that can be intuitively understood in the following sense: if $U : \mathcal{O}(A)$ is interpreted as a property of points of A, and a point $s : * \to_c \mathcal{P}^+_\Diamond(A)$ (where $*$ is the one-point space) is interpreted as a subspace of A, $\Diamond U : \mathcal{O}\left(\mathcal{P}^+_\Diamond(A)\right)$ holds of s if U holds of some point in s.

Just like $\Downarrow$ for lifted spaces, the $\Diamond$ operator preserves all the structure in the lattice $\mathcal{O}(A)$ that ought to be preserved in $\mathcal{O}\left(\mathcal{P}^+_\Diamond(A)\right)$: that is, it preserves joins and $\top$. This implies a particular fact that is critical in the computational properties of the nondeterministic powerspaces:

Theorem 3.11. *For every cover $\top \leq \bigvee_{i:I} V_i$ in A there is a cover $\top \leq \bigvee_{i:I} \Diamond V_i$ in $\mathcal{P}^+_\Diamond(A)$.*

Proof. Suppose $\top \leq \bigvee_{i:I} V_i$ in A. Since $\Diamond$ preserves $\top$ and joins,

$$\bigvee_{i:I} \Diamond V_i = \Diamond \left(\bigvee_{i:I} V_i \right) = \Diamond \top = \top.$$

□

The previous theorem can be understood computationally as allowing simulation of some nondeterministic result.

Classically, the points of $\mathcal{P}^+_\Diamond(B)$ are in correspondence with closed nonempty subspaces of B. Constructively, these subspaces also are *overt* [40], which is helpful for some computational tasks.

The specialization order on $\mathcal{P}^+_\Diamond(A)$ corresponds to subspace inclusion of possible values.

Vickers [40] describes how to construct this powerspace, as well the powerspace for $\mathbf{FSpc}_{nd,p}$, predicatively within $\mathbf{FSpc}$.

Vickers [40] shows that $\mathcal{P}^+_\Diamond$ defines a monad, but it remains to show that $\mathcal{P}^+_\Diamond$ in fact defines a *strong* monad. Its tensorial strength $s : A \times \mathcal{P}^+_\Diamond(B) \overset{nd}{\to}_c A \times B$ is defined by the inverse image map

$$\begin{aligned} s^* : \mathcal{O}_{\mathsf{B}}(A \times B) &\to \mathcal{O}\left(A \times \mathcal{P}^+_\Diamond(B)\right) \\ s^*(a \times b) &\triangleq a \times \Diamond b. \end{aligned}$$

The inverse image map s^* preserves joins and $\top$ since $\Diamond$ does.

3.3 Both partiality and nondeterminism

The category $\mathbf{FSpc}_{nd,p}$, where both nondeterminism and partiality are allowed, is quite similar to $\mathbf{FSpc}_{nd}$, where only nondeterminism is allowed.

In $\mathbf{FSpc}_{nd,p}$, inverse image maps are required only to preserve joins, and need not preserve any meets. Accordingly, $\mathbf{FSpc}_{nd,p}$ is equivalent to the category of suplattices, which has been studied in various contexts [37, 40, 15, 4].

3.3.1 Lower powerspaces

As mentioned previously, the "forgetful" functor from **FSpc** to $\mathbf{FSpc}_{nd,p}$ has a right adjoint, such that there is a monad $\mathcal{P}_\Diamond : \mathbf{FSpc} \to \mathbf{FSpc}$ giving a correspondence

$$\frac{A \overset{nd,p}{\to}_c B}{A \to_c \mathcal{P}_\Diamond(B)}$$

between continuous maps and their nondeterministic and partial counterparts [40]. By this correspondence, we can think of the space $\mathcal{P}_\Diamond(B)$ as the space of nondeterministic and partial values. Spaces of the form $\mathcal{P}_\Diamond(B)$ are known as *lower powerspaces* (also known as Hoare powerlocales).

There is a map $\Diamond : \mathcal{O}(B) \to \mathcal{O}(\mathcal{P}_\Diamond(B))$ (read "possibly") which distributes over joins but not necessarily meets. In particular,

$$\Diamond U \leq \bigvee_{i:I} \Diamond V_i$$

holds in $\mathcal{O}(\mathcal{P}_\Diamond(B))$ whenever $U \leq \bigvee_{i:I} V_i$ in $\mathcal{O}(B)$.

The monad $\mathcal{P}_\Diamond$ is, like $\mathcal{P}^+_\Diamond$, a strong monad, and its tensorial strength is almost the same: $s : A \times \mathcal{P}_\Diamond(B) \overset{nd,p}{\to}_c A \times B$ is defined by the inverse image map

$$s^* : \mathcal{O}_{\mathsf{B}}(A \times B) \to \mathcal{O}(A \times \mathcal{P}_\Diamond(B))$$
$$s^*(a \times b) \triangleq a \times \Diamond b.$$

The inverse image map s^* preserves joins since $\Diamond$ does.

3.4 Open maps and open embeddings

In pattern matching for functional programming, one may pattern match on an inductive type by checking whether it has the form of a particular constructor applied to some argument (i.e., it is in the image of the map defined by a particular constructor). The analogue of constructors for overlapping pattern matching will be the open maps and the open embeddings.

In this section, we will observe how *open maps* can be viewed as continuous maps that have nondeterministic and partial inverses, and *open embeddings* can be viewed as those that have partial (but deterministic) inverses. This is what makes the open maps and open embeddings relevant for constructing the patterns in overlapping pattern matching. We will also prove some properties of open maps and open embeddings that ensure that they will behave well as patterns.

3.4.1 Open maps

Definition 3.12 ([35]). *A continuous map $f : A \to_c B$ is an* open map *if the inverse image map $f^* : \mathcal{O}(B) \to \mathcal{O}(A)$ has a left adjoint $f_! : \mathcal{O}(A) \to \mathcal{O}(B)$, called the* direct image map*, that satisfies the Frobenius law,*

$$f_!(U \wedge f^*(V)) = f_!(U) \wedge V.$$

That is, f is an open map if the *image* of any open in A is open in B; the direct image map provides this mapping.

Example. *Identity maps $\mathsf{id} : A \to_c A$ are open maps, and the composition of open maps is an open map.*

Proof. We have $\mathsf{id}_!(U) = U$, which clearly satisfies $\mathsf{id}_! \dashv \mathsf{id}^*$ since they are all identity maps. It is also immediate that the Frobenius law holds for $\mathsf{id}_!$. Given open maps $f : A \to_c B$ and $g : B \to_c C$, we claim that the direct image map is $(g \circ f)_! = g_! \circ f_!$. We first confirm the adjunction:

$$\begin{aligned}
(g \circ f)_!(U) \leq V &\iff g_!(f_!(U)) \leq V \\
&\iff f_!(U) \leq g^*(V) && (g_! \dashv g^*) \\
&\iff U \leq f^*(g^*(V)) && (f_! \dashv f^*) \\
&\iff U \leq (g \circ f)^* V.
\end{aligned}$$

It only remains to confirm the Frobenius law for the composition $g_! \circ f_! : \mathcal{O}(A) \to \mathcal{O}(C)$:

$$\begin{aligned}
(g \circ f)_!(U \wedge (g \circ f)^*(V)) &= g_!(f_!(U \wedge f^*(g^*(V)))) \\
&= g_!(f_!(U) \wedge g^*(V)) && \text{(Frobenius law for } f) \\
&= g_!(f_!(U)) \wedge V \\
&= (g \circ f)_!(U) \wedge V. && \text{(Frobenius law for } g)
\end{aligned}$$

□

Proposition 3.13. *In the pullback square where p and q are open maps,*

$$\begin{array}{ccc}
A \times_X B & \xrightarrow{\theta} & A \\
\downarrow{\scriptstyle \varphi} & \lrcorner & \downarrow{\scriptstyle p} \\
B & \xrightarrow{q} & X
\end{array},$$

define $f : A \times_X B \to_c X$ by $f = p \circ \theta = q \circ \varphi$. Then

$$f_!(\top) = p_!(\top) \wedge q_!(\top).$$

Proof. Note that for any $U : \mathcal{O}(B)$, $p^*(q_!(U)) = \theta_!(\varphi^*(U))$ (proof in section 5.2 of [28]). Then

$$\begin{aligned} p_!(\top) \wedge q_!(\top) &= p_!(\top \wedge p^*(q_!(\top))) && \text{(Frobenius reciprocity)} \\ &= p_!(p^*(q_!(\top))) \\ &= p_!(\theta_!(\varphi^*(\top))) && \text{(the earlier equality)} \\ &= p_!(\theta_!(\top)) && (\varphi^* \text{ preserves } \top) \\ &= f_!(\top). \end{aligned}$$

□

Proposition 3.14. *Parallel composition of open maps yields an open map. That is, given $f : A \to_c B$ and $g : X \to_c Y$ open, $f \otimes g : A \times X \to_c B \times Y$ is open.*

Proof. We claim that the direct image map operates on basic opens (which are open rectangles of $A \times X$) by $(f \otimes g)_!(a \times x) = f_!(a) \times g_!(x)$ (this extends to all opens by taking joins). We confirm the adjunction (using the fact that every open is a join of basic opens):

$$\begin{aligned} (f \otimes g)_! \left(\bigvee_{i:I} a_i \times x_i \right) \leq \bigvee_{j:J} (b_j \times y_j) &\iff \bigvee_{i:I} f_!(a_i) \times g_!(x_i) \leq \bigvee_{j:J} (b_j \times y_j) \\ &\iff \forall i : I.\ \exists j : J.\ f_!(a_i) \times g_!(x_i) \leq b_j \times y_j \\ &\iff \forall i : I.\ \exists j : J.\ (f_!(a_i) \leq b_j) \text{ and } g_!(x_i) \leq y_j \\ &\iff \forall i : I.\ \exists j : J.\ (a_i \leq f^*(b_j)) \text{ and } x_i \leq g^*(y_j) \\ &\iff \forall i : I.\ \exists j : J.\ a_i \times x_i \leq f^*(b_j) \times g^*(y_j) \\ &\iff \bigvee_{i:I} a_i \times x_i \leq \bigvee_{j:J} f^*(b_j) \times g^*(y_j) \\ &\iff \bigvee_{i:I} a_i \times x_i \leq (f \otimes g)^* \left(\bigvee_{j:J} b_j \times y_j \right). \end{aligned}$$

It suffices to confirm the Frobenius law holds on basic opens [35]:

$$\begin{aligned} (f \otimes g)_!((a \times x) \wedge (f \otimes g)^*(b \times y)) &= (f \otimes g)_!((a \times x) \wedge (f^*(b) \times g^*(y))) \\ &= (f \otimes g)_!((a \wedge f^*(b)) \times (x \wedge g^*(y)) \\ &= f_!(a \wedge f^*(b)) \times g_!(x \wedge g^*(y)) \\ &= (f_!(a) \wedge b) \times (g_!(x) \wedge y) \\ &\qquad \text{(Frobenius law for } f \text{ and } g) \\ &= (f_!(a) \times g_!(x)) \wedge (b \times y) \\ &= (f \otimes g)_!(a \times x) \wedge (b \times y). \end{aligned}$$

□

We will now describe some facts that relate the open maps to nondeterministic maps.

Proposition 3.15. *For any open map $p : A \to_c B$, there is a (potentially) partial and nondeterministic inverse map $p^{-1} : B \overset{nd,p}{\to}_c A$ defined by*

$$p^{-1^*} : \mathcal{O}(B) \to \mathcal{O}(A)$$
$$p^{-1^*}(U) \triangleq p_!(U).$$

Proof. We only must prove that $p_!$ preserves joins, which follows from the fact that $p_!$ is a left adjoint (to p^*). □

The map $\{\cdot\} : A \to_c \mathcal{P}_\Diamond(A)$, defined as the "return" operation of the $\mathcal{P}_\Diamond$ monad, is open, with a direct image map taking opens U of A to $\Diamond U$ in $\mathcal{P}_\Diamond(A)$.

3.4.2 Open embeddings

Proposition 3.16. *An open map $f : A \to_c B$ factors through its direct image $f_!(\top)$, i.e., there is an $\tilde{f}$ such that the following diagram commutes:*

$$\begin{array}{ccc} A & \xrightarrow{\tilde{f}} & \{B \mid f_!(\top)\} \\ & \searrow^{f} & \downarrow{\scriptstyle \iota[f_!(\top)]} \\ & & B \end{array}$$

Proof. This statement is equivalent to that for all $U : \mathcal{O}(B)$, $f^*(U) \leq f^*(U \wedge f_!(\top))$. This is indeed the case:

$$\begin{aligned} f^*(U \wedge f_!(\top)) &= f^*(U) \wedge f^*(f_!(\top)) && (f^* \text{ preserves meets}) \\ &\geq f^*(U) \wedge \top && (f^* \circ f_! \text{ inflationary}) \\ &= f^*(U). \end{aligned}$$

□

Definition 3.17. *A map $f : A \to_c B$ is an* open embedding *(or* open inclusion*) if A is homeomorphic to its image under f in B, that is, if there is an open subspace $U : \mathcal{O}(B)$ and homeomorphism $\tilde{f} : A \to_c \{B \mid U\}$ such that the following diagram commutes:*

$$\begin{array}{ccc} A & \underset{\tilde{f}^{-1}}{\overset{\tilde{f}}{\rightleftarrows}} & \{B \mid U\} \\ & \searrow^{f} & \downarrow{\scriptstyle \iota[U]} \\ & & B \end{array}$$

The notation $f : A \hookrightarrow B$ indicates that the continuous map $f : A \to_c B$ is an open embedding.

Theorem 3.18. *A map $f : A \to_c B$ is an open embedding if and only if it is an open map and its direct image map $f_!$ preserves binary meets.*

Proof. Suppose $f : A \to_c B$ is an open embedding that factors through $\{B \mid U\}$, and let $\tilde{f}$ and $\tilde{f}^{-1}$ be the maps as in the above diagram. Then its direct image map is given by

$$\begin{aligned} f_! &: \mathcal{O}(A) \to \mathcal{O}(B) \\ f_!(V) &\triangleq \tilde{f}^{-1*}(V) \wedge U. \end{aligned}$$

We now confirm that $f_! \dashv f^*$. We have

$$\begin{aligned} f_!(V) \leq W &\iff \tilde{f}^{-1*}(V) \wedge U \leq W \\ &\iff \tilde{f}^{-1*}(V) \wedge \tilde{f}^{-1*}(\tilde{f}^*(U)) \leq W \\ &\iff \tilde{f}^{-1*}(V \wedge \tilde{f}^*(U)) \leq W \\ &\iff \tilde{f}^{-1*}(V \wedge \top) \leq W \\ &\iff \tilde{f}^{-1*}(V) \leq W \\ &\iff V \leq \tilde{f}^*(W) \end{aligned}$$

Moreover, $f_!$ preserves meets:

$$\begin{aligned} f_!(V \wedge W) &= \tilde{f}^{-1*}(V \wedge W) \wedge U \\ &= \tilde{f}^{-1*}(V) \wedge \tilde{f}^{-1*}(W) \wedge U \\ &= (\tilde{f}^{-1*}(V) \wedge U) \wedge (\tilde{f}^{-1*}(W) \wedge U) \\ &= f_!(V) \wedge f_!(W). \end{aligned}$$

We also confirm $f_!$ satisfies the Frobenius law:

$$\begin{aligned} f_!(V \wedge f^*(W)) &= f_!(V) \wedge f_!(f^*(W)) && (f_! \text{ preserves meets}) \\ &= f_!(V) \wedge \tilde{f}^{-1*}(f^*(W)) \wedge U \\ &= f_!(V) \wedge \tilde{f}^{-1*}(f^*(W)) && (f_!(V) \text{ already at most } U) \\ &= f_!(V) \wedge \tilde{f}^{-1*}(\tilde{f}^*(W \wedge U)) \\ &= f_!(V) \wedge W \wedge U \\ &= f_!(V) \wedge W && (f_!(V) \text{ already at most } U) \end{aligned}$$

Thus an open embedding is an open map with a meet-preserving direct image map. We now prove the converse. Given an open map $f : A \to_c B$ with a meet-preserving direct image map $f_!$, we claim that $A \cong \{B \mid f_!(\top)\}$. By Proposition 3.16, we already have a continuous map $\tilde{f} : A \to_c \{B \mid f_!(\top)\}$. We define $g : \{B \mid f_!(\top)\} \to_c A$ by

$$\begin{aligned} g^* &: \mathcal{O}(A) \to \mathcal{O}(\{B \mid f_!(\top)\}) \\ g^*(V) &\triangleq f_!(V) \wedge f_!(\top). \end{aligned}$$

We claim g^* indeed defines a continuous map. It preserves joins and binary meets since it is the composition of $f_!$ and $\cdot \wedge f_!(\top)$, both of which preserve joins and binary meets, so it suffices to show that $g^*(\top) = \top$, which is indeed the case, as

$$g^*(\top) = f_!(\top) \wedge f_!(\top) = f_!(\top)$$

which is equivalent to $\top$ in $\{B \mid f_!(\top)\}$. □

Example. *Identity maps* $\mathsf{id} : A \hookrightarrow A$ *are open embeddings, and the composition of open embeddings is an open embedding.*

Proof. We have $\mathsf{id}_!(U) = U$, which preserves binary meets. Given open maps $f : A \to_c B$ and $g : B \to_c C$, the composition $g_! \circ f_! : \mathcal{O}(A) \to \mathcal{O}(C)$ preserves binary meets since $f_!$ and $g_!$ both do. □

Proposition 3.19. *Parallel composition of open embeddings yields an open embedding. That is, given* $f : A \hookrightarrow B$ *and* $g : X \hookrightarrow Y$ *open embeddings,* $f \otimes g : A \times X \hookrightarrow B \times Y$ *is an open embedding.*

Proof. It suffices to confirm that the direct image preserves binary meets, and it suffices to check this by only checking the basic opens, which in this case are open rectangles:

$$\begin{aligned}
(f \otimes g)_! ((a_1 \times x_1) \wedge (a_2 \times x_2)) &= (f \otimes g)_! ((a_1 \wedge a_2) \times (x_1 \wedge x_2)) \\
&= f_!(a_1 \wedge a_2) \times g_!(x_1 \wedge x_2) \\
&= (f_!(a_1) \wedge f_!(a_2)) \times (g_!(x_1) \wedge g_!(x_2)) \\
&\qquad\qquad (f_! \text{ and } g_! \text{ preserve binary meets}) \\
&= (f_!(a_1) \times g_!(x_1)) \wedge (f_!(a_2) \times g_!(x_2)) \\
&= (f \otimes g)_!(a_1 \times x_1) \wedge (f \otimes g)_!(a_2 \times x_2)
\end{aligned}$$

□

Proposition 3.20. *The pullback of an open embedding is an open embedding.*

Proof. Since (up to homeomorphism) an open embedding is just the inclusion of an open subspace, it suffices to prove that the pullback of the inclusion of an open subspace is an open embedding. Given the open inclusion $\iota[U] : \{X \mid U\} \hookrightarrow X$, for any map $f : A \to_c X$ we have the pullback square

$$\begin{array}{ccc}
\{A \mid f^*(U)\} & \xhookrightarrow{\iota[f^*(U)]} & A \\
\downarrow{\scriptstyle f_{|U}} & \lrcorner & \downarrow{\scriptstyle f} \\
\{X \mid U\} & \xhookrightarrow{\iota[U]} & X
\end{array},$$

which expresses the fact that the preimage of opens are open. It is possible to confirm that the diagram commutes and that $\{A \mid f^*(U)\}$ satisfies the universal property of pullbacks. □

The open embeddings are closely related to the partial maps.

Proposition 3.21. *Given an open embedding $f : A \hookrightarrow B$, the "inverse map" $f^{-1} : B \overset{nd,p}{\to}_c A$ that any open map has is in fact deterministic, i.e., $f^{-1} : B \overset{p}{\to}_c A$.*

Proof. This follows directly from the fact that $f_!$ preserves joins and binary meets. □

We will now observe that the map $\mathsf{up} : A \to_c A_\perp$ that is the "return" operation of the lifting monad $\cdot_\perp$ is an open embedding. This allows us to view A as an open subspace of $A_\perp$. We can define up via the inverse and direct image maps

$$\begin{aligned} \mathsf{up}^* &: \mathcal{O}_{\mathsf{B}}(A_\perp) \to \mathcal{O}(A) \\ \mathsf{up}^*(\mathsf{T}) &\triangleq \top \\ \mathsf{up}^*(\Downarrow a) &\triangleq a \end{aligned}$$

$$\begin{aligned} \mathsf{up}_! &: \mathcal{O}(A) \to \mathcal{O}(A_\perp) \\ \mathsf{up}_!(U) &\triangleq \Downarrow U. \end{aligned}$$

Proposition 3.22. *This indeed defines an open embedding $\mathsf{up} : A \hookrightarrow A_\perp$.*

Proof. We first confirm up defines a continuous map:

MEET-0

$$\mathsf{up}^*(\top) \geq \mathsf{up}^*(\mathsf{T}) = \top.$$

MEET-2 Follows from the facts

$$\begin{aligned} \mathsf{T} \downarrow \Downarrow a &= \Downarrow a \\ \Downarrow a \downarrow \Downarrow b &= \Downarrow (a \downarrow b). \end{aligned}$$

SPLIT It suffices to check every axiom of $A_\perp$: every axiom is of the form $\Downarrow a \lhd_{A_\perp} \Downarrow U$, where $a \lhd_A U$. So we must confirm

$$\mathsf{up}^*(\Downarrow a) \lhd \mathsf{up}^*(\Downarrow U),$$

which is equivalent to $a \lhd U$, which we know by assumption.

We will now confirm that $\mathsf{up}_! = \Downarrow$ preserves joins and binary meets. We observe that it preserves binary meets since

$$\Downarrow a \downarrow \Downarrow b = \Downarrow (a \downarrow b).$$

It preserves joins since if $a \lhd_A U$ then $\Downarrow a \lhd_{A_\perp} \Downarrow U$.

Finally, it is straightforward to confirm that $\mathsf{up}_! \dashv \mathsf{up}^*$ and that the Frobenius law holds. □

3.5 Summary

The open maps and the open embeddings form part of lattice of categories of continuous maps whose *inverses* are potentially partial and nondeterministic; the inverse of an open map is partial and nondeterministic, whereas the inverse of an open embedding is partial, but deterministic. This will make the open maps and open embeddings relevant for pattern matching to produce either continuous maps or nondeterministic maps on spaces. Figure 3-4 depicts this lattice alongside the original one.

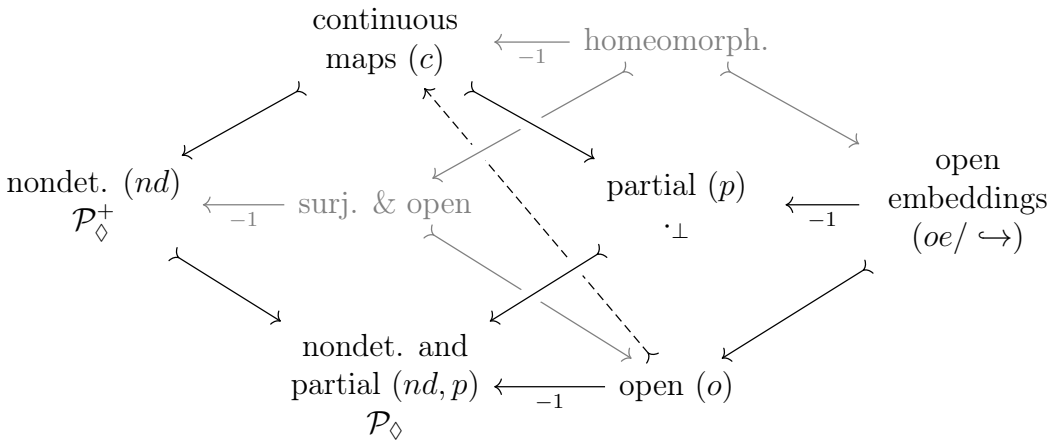


Figure 3-4: A summary of various categories of morphisms on formal spaces and functors relating them. "Forgetful" functors have tails and are unnamed. Functors named "-1" denote inverse functions determined by direct image maps. All paths in the diagram commute except for those involving the (dashed) forgetful functor $o \rightarrowtail c$.

Usually, when pattern matching, we do not expect that a single pattern covers the entire input space, and accordingly we expect that the "inverse" map of a particular pattern should be partial, rather than total. This explains why we focused on the open maps and open embeddings, whose inverses are necessarily partial. The greyed categories in Figure 3-4 describe those categories of maps whose inverses are in fact total. The inverses of surjective open maps are total but nondeterministic, and the inverses of homeomorphisms are just regular continuous maps (total and deterministic).

3.6 Variants of the programming language

We can extend the programming language for **FSpc** described in Section 2.4 to programming languages for nondeterministic or partial variants. This works since each of these variants describe strong monads M that act on **FSpc**.

For any strong monad M, we can define a syntax $\vdash_M$ for programming in that particular language:

$$\frac{(x : X) \in \Gamma}{\Gamma \vdash_M x : X}\ \textsc{var}$$

$$\frac{f : A_1 \times \cdots \times A_n \to_M B \qquad \Gamma \vdash_M x_1 : A_1 \qquad \cdots \qquad \Gamma \vdash_M x_n : A_n}{\Gamma \vdash_M f(x_1, \ldots, x_n) : B}\ \textsc{app},$$

Theorem 3.23. *Given any expression $\Gamma \vdash_M e : A$, we can construct a term $[\![e]\!] : \mathsf{Prod}(\Gamma) \to_M A$ in the Kleisli category for M.*

Proof. We proceed by induction on its structure of the syntax. In the VAR case, we compute from the witness $(x : X) \in \Gamma$ a projection $[\![x]\!] : \mathsf{Prod}(\Gamma) \to_c X$, which can also be considered as map in the Kleisli category for M. In the APP case, we are given maps $[\![x_i]\!] : \Gamma \to_M A_i$ for $i \in 1, \ldots n$. We use these maps to build a map $x : \Gamma \to_c M(A_1) \times \cdots \times M(A_n)$ by the universal property for products, i.e., $x \triangleq \langle [\![x_1]\!], \ldots, [\![x_n]\!] \rangle$.

The task then is to define a map $g : M(A_1 \times \cdots \times A_n) \to_M A_1 \times \cdots \times A_n$ such that we can produce the composition $f \circ g \circ x : \Gamma \to_M A_1 \times \cdots \times A_n$, which has the right type. This can be accomplished via repeated use of the strength operation $A \times M(B) \to_M A \times B$ (in conjunction with other monad operations). □

Chapter 4

Overlapping pattern matching

This chapter introduces a programming construct that ought to be useful for programming with spaces: *overlapping pattern matches*.

Reconsider the approximate root-finding task introduced in Section 1.1. The code

$$\mathsf{roots}_f : * \overset{nd}{\to}_c \{* \mid \forall x \in K.\ f(x) \neq 0\} + \{x : K \mid |f(x)| < \varepsilon\}$$

$$\mathsf{roots}_f \triangleq \mathsf{case}(\mathsf{tt}) \begin{cases} \iota[\exists x \in K.\ |f(x)| < \varepsilon]\,(y) & \Rightarrow \mathsf{inr}(\mathsf{simulate}(y)) \\ \iota[\forall x \in K.\ f(x) \neq 0]\,(n) & \Rightarrow \mathsf{inl}(n) \end{cases}$$

accomplishes this task with an overlapping pattern match. The program is nondeterministic (indicated by the flavor of the arrow $\overset{nd}{\to}_c$), returning either an $\mathsf{inr}(x)$ that is "almost a root" or a proof indicating that there are no roots in K. The notations and auxiliary definitions in this program will be explained over the course of this chapter and the following one, and we will return to this example to give a more thorough analysis in section 5.4.

Intuitively, overlapping patterns behave much like traditional pattern matches as in functional programming, but an input that matches several patterns may nondeterministically follow any matching branch.

The syntax of an overlapping pattern match determines a unique map that is potentially partial and nondeterministic, but there are simple conditions that ensure that a map is total and deterministic:

1. *Totality*: together, the cases cover the entire input space.

2. *Determinism*: patterns are open embeddings, and "maxima" exist for each pair of branches on their overlap.

Formal (constructive) proofs of these properties contribute to the computational behavior of the pattern match.

In the roots_f example, totality is satisfied essentially because either of the two earlier bulleted statements must hold (Section 5.4 provides further detail).

Contributions

- We demonstrate (Section 4.1), by informal example, the utility of overlapping patterns for a variety of purposes, including
  - (nondeterministic) approximate computation and decision-making (4.1.2)
  - manipulating partial data (4.1.3)
  - "sheafification" of constructions (4.1.4).
- We describe constructions of a few variations on overlapping pattern matches. We first characterize what families of patterns may be used if no properties are assumed about the code for the branches (4.2); these constructions generalize the conventional notion of pattern matching in functional programming. We then describe a construction that generalizes the sheaf gluing property of representable sheaves, where overlapping branches need not agree exactly (4.3). We then discuss some additional properties of overlapping patterns (4.4).

4.1 Example uses of overlapping pattern matching

This section presents an informal introduction to overlapping pattern matching, providing examples that demonstrate its usefulness in a variety of situations.

4.1.1 A familiar example

Here we will fix some notation and define concepts that will be used to explain the more exotic examples later in this section, by starting with a familiar example: we can define functions on coproducts (sums) via pattern matching, just as is possible in any functional programming language:

$$\mathsf{forget\_sign} : \{\mathbb{R} \mid \cdot < 0\} + \{\mathbb{R} \mid \cdot > 0\} \to_c \mathbb{R}$$
$$\mathsf{forget\_sign}(x) \triangleq \mathsf{case}(x) \begin{cases} \mathsf{inl}(\ell) & \Rightarrow \quad \iota[\cdot < 0]\,(\ell) \\ \mathsf{inr}(r) & \Rightarrow \quad \iota[\cdot > 0]\,(r) \end{cases}.$$

For general spaces A and B, $\mathsf{inl} : A \hookrightarrow A + B$ and $\mathsf{inr} : B \hookrightarrow A + B$ are open embeddings (theorem 2.15); every pattern must be an open embedding when defining deterministic functions. To confirm that $\mathsf{forget\_sign}$ defines a continuous map, we must check totality and determinism conditions. Since inl and inr together cover the input space, and since their images are disjoint, the program must be both total and deterministic.

4.1.2 Approximate computation and decision-making

Connected spaces, such as $\mathbb{R}$, cannot be partitioned into disjoint open subspaces. Accordingly, it is impossible to make a nontrivial discrete decision over these spaces;

for instance, any function $f : \mathbb{R} \to_c \mathbb{B}$ must be a constant function. Instead, it makes sense to allow decisions to be nondeterministic, which can be accomplished by producing output in the space $\mathcal{P}^+_\Diamond(\mathbb{B})$ of nondeterministic Boolean values, rather than $\mathbb{B}$.

For any error tolerance $\varepsilon > 0$, we can approximately compare a real number with 0 using the function

$$\begin{aligned} \mathsf{positive?}_\varepsilon &: \mathbb{R} \overset{nd}{\to}_c \mathbb{B} \\ \mathsf{positive?}_\varepsilon(x) &\triangleq \mathsf{case}(x) \begin{cases} \iota[\cdot > -\varepsilon]\,(\_) \Rightarrow \mathsf{true} \\ \iota[\cdot < \varepsilon]\,(\_) \Rightarrow \mathsf{false} \end{cases} . \end{aligned}$$

To confirm this definition is well-defined, we confirm that the two patterns cover $\mathbb{R}$. If an input lies in $-\varepsilon < \cdot < \varepsilon$, then the resulting behavior is the union of the behaviors of both branches: $\mathsf{true} \sqcup \mathsf{false} : * \overset{nd}{\to}_c \mathbb{B}$; that is, either decision might be made. Note that determining whether an input satisfies $\cdot > -\varepsilon$ is undecidable (likewise for $\cdot < \varepsilon$).

We can nondeterministically break a real number into an integer and a fractional part in $\{\mathbb{R} \mid -\varepsilon < \cdot < 1\}$ with

$$\begin{aligned} \mathsf{int\_frac}_\varepsilon &: \mathbb{R} \overset{nd}{\to}_c \mathbb{Z} \times \{\mathbb{R} \mid -\varepsilon < \cdot < 1\} \\ \mathsf{int\_frac}_\varepsilon(x) &\triangleq \mathsf{case}(x) \left\{ \mathsf{nplus}(n, y) \Rightarrow (n, y) \right. , \end{aligned}$$

where the open map $\mathsf{nplus} : \mathbb{Z} \times \{\mathbb{R} \mid -\varepsilon < \cdot < 1\} \to_o \mathbb{R}$ adds its arguments. It is possible for a real number to have several preimages under nplus; one can think of the pattern match as nondeterministically choosing all possible preimages.

Section 5 describes binary covers, giving many further examples of approximate decision-making using overlapping patterns.

4.1.3 Manipulating partial data

We can write, for instance, a partial Boolean or operation,

$$\begin{aligned} \mathsf{or} &: \mathbb{B}_\perp \times \mathbb{B}_\perp \overset{p}{\to}_c \mathbb{B} \\ \mathsf{or}(p) &\triangleq \mathsf{case}(p) \begin{cases} \mathsf{up}(\mathsf{true}), \_ & \Rightarrow \mathsf{true} \\ \_, \mathsf{up}(\mathsf{true}) & \Rightarrow \mathsf{true} \\ \mathsf{up}(\mathsf{false}), \mathsf{up}(\mathsf{false}) & \Rightarrow \mathsf{false} \end{cases} . \end{aligned}$$

This example demonstrates that it is possible to pattern match on products as well as to nest patterns. For instance, $\mathsf{up}(\mathsf{true})$ corresponds to the nested pattern

$$* \overset{\mathsf{true}}{\hookleftarrow} \mathbb{B} \overset{\mathsf{up}}{\hookleftarrow} \mathbb{B}_\perp .$$

Since $\mathbb{B}$ is discrete, its points are open embeddings.

Unlike a short-circuiting "or" operation in most programming languages, this or is

strict in neither argument, since

$$\mathsf{or}(\mathsf{up}(\mathsf{true}), \bot) = \mathsf{or}(\bot, \mathsf{up}(\mathsf{true})) = \mathsf{true},$$

and true is a total value.

We must check that the map is deterministic, by ensuring pairs of branches are compatible on their overlaps. The first and second cases do overlap, but they agree exactly, and other case pairs are disjoint from each other. So or is well-defined.

As we can see, when constructing a partial map with a pattern match, the patterns need not cover the entire input space.

Other operations on lifted spaces are naturally expressed with overlapping patterns. For instance, any $f : A \to_c B$ can be lifted to operate on lifted spaces with the overlapping pattern

$$\begin{aligned} f_\bot &: A_\bot \xrightarrow{p}_c B \\ f_\bot(x) &\triangleq \mathsf{case}(x) \left\{ \mathsf{up}(z) \quad \Rightarrow f(z), \right. \end{aligned}$$

which forces its argument to compute the result, but if the input is $\bot$, then so is the result. Since there is only one branch that does not return $\bot$, the gluing condition is satisfied. Lifting of spaces forms a monad, whose join operation can be expressed with the pattern match

$$\begin{aligned} \mathsf{join} &: (A_\bot)_\bot \xrightarrow{p}_c A \\ \mathsf{join}(x) &\triangleq \mathsf{case}(x) \left\{ \mathsf{up}(\mathsf{up}(y)) \quad \Rightarrow y. \right. \end{aligned}$$

Comparison with Haskell pattern matching

Pattern matching definitions of the form $A_{1\bot} \times \ldots \times A_{n\bot} \xrightarrow{p}_c B$ may resemble pattern matches in Haskell. The translated definitions sometimes, but not always, represent valid Haskell functions. For instance, the partial Boolean or operation corresponds to the following prospective Haskell definition

```
or :: Bool -> Bool -> Bool
or True _ = True
or _ True = True
or False False = False
```

which has overlapping cases, thus making it invalid Haskell code (Haskell gives a warning and ignores the second case). The problem is that both of the first two cases apply if both arguments are `True`, and in Haskell there is no way to guarantee that the results are compatible in either case. Our pattern matching construction allows overlapping patterns, since it requires proof that overlapping patterns indeed are reconcilable on their overlap.

4.1.4 "Sheafification" of constructions

Overlapping patterns naturally allow "sheafification" (see Section 4.3) of constructions, yielding a very general tool that aids in defining continuous maps. For instance, suppose we want to define multiplication on $\mathbb{R}$, where $\mathbb{R}$ is defined as a metric completion of the set $\mathbb{Q}$ (as in [41]). Using a straightforward construction to extend a Lipschitz function over metric sets to their metric completions, it is possible to define a family of Lipschitz functions that compute multiplication where one argument is bounded,

$$\mathsf{scale}_L : \{\mathbb{R} \mid -L < \cdot < L\} \times \mathbb{R} \to_c \mathbb{R}$$

for $L : \mathbb{Q}^+$. Multiplication on $\mathbb{R}$ is not Lipschitz and so cannot be defined directly with the Lipschitz construction. However, multiplication is *locally* Lipschitz: the family of bounded multiplication functions defined above cover the entire input domain $\mathbb{R} \times \mathbb{R}$, and so it is possible to define multiplication with the overlapping pattern

$$\begin{aligned} \times &: \mathbb{R} \times \mathbb{R} \to_c \mathbb{R} \\ x \times y &\triangleq \mathsf{case}(x) \\ &\quad \left\{ [L : \mathbb{Q}^+] \quad \iota[-L < \cdot < L]\,(x') \;\Rightarrow\; \mathsf{scale}_L(x', y). \right. \end{aligned}$$

The covering condition is clearly satisfied, and the gluing condition is satisfied because any two branches agree exactly on their region of overlap.

Just as the Lipschitz extension construction can be "sheafified" to allow extension of locally Lipschitz functions, pattern matching naturally enables "sheafification" of constructions more generally.

4.2 Pattern families

This section characterizes those families of patterns which may be used to match on a scrutinee that comes from a space A, if nothing is to be assumed about the branches. The idea is that we compose a function $f : A \stackrel{?}{\to}_c B$ by factoring through a disjoint sum over collection of spaces representing the possible patterns and branches, $\sum_{i:I} U_i$, i.e., a composition

$$A \xrightarrow{\mathsf{inv}} \textstyle\sum_{i:I} U_i \xrightarrow{e} B$$

of a "pattern matching" part inv followed by the "branch execution" part e. The collection of branches $(e_i : U_i \stackrel{?}{\to}_c B)_{i:I}$ exactly correspond to the branch execution function e, but the pattern matching part inv is more interesting; this section will address those families of patterns which may yield valid functions of this sort.

Semantically, we think of a single pattern as representing a space U together with a map $p : U \to_c A$ that represents the possibility that the scrutinee can be represented as a point in the image of p. For a single pattern $p : U \to_c A$ to be somehow implementable, it must have a well-behaved inverse $p^{-1} : A \stackrel{nd,p}{\to}_c U$ that is partial and also may be nondeterministic. That means that p should at the very least

be an open map. If we are building a program that is deterministic, then p^{-1} should be deterministic, which implies p must be an open embedding. If we are building a program that is total, then we do not need each p^{-1} to be total, but we do need the collection of them to cover A.

In general, we have an entire family of patterns $p_i : U_i \to_o A$ for $i : I$ where I is some index type. We can use this to construct the nondeterministic inverse

$$\mathsf{inv} : A \overset{nd,p}{\to}_c \sum_{i:I} U_i$$
$$\mathsf{inv}(x) \triangleq \bigsqcup_{i:I} \mathsf{inj}_i(p_i^{-1}(x)).$$

Therefore, for any index type I, any collection of opens maps $p_i : U_i \to_o A$ is a collection of patterns for defining a nondeterministic and partial map using an overlapping pattern match. Given an arbitrary collection of branches $e_i : U_i \overset{nd,p}{\to}_c B$, or equivalently, $e : \sum_{i:I} U_i \overset{nd,p}{\to}_c B$, the overlapping pattern match is just the composition $e \circ \mathsf{inv} : A \overset{nd,p}{\to}_c B$. As expected, the overlapping pattern match is a nondeterministic union of its branches.

For those categories/languages that require totality or determinism, we'd like to characterize the families of patterns which are suitable in those cases. These will be subcollections of the collection of pattern families in the nondeterministic and partial case.

We will observe that they form a lattice of Grothendieck pretopologies. This structure is useful: it tells us that we there are certain techniques that we can always use to form pattern families, and that pattern families will have important structural properties. For instance, the transitivity axiom corresponds to the ability to flatten nested pattern matches into a single one.

Definition 4.1 ([19])**.** *A* Grothendieck pretopology *(or* basis for a Grothendieck topology*) is an assignment to each space A of a collection of families $(U_i \to_c A)_{i:I}$ of continuous maps, called* covering families *such that*

1. homeomorphisms cover – *every family consisting of a single isomorphism $\{V \overset{\cong}{\to}_c A\}$ is a covering family;*

2. stability axiom – *the collection of covering families is stable under pullback: if $(U_i \to_c A)_{i:I}$ is a covering family and $f : V \to_c A$ is any continuous map, then the family of pullbacks $(f^* U_i \to_c V)_{i:I}$ is a covering family;*

3. transitivity axiom – *if $(U_i \to_c A)_{i \in I}$ is a covering family and for each i also $(U_{i,j} \to_c U_i)_{j:J_i}$ is a covering family, then also the family of composites $(U_{i,j} \to_c U_i \to_c A)_{i:I, j:J_i}$ is a covering family.*

Proposition 4.2 (Product axiom)**.** *In any Grothendieck pretopology, given covering families $(p_i : U_i \to_c A)_{i:I}$ and $(q_j : V_j \to_c B)_{j:J}$, there is a product covering family $(p_i \otimes q_j : U_i \times V_j \to_c A \times B)_{(i,j):I \times J}$.*

Proof. First, we use pullback stability along $\mathsf{fst} : A \times B \to_c A$ to produce the covering family $(\mathsf{fst}^* p_i : U_i \times B \to_c A \times B)_{i:I}$. Then, for each $i : I$, we use pullback stability along $\mathsf{snd} : U_i \times B \to_c B$ to produce the covering family $(\mathsf{snd}^* q_j : U_i \times V_j \to_c U_i \times B)_{j:J}$. Finally, by the transitivity axiom, we get the covering family

$$(\mathsf{fst}^* p_i \circ \mathsf{snd}^* q_j : U_i \times V_j \to_c A \times B)_{i:I, j:J}.$$

Then one can confirm that

$$\mathsf{fst}^* p_i \circ \mathsf{snd}^* q_j = p_i \otimes q_j.$$

□

Proposition 4.3. *The collection of pattern families for* $\boldsymbol{FSpc}_{nd,p}$*, that is, open maps* $p_i : U_i \to_o A$*, form a Grothendieck pretopology* $\mathcal{J}_{nd,p}$*.*

Proof.

1. Homeomorphisms cover, since any homeomorphism is (an open embedding and thus) an open map.

2. Stability follows from the fact that the pullback of an open map against any continuous map is open (see Proposition C3.1.11 (i) in [15]).

3. Transitivity follows from the fact that open maps are closed under composition.

□

4.2.1 Totality

If we want to ensure that an overlapping pattern match is total, only requiring that the branches themselves $e : \sum_{i:I} U_i \overset{nd}{\to}_c B$ are total, then it suffices to require that $\mathsf{inv}^*(\top) = \top$, or equivalently,

$$\top \leq \bigvee_{i:I} p_{i!}(\top),$$

meaning that the patterns cover the whole input space.

Proposition 4.4. *The collection of pattern families for* $\boldsymbol{FSpc}_{nd}$*, that is, open maps* $p_i : U_i \to_o A$ *satisfying* $\top \leq \bigvee_{i:I} p_{i!}(\top)$*, form a Grothendieck pretopology* $\mathcal{J}_{nd}$*.*

Proof.

1. An isomorphism $p : V \to_o A$ satisfies $p_!(\top) = \top$ and so it alone is a pattern family on A.

2. Given a pattern family $p_i : U_i \to_o A$ and a continuous map $f : V \to_c A$, the pullback object $f^* U_i$ is homeomorphic to $\{V \mid f^*(p_{i!}(\top))\}$. Using this definition of the pullback object, we confirm

$$\bigvee_{i:I} (f^* p_i)_!(\top) = \bigvee_{i:I} f^*(p_{i!}(\top)) = f^* \left(\bigvee_{i:I} p_{i!}(\top) \right) = f^*(\top) = \top.$$

3. The family of composites indeed covers, since

$$\begin{aligned}\bigvee_{i:I,j:J_i} (p_i \circ p_{i,j})_!(\top) &= \bigvee_{i:I,j:J_i} p_{i!}(p_{i,j!}(\top)) \\ &= \bigvee_{i:I} p_{i!}\left(\bigvee_{j:J_i} p_{i,j!}(\top)\right) \qquad \text{(direct images preserve joins)} \\ &= \bigvee_{i:I} p_{i!}(\top) \\ &= \top.\end{aligned}$$

□

4.2.2 Determinism

Suppose we want to ensure an overlapping pattern match is deterministic, while allowing any arbitrary family of branches $e : \sum_{i:I} U_i \xrightarrow{p}_c B$ (where each branch e_i is indeed deterministic).

In this case, we will recover the familiar condition required of pattern matching in functional programming: disjointness of patterns (i.e., patterns are not allowed to overlap).

Definition 4.5. *A family of open maps $p_i : U_i \to_o A$ is* pairwise disjoint *if whenever $p_{i!}(\top) \wedge p_{j!}(\top)$ is positive in A, then $i \equiv j$.*

Theorem 4.6. *The map inv^* preserves binary meets if and only if both of the following conditions hold:*

1. *Each pattern $p_i : U_i \to_o A$ is in fact an open embedding.*

2. *The patterns are pairwise disjoint.*

Proof. First, we prove the two conditions hold if inv^* preserves binary meets.

1. It suffices to show that each $p_{i!}$ preserves binary meets, which follows from the calculation

$$p_{i!}(a \wedge b) = \mathsf{inv}^*((i,a) \wedge (i,b)) = \mathsf{inv}^*(i,a) \wedge \mathsf{inv}^*(i,b) = p_{i!}(a) \wedge p_{i!}(b).$$

2. We must show that if $p_{i!}(\top) \wedge p_{j!}(\top)$ is positive in A, then $i \equiv j$.

$$\begin{aligned}p_{i!}(\top) \wedge p_{j!}(\top) &= \mathsf{inv}^*(i,\top) \wedge \mathsf{inv}^*(j,\top) \\ &= \mathsf{inv}^*((i,\top) \wedge (j,\top)).\end{aligned}$$

   Since inv^* preserves joins, by Proposition 2.14 if $p_{i!}(\top) \wedge p_{j!}(\top)$ is positive, so is $(i,\top) \wedge (j,\top)$, which implies that $i \equiv j$.

Now we prove the converse: if a family $p_i : U_i \hookrightarrow A$ of open embeddings is pairwise disjoint, its inv map is in fact deterministic. It suffices to prove that inv^* preserves binary meets of basic opens. For any open $a : \mathcal{O}(A)$ and proposition Q, let $\chi_Q(a) \triangleq \bigvee_{q:Q} a$. If Q is true, then $\chi_Q(a) = a$ and if Q is false, then $\chi_Q(a) = \bot$.

$$\begin{aligned}
\mathsf{inv}^*((k,a) \wedge (\ell,b)) &= \bigvee_{i:I} p_{i!}(\chi_{i\equiv k}(a) \wedge \chi_{i\equiv \ell}(b)) \\
&= \bigvee_{i:I} p_{i!}(\chi_{i\equiv k}(a)) \wedge p_{i!}(\chi_{i\equiv \ell}(b)) && (p_{i!} \text{ preserves meets}) \\
&= \bigvee_{i:I,j:I} p_{i!}(\chi_{i\equiv k}(a)) \wedge p_{j!}(\chi_{j\equiv \ell}(b)) && (p_i\text{s pairwise disjoint}) \\
&= \left(\bigvee_{i:I} p_{i!}(\chi_{i\equiv k}(a))\right) \wedge \left(\bigvee_{j:I} p_{j!}(\chi_{j\equiv \ell}(b))\right) \\
&= \mathsf{inv}^*(k,a) \wedge \mathsf{inv}^*(\ell,b).
\end{aligned}$$

□

Proposition 4.7. *The collection of pattern families for* $\boldsymbol{FSpc}_p$*, that is, open embeddings* $p_i : U_i \hookrightarrow A$ *that are pairwise disjoint, form a Grothendieck pretopology* $\mathcal{J}_p$*.*

Proof. 1. Any cover with a single pattern is trivially pairwise disjoint.

2. By Proposition 3.20, the pullback of an open embedding is an open embedding. Thus it remains to confirm that pullback preserves disjointness. From the computation

$$(f^*p_i)_!(\top) \wedge (f^*p_j)_!(\top) = f^*(p_{i!}(\top)) \wedge f^*(p_{j!}(\top)) = f^*(p_{i!}(\top) \wedge p_{j!}(\top)),$$

and since f^* preserves joins, by Proposition 2.14 if $(f^*p_i)_!(\top) \wedge (f^*p_j)_!(\top)$ is positive, then so is $p_{i!}(\top) \wedge p_{j!}(\top)$, which implies that $i \equiv j$. Therefore, the pullback of the cover is still pairwise disjoint.

3. Since the composition of open embeddings is an open embedding, it only remains to confirm that transitivity preserves disjointness. Suppose we consider two composites $p_i \circ p_{i,j}$ and $p_{i'} \circ p_{i',j'}$ such that

$$(p_i \circ p_{i,j})_!(\top) \wedge (p_{i'} \circ p_{i',j'})_!(\top) = p_{i!}(p_{i,j_!}(\top)) \wedge p_{i'!}(p_{i',j'_!}(\top))$$

is positive. This implies that the larger open

$$p_{i!}(\top) \wedge p_{i'!}(\top)$$

is also positive, and therefore, $i \equiv i'$. We will now prove that $j \equiv j'$. Since

$i \equiv i'$, by equality induction we can consider them both i, so that we know that

$$\begin{aligned}(p_i \circ p_{i,j})_!(\top) \wedge (p_i \circ p_{i,j'})_!(\top) &= p_{i!}(p_{i,j!}(\top)) \wedge p_{i!}(p_{i,j'!}(\top)) \\ &= p_{i!}(p_{i,j!}(\top) \wedge p_{i,j'!}(\top)) \quad (p_{i!} \text{ preserves meets})\end{aligned}$$

is positive. Note that any direct image map $f_!$ preserves joins, and accordingly, if $f_!(U)$ is positive then U is positive. Therefore,

$$p_{i,j!}(\top) \wedge p_{i,j'!}(\top)$$

is positive, and by pairwise disjointness of the covering family, we know $j \equiv j'$. Therefore, $(i,j) \equiv (i',j')$.

□

4.2.3 Totality and determinism

To create a pattern match which is both total and deterministic, we simply combine the separate conditions for totality and determinism.

Proposition 4.8. *The collection of pattern families for **FSpc**, that is, open embeddings $p_i : U_i \hookrightarrow A$ that are pairwise disjoint and cover A, form a Grothendieck pretopology $\mathcal{J} = \mathcal{J}_{nd} \cap \mathcal{J}_p$ on **FSpc**.*

Proof. It is straightforward from the definition of a Grothendieck pretopology that they are closed under arbitrary intersection. □

Definition 4.9. *An open U is* clopen *(for "closed" and "open") if it has a Boolean complement, i.e., there is another open V such that $U \vee V = \top$ and $U \wedge V = \bot$.*

Proposition 4.10. *For a pattern family $(p_i : U_i \hookrightarrow A)_{i:I}$ on **FSpc**, if the index type I has decidable equality, then for each $i : I$, $p_{i!}(\top)$ is clopen in A.*

Proof. Fix some $i : I$. We claim that the Boolean complement of $p_{i!}(\top)$ is $\bigvee_{j:I|j\neq i} p_{j!}(\top)$. Since I has decidable equality, their join is $\top$:

$$p_{i!}(\top) \vee \bigvee_{j:I|j\neq i} p_{j!}(\top) = \bigvee_{i:I} p_{i!}(\top) = \top.$$

Pairwise disjointness implies that their meet is $\bot$:

$$p_{i!}(\top) \wedge \bigvee_{j:I|j\neq i} p_{j!}(\top) = \bigvee_{j:I|j\neq i} p_{i!}(\top) \wedge p_{j!}(\top) = \bigvee_{j:I|j\neq i} \bot = \bot.$$

□

This means that if the index type I has decidable equality, the predicate corresponding to any given pattern is decidable. This recovers the usual understanding of pattern matching in functional programming, where patterns are disjoint and correspond to decidable predicates.

4.2.4 Syntax of patterns

We will define the admissable syntax of patterns, where $p : A \dashv \Gamma$ intuitively means that the syntax p provides a "context" of pattern matching variables Γ by pattern matching on a space A. For instance, we could have the pattern

$$(\mathsf{up}(x), y) : A_\perp \times B \dashv x : A, y : B$$

on a space $A_\perp \times B$ that provides variables $x : A$ and $y : B$ to be used in the branch corresponding to that pattern.

$$\frac{f : * \to_? A}{f : A \dashv_? \cdot}\ \text{CONSTANT} \qquad \frac{}{v : A \dashv_? v : A}\ \text{VAR} \qquad \frac{}{\_ : A \dashv_? \cdot}\ \text{WILDCARD}$$

$$\frac{p : U \dashv_? \Gamma \qquad f : U \to_? A}{f(p) : A \dashv_? \Gamma}\ \text{COMPOSE} \qquad \frac{p : A \dashv_? \Gamma \qquad q : B \dashv_? \Delta}{p, q : A \times B \dashv_? \Gamma, \Delta}\ \text{PRODUCT}$$

Figure 4-1: The syntax for patterns. The symbol ? can either be *o*, for open maps, or *oe* for open embeddings.

Figure 4-1 characterizes the syntax of patterns. We can either create patterns of open maps (when building a nondeterministic function) or open embeddings (when building a necessarily deterministic continuous map).

Theorem 4.11. *For ? instantiated as either o (for open maps) or oe (for open embeddings), given any pattern derivation $p : A \dashv_? \Gamma$, there is a map $[\![p]\!] : \mathsf{Prod}(\Gamma) \times \Delta \to_? A$ for some space Δ which collects the "discarded variables" from the wildcards.*

Proof. By induction on the derivation of the pattern:

CONSTANT By assumption, we have a constant map of the right kind.

VAR We use $\mathsf{id} : A \to_{oe} A$ (which is also an open map).

WILDCARD We *also* use $\mathsf{id} : A \to_{oe} A$, but since we require $\Gamma \cong *$, we set the "garbage space" Δ to be A.

COMPOSE By induction, we have a map $[\![p]\!] : \Gamma \times \Delta \to_? U$ for some Δ. Then we use the composition $(f \otimes \mathsf{id}_\Delta) \circ [\![p]\!] : \Gamma \times \Delta \to_? A$, threading through the "garbage" space Δ.

PRODUCT By induction, we have maps $[\![p]\!] : \Gamma_1 \times \Delta_1 \to_? A$ and $[\![q]\!] : \Gamma_2 \times \Delta_2 \to_B$. Regardless of whether $? = o$ or $? = oe$, their parallel composition $[\![p]\!] \otimes [\![q]\!] : \Gamma_1 \times \Gamma_2 \to_? A \times B$ is also a map of the right kind, which we can use, together with some homeomorphisms to rearrange the "garbage" spaces Δ_1 and Δ_2 and to pass them through.

□

Proposition 4.12. *If $p : A \dashv_{oe} \Gamma$, then $p : A \dashv_{o} \Gamma$ as well.*

Proof. Straightforward, following from the fact that every open embedding is an open map. □

Note that the definition of patterns we give here does *not* admit some structural rules that may be expected, such as

Strengthening Given $p : A \dashv \Gamma$, we cannot necessarily derive $p : A \dashv \Delta$ where Δ is a sub-list of Γ.

Exchange Given $p : A \dashv \Gamma, \Delta$, we cannot necessarily derive $p : A \dashv \Delta, \Gamma$.

These structural properties aren't necessary since patterns are *providing* a context to be *used* by the expression language, which *does* have the corresponding rules weakening and exchange. Regardless, the WILDCARD rule still gives a sort of "explicit" strengthening, allowing the user to discard some variables.

We can now define a general (mostly) syntactic rule for "compiling" the sorts of pattern matches described in this section.

Theorem 4.13. *We can compile the syntax*[1]

$$\frac{\Gamma \vdash_? s : A \qquad \prod_{i:I} p_i : A_i \dashv_{¿} A \qquad \prod_{i:I} \Gamma, A_i \vdash_? e_i : B \qquad (p_i)_{i:I} \in \mathcal{J}_?}{\Gamma \vdash_? \left(\mathsf{case}(s) \left\{ [i : I] \quad p_i \quad \Rightarrow e_i \right\}\right) : B} \text{ CASE-?}$$

, where ? and ¿ are instantiated with any of the combinations in this table:

?	¿
	oe
p	*oe*
nd	*o*
nd, p	*o*

.

Proof. The syntactic constructions give us maps $[\![s]\!] : \Gamma \xrightarrow{?}_c A$, $[\![p_i]\!] : A_i \times \Delta_i \to_{¿} A$ (for some spaces Δ_i representing discarded variables in the pattern p_i), and $e_i : \Gamma \times A_i \xrightarrow{?}_c B$. The condition $(p_i)_{i:I} \in \mathcal{J}_?$ is interpreted to mean that these "compiled" maps appropriately cover A, so that we get an appropriately behaved map $\mathsf{inv} : A \xrightarrow{?}_c \sum_{i:I} A_i \times \Delta_i$.

We must produce a map $f : \Gamma \xrightarrow{?}_c B$. We can do so by defining[2]

$$\begin{aligned} &f : \Gamma \xrightarrow{?}_c B \\ &f(\gamma) \triangleq \text{ let } \langle i, x \rangle \triangleq \mathsf{inv}(s(\gamma)) \text{ in } e_i(\gamma, \mathsf{fst}(x)). \end{aligned}$$

[1] As mnemonics, I stands for *index* type, s stands for *scrutinee* of a case expression, p stands for *pattern*, and e for *expression*.

[2] While the "let-in" syntax for using the universal property (itself a sort of pattern matching) of sums has not been formally described, hopefully it is clear how it can be implemented via categorical semantics.

□

Note that the condition $(p_i)_{i:I} \in \mathcal{J}_?$ that the patterns lie in the appropriate Grothendieck topology is trivial when $? = (nd, p)$.

4.3 Pattern matching with "gluing" conditions

One limitation of the pattern matching constructions described in the previous section is that, when determinism is required, patterns are required to be disjoint (and if totality is also required, then under mild conditions each pattern must be clopen as well).

But we should be able to produce deterministic functions even when patterns overlap if we require some notion of compatibility of overlapping branches. The previous example of the partial Boolean or operation is such a case where the patterns overlap, but the function is still deterministic, because the outputs are compatible whenever there is overlap.

There is an analogous concept of a sort of "overlapping pattern matching" in sheaf theory, which effectively says that if branches are *identical* when they overlap, one can glue together the branches to form a single function. This is nice, but in fact we do not need to require that maps be identical when they overlap; we can get away with something more permissive. This section will characterize this more permissive notion of overlapping pattern matching that demands some sort of "compatibiliity" of overlapping branches.

First, let's briefly examine the similar concept from sheaf theory. Though we described four Grothendieck pretopologies in the previous section, we have yet to define perhaps the most popular Grothendieck pretopology treated in sheaf theory, which is the *open cover topology*.

Definition 4.14. *The* open cover topology *states that a collection of maps* $(p_i : A_i \hookrightarrow A)$ *covers the space* A *if each* p_i *is an open embedding and together their images cover* A*, i.e.,*

$$\top \leq \bigvee_{i:I} p_{i!}(\top).$$

Proposition 4.15. *The open cover topology is a Grothendieck pretopology.*

Proof. A minor variation of the proof of Proposition 4.8, which proved that if one also demands pairwise disjointness of covers, then one has a Grothendieck pretopology. □

The open cover topology is subcanonical, which implies that it is possible to define a continuous map $A \to_c B$ by splitting up A into a potentially overlapping collection of open spaces $(U_i)_{i:I}$ with an a cover $(p_i : U_i \hookrightarrow A)_{i:I}$ in the open cover topology and defining continuous maps $f_i : U_i \to_c B$, so long as the f_is agree on their overlap (which we will define more precisely soon).

Accordingly, we will restrict our attention to deterministic maps in this section, as the constructions in the previous section were sufficient in the nondeterministic case.

First, we'll consider the construction of a partial map.

To make the analysis easier, before handing the general syntactic case, we'll initially restrict to a simpler scenario, where we directly scrutinize the input, and then do not use the original input in any of the branches:

$$\begin{aligned} f &: A \xrightarrow{p}_c B \\ f(x) &\triangleq \mathsf{case}(x) \left\{ [i : I] \quad p_i(x_i) \quad \Rightarrow e_i(x_i) \quad , \right. \end{aligned}$$

where i ranges over the index type $I : \mathcal{U}$, each $p_i : A_i \hookrightarrow A$ is an open embedding, each $e_i : A_i \xrightarrow{p}_c B$ is an expression representing a deterministic but partial map, and each $x_i : A_i$ is a pattern variable binding. Note that each p_i must be an open embedding rather than just an open map, or else a single pattern on its own could introduce nondeterminism.

We can certainly define an implementation of the case expression that is potentially nondeterministic

$$\begin{aligned} f &: A \xrightarrow{nd,p}_c B \\ f &\triangleq \bigsqcup_{i:I} e_i \circ p_i^{-1}. \end{aligned}$$

We will now attempt to devise a condition on the *branches* (rather than any condition on the patterns) to ensure that f is actually deterministic, i.e., that f^* preserves binary meets. First, we define a notion of compatibility of continuous maps that is weaker than equality but strong enough to ensure that the map f is deterministic when overlapping branches are compatible.

Definition 4.16. *Two continuous maps $f, g : A \to_c B$ have a* maximum *if the map*

$$\begin{aligned} (f \sqcup g)^* &: \mathcal{O}(B) \to \mathcal{O}(A) \\ (f \sqcup g)^*(U) &\triangleq f^*(U) \vee g^*(U) \end{aligned}$$

preserves binary meets, which implies that it defines a continuous map $f \sqcup g : A \to_c B$.

Trivially, any map has a maximum with itself, $f \sqcup f = f$, so if two maps are equal, then they have a maximum.

Remark 4.17. *If $f, g : A \to_c B$ have a maximum, then for all $U, V : \mathcal{O}(B)$,*

$$f^*(U) \wedge g^*(V) \leq f^*(V) \vee g^*(U).$$

Assuming classical logic, this implies that for every (global) point x of A, either $f(x) \leq g(x)$ or $g(x) \leq f(x)$ (in terms of specialization order).

Proof. Since

$$(f \sqcup g)^*(U \wedge V) = (f \sqcup g)^*(U) \wedge (f \sqcup g)^*(V),$$

we have

$$\begin{aligned} f^*(U \wedge V) \vee g^*(U \wedge V) &= (f^*(U) \vee g^*(U)) \wedge (f^*(V) \vee g^*(V)) \\ &= (f^*(U) \wedge f^*(V)) \vee (f^*(U) \wedge g^*(V)) \\ &\quad \vee (g^*(U) \wedge f^*(V)) \vee (g^*(U) \wedge g^*(V)) \\ &= f^*(U \wedge V) \vee g^*(U \wedge V) \\ &\quad \vee (f^*(U) \wedge g^*(V)) \vee (g^*(U) \wedge f^*(V)) \end{aligned}$$

and thus

$$(f^*(U) \wedge g^*(V)) \vee (g^*(U) \wedge f^*(V)) \leq f^*(U \wedge V) \vee g^*(U \wedge V),$$

which implies that

$$\begin{aligned} f^*(U) \wedge g^*(V) &\leq f^*(U \wedge V) \vee g^*(U \wedge V) \\ &\leq f^*(V) \vee g^*(U). \end{aligned}$$

We now prove that assuming classical logic, for any point x of A, either $f(x) \leq g(x)$ or $g(x) \leq f(x)$. Suppose that $f(x) \not\leq g(x)$, which means there is some $U : \mathcal{O}(B)$ such that $f(x) \Vdash U$ (i.e., $x^*(f^*(U)) = \top$) but not $g(x) \Vdash U$ (i.e., $x^*(g^*(U)) = \bot$). Using the law we derived, we know (for all V)

$$x^*(f^*(U)) \wedge x^*(g^*(V)) \leq x^*(f^*(V)) \vee x^*(g^*(U)),$$

and thus

$$x^*(g^*(V)) \leq x^*(f^*(V)),$$

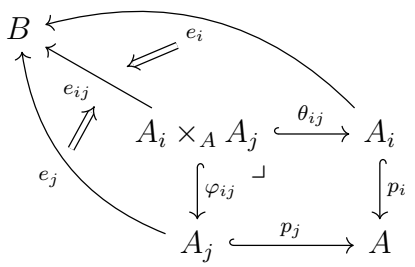
which means that $g(x) \leq f(x)$. □

Example. *The maps $t_1, t_2 : \mathbb{B} \to_c \Sigma$ defined by*

$$\begin{aligned} t_1(\mathsf{true}) &\triangleq \top_\Sigma & t_2(\mathsf{true}) &\triangleq \bot_\Sigma \\ t_1(\mathsf{false}) &\triangleq \bot_\Sigma & t_2(\mathsf{false}) &\triangleq \top_\Sigma \end{aligned}$$

have the maximum $f : \mathbb{B} \to_c \Sigma$ defined by $f(x) \triangleq \top_\Sigma$.

Theorem 4.18 (Determinism)**.** *If branches are "compatible" when they overlap: for each $i, j : I$, there is a map $e_{ij} : A_i \times_A A_j \to_c B$ that is the maximum of $e_i \circ \theta_{ij}$ and $e_j \circ \varphi_{ij}$ in the pullback diagram*



then the overlapping pattern match f is in fact deterministic, i.e., $f : A \xrightarrow{p}_c B$.

Proof. We must show f^* preserves binary meets. Since f^* preserves joins, it is monotone, and thus

$$f^*(U \wedge V) \leq f^*(U) \wedge f^*(V),$$

so it remains only to prove the reverse inequality.

Define $p_{ij} : A_i \times_A A_j \hookrightarrow A$ by $p_{ij} = p_j \circ \varphi_{ij} = p_i \circ \theta_{ij}$. By Proposition 3.13, we have

$$p_{ij!}(\top) = p_{i!}(\top) \wedge p_{j!}(\top).$$

We first note the equivalence

$$f^*(U) = \bigvee_{i:I} p_{i!}(e_i^*(U)) = \bigvee_{i,j:I} p_{ij!}(e_{ij}^*(U)).$$

That the middle expression is at most the right follows from the fact that $e_{ii} = e_i$, $p_{ii} = p_i$. The reverse inequality follows from the following inequality:

$$\begin{aligned}
p_{ij!}(e_{ij}^*(U)) &= p_{ij!}((e_i \circ \theta_{ij})^*(U) \vee (e_j \circ \varphi_{ij})^*(U)) && (e_{ij} \text{ “compatible”}) \\
&= p_{ij!}((e_i \circ \theta_{ij})^*(U)) \vee p_{ij!}(e_j \circ \varphi_{ij})^*(U)) && (p_{ij!} \text{ preserves joins}) \\
&= p_{i!}(\theta_{ij!}(\theta_{ij}^*(e_i^*(U)))) \vee p_{j!}(\varphi_{ij!}(\varphi_{ij}^*(e_j^*(U)))) && (\text{pullback equations}) \\
&= p_{i!}(\theta_{ij!}(\top) \wedge e_i^*(U)) \vee p_{j!}(\varphi_{ij!}(\top) \wedge e_j^*(U)) && (\text{Frobenius reciprocity}) \\
&\leq p_{i!}(e_i^*(U)) \vee p_{j!}(e_j^*(U)).
\end{aligned}$$

This inequality has a sort of partial converse:

$$\begin{aligned}
p_{ij!}(\top) \wedge p_{i!}(e_i^*(U)) &= p_{i!}(\theta_{ij!}(\top)) \wedge p_{i!}(e_i^*(U)) \\
&= p_{i!}(\theta_{ij!}(\top) \wedge e_i^*(U)) && (p_{i!} \text{ preserves meets}) \\
&\leq p_{i!}(\theta_{ij!}(\top) \wedge e_i^*(U)) \vee p_{j!}(\varphi_{ij!}(\top) \wedge e_j^*(U)) \\
&= p_{ij!}(e_{ij}^*(U)).
\end{aligned}$$

Note that the direct image map $p_{i!}$ (as well as the other direct image maps) preserves meets since p_i is an open embedding.

Putting these arguments together, we can confirm that f^* preserves meets:

$$\begin{aligned}
f^*(U) \wedge f^*(V) &= \left(\bigvee_{i:I} p_{i!}(e_i^*(U))\right) \wedge \left(\bigvee_{j:I} p_{j!}(e_j^*(V))\right) \\
&= \bigvee_{i,j:I} p_{i!}(e_i^*(U)) \wedge p_{j!}(e_j^*(V)) \\
&= \bigvee_{i,j:I} p_{i!}(\top) \wedge p_{i!}(e_i^*(U)) \wedge p_{j!}(\top) \wedge p_{j!}(e_j^*(V)) \\
&= \bigvee_{i,j:I} p_{ij!}(\top) \wedge p_{i!}(e_i^*(U)) \wedge p_{ij!}(\top) \wedge p_{j!}(e_j^*(V)) \\
&\leq \bigvee_{i,j:I} p_{ij!}(e_{ij}^*(U)) \wedge p_{ij!}(e_{ij}^*(V)) && \text{(previous argument)} \\
&= \bigvee_{i,j:I} p_{ij!}(e_{ij}^*(U \wedge V)) && (p_{ij!} \text{ preserves meets}) \\
&= f^*(U \wedge V)
\end{aligned}$$

□

It can be informative to consider the case where every open embedding is just an inclusion of an open sublocale (which is, up to isomorphism, just as general). Then the pullback just represents the intersection, noting the homeomorphism

$$\{A \mid p_{i!}(\top) \wedge p_{j!}(\top)\} \cong A_i \times_A A_j.$$

If we additionally want totality, it is easy to specify the conditions which guarantee that, which are effectively the same as in the previous section:

Proposition 4.19 (Totality)**.** *If each $e_i : A_i \to_c B$ is total, then the overlapping pattern match f is in fact total, i.e., $f : A \overset{nd}{\to}_c B$, if and only if the patterns together cover A, that is,*

$$\top \leq \bigvee_{i:I} p_{i!}(\top),$$

Proof. Given that each e_i is total, we calculate that

$$f^*(\top) = \bigvee_{i:I} p_{i!}(e_i^*(\top)) = \bigvee_{i:I} p_{i!}(\top)$$

And thus $f^*(\top) = \top$ if and only if the patterns together cover A. □

Note that we defined a nondeterministic map in the above definition. If we also add the previous condition on branches for determinism, we will get a total and deterministic (i.e., continuous) map.

More general syntax

We now return to the more general syntax, where the definition is of the form

$$\begin{aligned} f &: \Gamma \to_c B \\ f(\gamma) &\triangleq \mathsf{case}(s(\gamma)) \left\{ [i : I] \quad p_i(x_i) \quad \Rightarrow e_i(\gamma, x_i) \quad , \right. \end{aligned}$$

we had

$$f = \bigsqcup_{i:I} e_i \circ \langle \mathsf{id}, p_i^{-1} \circ s \rangle.$$

We can put in a more convenient form,

$$\begin{aligned} f &= \bigsqcup_{i:I} e_i \circ \langle \mathsf{id}, p_i^{-1} \circ s \rangle \\ &= \bigsqcup_{i:I} e_i \circ (\mathsf{id} \otimes p_i^{-1}) \circ \langle \mathsf{id}, s \rangle \\ &= \left(\bigsqcup_{i:I} e_i \circ (\mathsf{id} \otimes p_i^{-1}) \right) \circ \langle \mathsf{id}, s \rangle \\ &= \left(\bigsqcup_{i:I} e_i \circ (\mathsf{id} \otimes p_i)^{-1} \right) \circ \langle \mathsf{id}, s \rangle, \end{aligned}$$

where it becomes apparent that the left parenthesized term takes on the form of a "simpler" pattern match as we studied earlier,

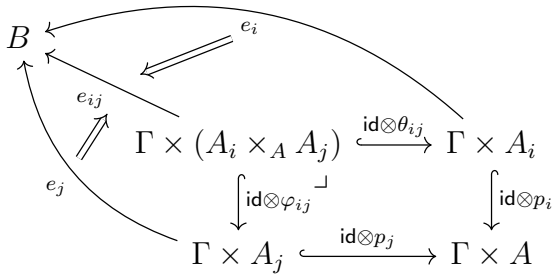
$$\begin{aligned} \tilde{f} &: \Gamma \times A \to_c B \\ \tilde{f}(z) &\triangleq \mathsf{case}(z) \left\{ [i : I] \quad \gamma, p_i(x_i) \quad \Rightarrow e_i(\gamma, x_i) \quad , \right. \end{aligned}$$

such that

$$f = \tilde{f} \circ \langle \mathsf{id}, s \rangle.$$

We can use this decomposition to give sufficient conditions for determinism in the general case.

Theorem 4.20 (Determinism)**.** *If branches are "compatible" when they overlap: for each $i, j : I$, there is a map $e_{ij} : \Gamma \times (A_i \times_A A_j) \to_c B$ that is the maximum of $e_i \circ (\mathsf{id} \otimes \theta_{ij})$ and $e_j \circ (\mathsf{id} \otimes \varphi_{ij})$ in the pullback diagram*

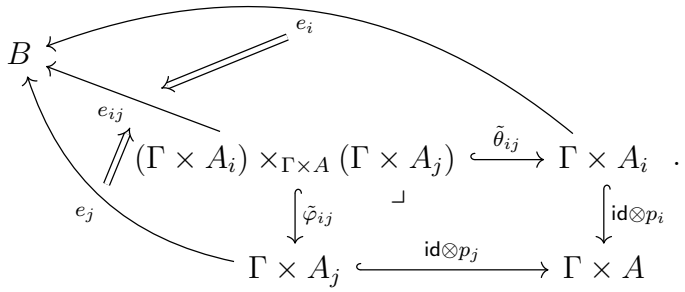


then the overlapping pattern match f is in fact deterministic, i.e., $f : \Gamma \xrightarrow{p}_c B$:

$$\frac{\Gamma \vdash_p s : A \qquad \prod_{i:I} p_i : A_i \dashv_{oe} A \qquad \prod_{i:I} \Gamma, A_i \vdash_p e_i : B \qquad \textit{pairwise branch compat.}}{\Gamma \vdash_p \left(\mathsf{case}(s) \left\{ [i : I] \quad p_i \quad \Rightarrow e_i \right) : B} \text{ CASE-P}$$

Proof. Since $f = \tilde{f} \circ \langle \mathsf{id}, s \rangle$ and since s is deterministic, it suffices to confirm that $\tilde{f}$ is deterministic.

We must only confirm that our diagram above is actually equivalent to the one that we would get for $\tilde{f}$ from Proposition 4.18,



It suffices to confirm that $\Gamma \times (A_i \times_A A_j)$ is indeed the pullback of $\mathsf{id} \otimes p_i$ and $\mathsf{id} \otimes p_j$. This follows from the fact that the pullback of "parallel" maps such as this is the product of the pullbacks, and that the pullback of id_Γ with itself is Γ. □

To define a continuous map that is total and deterministic, we simply combine the requirements:

$$\frac{\Gamma \vdash s : A \qquad \prod_{i:I} p_i : A_i \dashv_{oe} A \qquad \prod_{i:I} \Gamma, A_i \vdash e_i : B \qquad \top \leq \bigvee_{i:I} p_{i!}(\top) \qquad \text{pairwise branch compat.}}{\Gamma \vdash \left(\mathsf{case}(s) \left\{ [i : I] \quad p_i \quad \Rightarrow e_i \right) : B} \text{ CASE}$$

4.4 Properties of overlapping pattern matching

4.4.1 Partial patterns and lifted spaces

It's possible to see the conditions for totality of overlapping patterns in another way. There is a natural correspondence between a partial pattern match

$$f : A \xrightarrow{p}_c B$$
$$f(x) \triangleq \mathsf{case}(x) \left\{ [i : I] \quad p_i(x_i) \quad \Rightarrow e_i(x_i) \right.$$

where each $e_i : A_i \xrightarrow{p}_c B$ and a total pattern match that instead outputs to a lifted space,

$$\begin{aligned} f &: A \to_c B_\perp \\ f(x) &\triangleq \mathsf{case}(x) \begin{cases} [i : I] & p_i(x_i) & \Rightarrow e_i(x_i) \\ & \_ & \Rightarrow \perp_B \end{cases} \end{aligned}$$

where here we consider each $e_i : A_i \to_c B_\perp$ as a total map to the lifted space.

We call the additional case a catch-all case. The catch-all case already suffices to cover the whole space, of course, trivially showing that the totality condition holds for such a transformed pattern match. Since $\perp_B$ has maxima with all points of B, the additional determinism conditions induced by adding the catch-all case are satisfied automatically.

It should be apparent that these two pattern match definitions are equivalent.

4.4.2 Related to Grothendieck pretopologies

As the various pattern families we have considered thus far all form Grothendieck pretopologies, all pattern matches have certain properties. For instance, the facts that homeomorphisms alone are covers and covers can be composed each correspond to constructions involving overlapping patterns.

Any homeomorphism is a cover by itself ("reflexivity"), corresponding to a pattern that may change the "view" of data to an isomorphic space (without breaking it apart). For instance, given the homeomorphism $\mathsf{exp} : \mathbb{R} \hookrightarrow \{\mathbb{R} \mid \cdot > 0\}$, we can define multiplication on $\{\mathbb{R} \mid \cdot > 0\}$ using addition on $\mathbb{R}$ with the pattern match

$$\begin{aligned} \mathsf{mult} &: \{\mathbb{R} \mid \cdot > 0\} \times \{\mathbb{R} \mid \cdot > 0\} \to_c \{\mathbb{R} \mid \cdot > 0\} \\ \mathsf{mult}(p) &\triangleq \mathsf{case}(p) \left\{ \mathsf{exp}(x), \mathsf{exp}(y) \quad \Rightarrow \mathsf{exp}(x + y) \right. . \end{aligned}$$

There is also a notion of composition of covers ("transitivity"), which corresponds to the potential to flatten nested patterns. For instance, the definition

$$\begin{aligned} f &: A_\perp + B_\perp \xrightarrow{p}_c A + B \\ f(x) &\triangleq \mathsf{case}(x) \begin{cases} \mathsf{inl}(\ell) & \Rightarrow \mathsf{case}(\ell) \left\{ \mathsf{up}(\ell') \Rightarrow \mathsf{inl}(\ell') \right. \\ \mathsf{inr}(r) & \Rightarrow \mathsf{case}(r) \left\{ \mathsf{up}(r') \Rightarrow \mathsf{inr}(r') \right. \end{cases} \end{aligned}$$

can be flattened into

$$f(x) \triangleq \mathsf{case}(x) \begin{cases} \mathsf{inl}(\mathsf{up}(\ell)) & \Rightarrow \mathsf{inl}(\ell) \\ \mathsf{inr}(\mathsf{up}(r)) & \Rightarrow \mathsf{inr}(r) \end{cases} .$$

Pulling back covers

Since covering families of open embeddings form a Grothendieck pretopology, given a cover $(p_i : B_i \hookrightarrow B)_{i:I}$ of B and a map $f : A \to_c B$, the family

$$f^*p_i : \{A \mid f^*(p_{i!}(\top))\} \hookrightarrow A \qquad (i : I)$$

of open embeddings pulled back along f covers B, where we have the pullback diagram

$$\begin{array}{ccc} \{A \mid f^*(p_{i!}(\top))\} & \xhookrightarrow{f^*p_i} & A \\ \downarrow & \lrcorner & \downarrow f \\ B_i & \xhookrightarrow{p_i} & B \end{array}.$$

By using pulled-back covers, it is possible to pattern match on an input $x : A$ by doing a case analysis on $f(x) : B$, such that in each branch it is known that x lies in a particular open subspace of A (rather than only knowing that $f(x)$ lies in a particular open subspace of B). The introductory example, as explained in section 5.4, uses a pulled-back cover in this way.

4.4.3 Evaluation of patterns

The purpose of this section is to explain how overlapping pattern matches behave under evaluation (i.e., substitution or composition). Intuitively, when we evaluate an overlapping pattern match on a value x, we collect those patterns that x falls into, and the result is the nondeterministic union of the branches corresponding to that pattern. We will now make this notion formal.

A map $f : A \overset{?}{\to}_c B$ (for $? \in \{c, nd, p, (nd, p)\}$) defined by the pattern match

$$f(x) \triangleq \mathsf{case}(x) \left\{ [i : I] \quad p_i(x_i) \quad \Rightarrow \quad e_i(x_i) \right.$$

where each $p_i : A_i \to_o A$ and $e_i : A_i \overset{?}{\to}_c B$, when evaluated on a (generalized) point $x : \Gamma \overset{?}{\to}_c A$, can be described by

$$(f \circ x)(\gamma) = \mathsf{case}(\gamma) \left\{ [i : I] \quad f^*p_i(\gamma_i) \quad \Rightarrow \quad e_i(x_i(\gamma_i)), \right.$$

where each f^*p_i and x_i come from the pullback diagram

$$\begin{array}{ccc} \{\Gamma \mid f^*(p_{i!}(\top))\} & \xhookrightarrow{f^*p_i} & \Gamma \\ \downarrow x_i & \lrcorner & \downarrow x \\ A_i & \xhookrightarrow{p_i} & A \end{array}.$$

In particular, suppose for each $i : I$, the open

$$f^*(p_{i!}(\top)) : \mathcal{O}(\Gamma)$$

is either $\top$ or $\bot$. If x is a global point, meaning $\Gamma \cong *$, then classically this is the case. Let $I_\top$ and $I_\bot$ represent the indices where the opens are $\top$ and $\bot$, respectively, and for $i : I_\top$, let $x_{A_i} : \Gamma \to_c \{A \mid p_{i!}(\top)\}$ be x with a restricted range. Then for any $U : \mathcal{O}(B)$,

$$\begin{aligned}(f \circ x)^*(U) &= \bigvee_{i:I} (e_i \circ x_i)^*(U) \\ &= \bigvee_{i:I_\top} (e_i \circ x_i)^*(U) \\ &= \bigvee_{i:I_\top} (e_i \circ p_i^{-1} \circ x_{A_i})^*(U)\end{aligned}$$

These arguments confirm the intuitive explanation about evaluation of overlapping patterns: collect the possible branches, apply them to the point, and the behavior will be the nondeterministic union of those results.

For instance, in section 4.1.3, we found that the partial Boolean or satisfies

$$\mathsf{or}(\mathsf{up}(\mathsf{true}), \bot) = \mathsf{true}.$$

We confirm this by "pulling back" the pattern match defining or on this input to

$$\mathsf{or}(\mathsf{up}(\mathsf{true}), \bot) = \mathsf{case}(\mathsf{tt}) \begin{cases} \_ & \Rightarrow \mathsf{true} \\ \mathsf{abort}(x) & \Rightarrow \mathsf{abort}(x) \\ \mathsf{abort}(x) & \Rightarrow \mathsf{abort}(x) \\ \_ & \Rightarrow \bot \end{cases},$$

where for any space A, $\mathsf{abort} : \varnothing \hookrightarrow A$ is the trivial embedding of the empty space $\varnothing$. Patterns that match the scrutinee reduce to the embedding $\mathsf{id} : * \hookrightarrow *$ (the wildcard pattern), and patterns that it misses entirely reduce to abort. We can then collect only the possible branches to conclude that

$$\mathsf{or}(\mathsf{up}(\mathsf{true}), \bot) = \bot \sqcup \mathsf{true} = \mathsf{true}.$$

Chapter 5

Binary covers: nondeterministic decision procedures

In this section, we develop a theory of *binary covers*, which are pairs of opens that cover a given space. These binary covers are useful for computing approximate decisions on spaces, where computing exact decisions might not be possible. For instance, $\mathbb{R}$ has no nontrivial decidable predicates since it is connected, but it admits a wealth of binary covers.

The basic idea is to generalize the notions of exhaustive reasoning from general functional programming, where it is possible to compute whether or not decidable propositions hold. There is a calculus of decidable predicates: they are closed under conjunction, disjunction, and negation, and universal and existential quantification over finite[1] sets yields decidable predicates as well. We will generalize this calculus to binary covers, with finite sets generalized to *compact/overt* spaces.

We can relax the notion of decidable predicates to binary covers. Whereas a decidable proposition P is one where either P or $\neg P$ holds, a binary cover is a pair of propositions P and Q such that either P or Q holds. In a binary cover, it is possible that P and Q *both* hold, which is of course never the case for P and $\neg P$ in decidable propositions. So every decidable proposition can be interpreted as a binary cover, but not the other way around.

Translating to spatial language, a decidable predicate on a space A is an open P of A such that there is a disjoint open $\neg P$ satisfying $\top \leq P \vee \neg P$ (equivalently, P is *clopen*). A binary cover is a pair of opens P and Q that cover the space, i.e., $\top \leq P \vee Q$. For example, the $\mathsf{positive}_\varepsilon$ program in Section 4.1.2 covers $\mathbb{R}$ with the pair of opens $\cdot > -\varepsilon$ and $\cdot < \varepsilon$. More generally, any overlapping pattern match with two cases specifies a binary cover of the input space.

5.1 Binary covers as nondeterministic decisions

Definition 5.1. *A* binary cover *of a space A is an ordered pair (P, Q) of opens of A satisfying $\top \leq P \vee Q$.*

[1] By "finite," we always mean Kuratowski-finite [14].

Proposition 5.2. *Binary covers of A are in bijective correspondence with nondeterministic Boolean-valued maps $A \overset{nd}{\to}_c \mathbb{B}$.*

Proof. Suppose we have a binary cover (P, Q) of A. Then define $f^* : \mathcal{O}(\mathbb{B}) \to \mathcal{O}(A)$ by $f^*(\cdot = \mathsf{true}) \triangleq P$ and $f^*(\cdot = \mathsf{false}) \triangleq Q$. To confirm f^* defines a nondeterministic map $f : A \overset{nd}{\to}_c \mathbb{B}$, we must confirm f^* preserves joins and $\top$. Since $\mathbb{B}$, as a discrete space, has no covering axioms, f^* trivially preserves joins. And f^* also preserves $\top$, since

$$f^*(\top) = f^*((\cdot = \mathsf{true}) \vee (\cdot = \mathsf{false})) = P \vee Q = \top.$$

Conversely, given a nondeterministic map $f : A \overset{nd}{\to}_c B$, we get a binary cover of A given by

$$(f^*(\cdot = \mathsf{true}), f^*(\cdot = \mathsf{false})),$$

which we confirm is covering since

$$f^*(\cdot = \mathsf{true}) \vee f^*(\cdot = \mathsf{false}) = f^*((\cdot = \mathsf{true}) \vee (\cdot = \mathsf{false})) = f^*(\top) = \top.$$

□

This establishes two alternative perspectives on approximate decision-making, one spatial and one more algorithmic in flavor. The algorithmic one makes clear that we should be able to use Boolean operations to combine binary covers into new ones, just as we can perform Boolean operations on the decidable predicates. This will yield a language for approximate decision procedures.

The nondeterminism means that it won't behave quite like the deterministic decision procedures. We can characterize this algebraically. In **Set**, $\mathbb{B}$ forms a Boolean algebra. Since the functor $\mathsf{Discrete} : \mathbf{Set} \to \mathbf{FSpc}$ is full, faithful, and preserves binary products and the terminal object, this lifts to an internal Boolean algebra (within **FSpc**) on $\mathbb{B}$ the space. We can again lift these operations by the functor $\mathsf{NonDet} : \mathbf{FSpc} \to \mathbf{FSpc}_{nd}$; for instance, the lifted version of the Boolean "and" operation ($\&\& : \mathbb{B} \times \mathbb{B} \overset{nd}{\to}_c \mathbb{B}$) is potentially true if its argument potentially takes on values whose conjunction is equal to true.

Proposition 5.3. *However, the operations on $\mathbb{B}$ in $\boldsymbol{FSpc}_{nd}$ no longer form a Boolean algebra.*

Proof. In particular, there is the nondeterministic value $\mathsf{both} : * \overset{nd}{\to}_c \mathbb{B}$ satisfying

$$\mathsf{both}^*(\cdot = \mathsf{true}) = \mathsf{both}^*(\cdot = \mathsf{false}) = \top,$$

such that

$$\mathsf{both} \,||\, !\,\mathsf{both} = \mathsf{both} \neq \mathsf{true},$$

whereas in a Boolean algebra there is the identity $x \,||\, !\,x = \mathsf{true}$. □

Though $\mathbb{B}$ doesn't form a Boolean algebra in $\mathbf{FSpc}_{nd}$, it comes close!

Proposition 5.4. *The space $\mathbb{B}$ forms a quasi-Boolean algebra (or de Morgan algebra) in $\boldsymbol{FSpc}_{nd}$, meaning that $\mathbb{B}$ with $\&\&, ||, \mathsf{true}$, and false forms a bounded distributive lattice, and $!$ is a de Morgan involution, in that it satisfies $!!x = x$ and $!(x \,\&\&\, y) = !\,x \,||\, !\,y$.*

Proof. It is instructive to observe how the operations act on generalized points $\Gamma \overset{nd}{\to}_c \mathbb{B}$; we will use their equivalent representation as binary covers of Γ. Observe that

$$\begin{aligned}(P_1, Q_1) \,\&\&\, (P_2, Q_2) &= (P_1 \wedge P_2, Q_1 \vee Q_2)\\ (P_1, Q_1) \,||\, (P_2, Q_2) &= (P_1 \vee P_2, Q_1 \wedge Q_2)\\ !(P, Q) &= (Q, P)\\ \mathsf{true} &= (\top, \bot)\\ \mathsf{false} &= (\bot, \top).\end{aligned}$$

We can use this to confirm the various laws, for instance, that true is the identity for $\&\&$,

$$\mathsf{true} \,\&\&\, (P, Q) = (\top \wedge P, \bot \vee Q) = (P, Q).$$

We similarly can confirm false is the identity for $||$, and it is easy to observe that $\&\&$ and $||$ are commutative and associative. Absorption of $\&\&$ and $||$ follows from the similar absorption properties of opens, and likewise for their distributivity properties.

It remains to confirm that $!$ is a deMorgan involution. We have

$$!\,!(P, Q) = !(Q, P) = (P, Q)$$

and

$$\begin{aligned}!((P_1, Q_1) \,\&\&\, (P_2, Q_2)) &= !(P_1 \wedge P_2, Q_1 \vee Q_2) = (Q_1 \vee Q_2, P_1 \wedge P_2)\\ &= (Q_1, P_1) \,||\, (Q_2, P_2) = !(P_1, Q_2) \,||\, !(P_2, Q_2).\end{aligned}$$

□

That these nondeterministic Boolean operations form a quasi-Boolean algebra is not very interesting on its own. What makes truly binary covers powerful is that they admit quantification over compact/overt spaces, which is what we will now show.

5.2 Quantification over compact/overt spaces

When working with sets, if a predicate P on a set A is decidable and if A is finite, then $\forall a : A.\ P(a)$ and $\exists a : A.\ P(a)$ are decidable as well. The spatial analogue of the finite sets are the compact/overt spaces.

5.2.1 On compact/overt spaces

Definition 5.5. *A space A is* compact *if for every space Γ, the functor $- \times \top_A : \mathcal{O}(\Gamma) \to \mathcal{O}(\Gamma \times A)$ has a right adjoint $\forall_A : \mathcal{O}(\Gamma \times A) \to \mathcal{O}(\Gamma)$. That is, for each*

open $U : \mathcal{O}(\Gamma \times A)$*, there is an open* $\forall_A U : \mathcal{O}(\Gamma)$ *such that for every* $V : \mathcal{O}(\Gamma)$*,*

$$V \leq_\Gamma \forall_A U \qquad \iff \qquad V \times \top_A \leq_{\Gamma \times A} U.$$

Similarly, a space A *is* overt *if for every space* Γ*,* $- \times \top_A$ *has a* left *adjoint* $\exists_A : \mathcal{O}(\Gamma \times A) \to \mathcal{O}(\Gamma)$*. That is, for every open* $U : \mathcal{O}(\Gamma \times A)$*, there is an open* $\exists_A U : \mathcal{O}(\Gamma)$ *such that for every* $V : \mathcal{O}(\Gamma)$*,*

$$\exists_A U \leq_\Gamma V \qquad \iff \qquad U \leq_{\Gamma \times A} V \times \top_A.$$

These conditions are the definitions of universal and existential quantification in terms of adjoints, viewing Γ as some context and opens as truth values in a context. This definition of compactness is equivalent to the more common one, that every open cover has finite subcover.

Definition 5.6. *A* compact/overt *space is a space* A *that is both compact and overt, as well as satisfying an additional property, that says for all* Γ *and all* $P, Q : \mathcal{O}(\Gamma \times A)$*, we have*

$$\forall_A(P \vee Q) \leq_\Gamma \forall_A P \vee \exists_A Q$$
$$\forall_A P \wedge \exists_A Q \leq_\Gamma \exists_A(P \wedge Q).$$

Assuming classical logic, all spaces are overt, so it is acceptable intuition (spatially) to conflate the compact/overt spaces with the compact ones.

These properties allow us to define syntax for quantification of Σ-valued continuous maps on compact/overt spaces:

$$\frac{\Gamma, x : A \vdash e : \Sigma \qquad A \text{ compact}}{\Gamma \vdash \forall x \in A.\ e : \Sigma}\ \forall\text{-}\Sigma\text{-FORM} \qquad \frac{\Gamma, x : A \vdash e : \Sigma \qquad A \text{ overt}}{\Gamma \vdash_{nd} \exists x \in A.\ e : \Sigma}\ \exists\text{-}\Sigma\text{-FORM}$$

These properties allow us to quantify binary covers over compact/overt spaces, too. That is, we can add some syntax

$$\frac{\Gamma, x : A \vdash_{nd} e : \mathbb{B} \qquad A \text{ compact/overt}}{\Gamma \vdash_{nd} \forall x \in A.\ e : \mathbb{B}}\ \forall\text{-}\mathbb{B}\text{-FORM}$$

$$\frac{\Gamma, x : A \vdash_{nd} e : \mathbb{B} \qquad A \text{ compact/overt}}{\Gamma \vdash_{nd} \exists x \in A.\ e : \mathbb{B}}\ \exists\text{-}\mathbb{B}\text{-FORM}$$

that behaves as we'd expect (for a quasi-Boolean algebra, at least). We compile this syntax by defining quantification functionals of the type $(\Gamma \times A \overset{nd}{\to}_c \mathbb{B}) \to (\Gamma \overset{nd}{\to}_c \mathbb{B})$.

For a compact/overt space A, we define a universal quantification functional

$$\forall_A : (\Gamma \times A \overset{nd}{\to}_c \mathbb{B}) \to (\Gamma \overset{nd}{\to}_c \mathbb{B})$$
$$\forall_A(P, Q) \triangleq (\forall_A P, \exists_A Q),$$

and confirm that this pair of opens is indeed covering with the derivation

$$\begin{aligned} \top_\Gamma &\leq \forall_A(\top_{\Gamma\times A}) && (\forall_A \text{ adjointness}) \\ &\leq \forall_A(P \vee Q) && ((P,Q) \text{ cover } \Gamma \times A,\ \forall_A \text{ monotone}) \\ &\leq \forall_A P \vee \exists_A Q. && (A \text{ compact/overt}) \end{aligned}$$

We can similarly define the existential quantification functional

$$\begin{aligned} \exists_A : (\Gamma \times A \overset{nd}{\to}_c \mathbb{B}) &\to (\Gamma \overset{nd}{\to}_c \mathbb{B}) \\ \exists_A(P,Q) &\triangleq (\exists_A P, \forall_A Q), \end{aligned}$$

which indeed defines a binary cover for the same reason as $\forall_A$. By inspection, the two quantifiers are related by the law

$$\neg(\forall_A f) = \exists_A(\neg f).$$

While intuitively it seems that these operations indeed compute some nondeterministic quantification over compact spaces, we can make this statement precise by showing that these quantification functionals are adjoints to a weakening functional. To do this, we must first define an order on binary covers that relates their degrees of truth.

The quasi-Boolean algebra structure determines a preorder that can be placed on binary covers of any space. For binary covers (P_1, Q_1) and (P_2, Q_2) of a space Γ, we say $(P_1, Q_1) \leqq (P_2, Q_2)$ if and only if both $P_1 \leq P_2$ and $Q_2 \leq Q_1$. We will call this *truth order*. (Note that truth order is different from the specialization order.)

For any spaces Γ and A, the weakening operation,

$$(- \circ \mathsf{fst}) : (\Gamma \overset{nd}{\to}_c \mathbb{B}) \to (\Gamma \times A \overset{nd}{\to}_c \mathbb{B}),$$

which acts according to

$$(- \circ \mathsf{fst})(U,V) = (U \times \top, V \times \top),$$

is monotone with respect to truth order. Now we can show that the quantification operators on binary covers deserve their names.

Theorem 5.7. *The existential and universal quantification functionals are left and right adjoints to weakening, respectively, with respect to truth order, i.e.,*

$$\exists_A \dashv (- \circ \mathsf{fst}) \dashv \forall_A.$$

Proof. First we prove that $\exists_A$ is left adjoint to weakening: given (P,Q) a binary cover

of $\Gamma \times A$ and (U, V) a binary cover of Γ, we must show

$$\begin{array}{rcl} (P,Q) \leqq (- \circ \mathsf{fst})(U,V) & \text{if and only if} & \exists_A(P,Q) \leqq (U,V) \\ (P,Q) \leqq (U \times \top, V \times \top) & \text{if and only if} & (\exists_A P, \forall_A Q) \leqq (U,V) \\ P \leq U \times \top \text{ and } V \times \top \leq Q & \text{if and only if} & \exists_A P \leq U \text{ and } V \leq \forall_A Q. \end{array}$$

In the final form, we see that this follows from adjoint properties of the $\exists_A$ and $\forall_A$ operations that act on opens of $\Gamma \times A$.

The proof that $\forall_A$ is a right adjoint is a mirror image of the proof regarding $\exists_A$. □

5.2.2 On compact/overt subspaces

Sometimes, the space that we might want to quantify over could depend on some continuous variables in the context. For instance, we may want to quantify a binary cover $f : \mathbb{R} \times \mathbb{R} \overset{nd}{\to}_c \mathbb{B}$ over the unit simplex in $\mathbb{R} \times \mathbb{R}$ (i.e., the triangle bounded by $(0,0)$, $(1,0)$, and $(0,1)$). We will describe a formalism whereby it will be possible to write this as

$$\forall x \in [0,1].\ \forall y \in [0, 1-x].\ f(x,y).$$

We handle this situation by considering certain spaces whose points represent compact/overt subspaces of some space.

First, we note the connection between the overt spaces and $\mathcal{P}_\Diamond$, which represents (some) overt subspaces of a space A as points of the powerspace $\mathcal{P}_\Diamond(A)$.

Proposition 5.8 (Theorem 32 of [43])**.** *Every point of $\mathcal{P}_\Diamond(A)$ corresponds to an overt subspace of A.*[2]

Similarly, for each space A there is a powerspace $\mathcal{P}_\Box(A)$ whose points correspond with compact subspaces of A. Vickers describes the construction in detail [45, 40], but we summarize the salient characteristics. There is a "necessity" modality $\Box : \mathcal{O}(A) \to \mathcal{O}(\mathcal{P}_\Box(A))$ that distributes over meets and directed joins (analogous to the "possibility" modality $\Diamond : \mathcal{O}(A) \to \mathcal{O}(\mathcal{P}_\Diamond(A))$ for the lower powerspace). Continuous maps $\Gamma \to_c \mathcal{P}_\Box(A)$ are in bijective correspondence with inverse image maps $\mathcal{O}(A) \to \mathcal{O}(\Gamma)$ that preserve meets and directed joins. Like $\mathcal{P}_\Diamond$, $\mathcal{P}_\Box$ is a strong monad.

The powerspace analogue of the compact/overt spaces is called the *Vietoris powerspace* $\mathcal{P}_{\Box\Diamond}$. Points of $\mathcal{P}_{\Box\Diamond}(A)$ correspond to compact/overt subspaces of A. The space $\mathcal{P}_{\Box\Diamond}$ has both a "possibility" modality $\Diamond : \mathcal{O}(A) \to \mathcal{O}(\mathcal{P}_{\Box\Diamond}(A))$ a "necessity" modality $\Box : \mathcal{O}(A) \to \mathcal{O}(\mathcal{P}_{\Box\Diamond}(A))$ that interact exactly as with the compact/overt spaces; that is, for any opens $P, Q : \mathcal{O}(A)$, the following laws hold:

$$\Box(P \vee Q) \leq \Box P \vee \Diamond Q \tag{5.1}$$

$$\Box P \wedge \Diamond Q \leq \Diamond(P \wedge Q). \tag{5.2}$$

[2] Specifically, the points of $\mathcal{P}_\Diamond(A)$ are in bijective correspondence with the *weakly closed* (defined in [43]) overt subspaces of A.

Like for the lower powerspaces, there is the "positive" subspace $\mathcal{P}^{+}_{\Box\Diamond}(A)$ of $\mathcal{P}_{\Box\Diamond}(A)$ that additionally satisfies $\top \leq \Diamond\top$ and $\Box\bot \leq \bot$.

We can add some additional syntax to make it easier to describe opens (via the correspondence between Σ-valued maps $\Gamma \to_c \Sigma$ and opens $\mathcal{O}(\Gamma)$) with these modalities:

$$\frac{\Gamma \vdash s : \mathcal{P}_{\Diamond}(A) \qquad \Gamma, x : A \vdash e : \Sigma}{\Gamma \vdash \exists x \in s.\ e : \Sigma}\ \exists\text{-}\Sigma\text{-SUB-FORM}$$

$$\frac{\Gamma \vdash s : \mathcal{P}_{\Box}(A) \qquad \Gamma, x : A \vdash e : \Sigma}{\Gamma \vdash \forall x \in s.\ e : \Sigma}\ \forall\text{-}\Sigma\text{-SUB-FORM}\ .$$

These are compiled using the correspondence [37]

$$\Sigma \cong \mathcal{P}_{\Diamond}(*) \cong \mathcal{P}_{\Box}(*).$$

Replacing Σ with $\mathcal{P}_{\Diamond}(*)$ in $\exists$-Σ-SUB-FORM and with $\mathcal{P}_{\Box}(*)$ in $\forall$-Σ-SUB-FORM, these rules just correspond to the bind operation of a strong monad M, which composes maps $\Gamma \to M(A)$ and $\Gamma \times A \to M(B)$ to produce $\Gamma \to M(B)$.

These properties allow us to quantify binary covers over compact/overt subspaces as well, implementing the syntax

$$\frac{\Gamma \vdash s : \mathcal{P}_{\Box\Diamond}(A) \qquad \Gamma, x : A \vdash_{nd} e : \mathbb{B}}{\Gamma \vdash_{nd} \forall x \in s.\ e : \mathbb{B}}\ \forall\text{-}\mathbb{B}\text{-SUB-FORM}$$

$$\frac{\Gamma \vdash s : \mathcal{P}_{\Box\Diamond}(A) \qquad \Gamma, x : A \vdash_{nd} e : \mathbb{B}}{\Gamma \vdash_{nd} \exists x \in s.\ e : \mathbb{B}}\ \exists\text{-}\mathbb{B}\text{-SUB-FORM}\ .$$

Given any binary cover $f : \Gamma \times A \overset{nd}{\to}_c \mathbb{B}$ of A, we define its universal quantification

$$\begin{aligned}
&\forall_{[\cdot]} f : \Gamma \times \mathcal{P}_{\Box\Diamond}(A) \overset{nd}{\to}_c \mathbb{B} \\
&\forall_{[\cdot]} f \triangleq \mathsf{case}(p) \begin{cases} \iota[\lambda(\gamma, s).\ \forall x \in s.\ f(\gamma, x) = \mathsf{true}]\,(_) & \Rightarrow \mathsf{true} \\ \iota[\lambda(\gamma, s).\ \exists x \in s.\ f(\gamma, x) = \mathsf{false}]\,(_) & \Rightarrow \mathsf{false} \end{cases}.
\end{aligned}$$

We can confirm this pattern match is covering, i.e.,

$$\top \leq (\lambda(\gamma, s).\ \forall x \in s.\ f(\gamma, x) = \mathsf{true}) \vee (\lambda(\gamma, s).\ \exists x \in s.\ f(\gamma, x) = \mathsf{false})$$

with the derivation[3]

$$\begin{aligned}
\top &\leq \lambda(\gamma, s).\ \forall x \in s.\ \top && (\Box \text{ preserves } \top)\\
&\leq \lambda(\gamma, s).\ \forall x \in s.\ f(x) = \mathsf{true} \vee f(x) = \mathsf{false}\\
&\leq \lambda(\gamma, s).\ (\forall x \in s.\ f(x) = \mathsf{true}) \vee \exists x \in s.\ f(x) = \mathsf{false}. && (\text{law } 5.1)
\end{aligned}$$

We can define $\exists_{[.]} f : \mathcal{P}_{\Box\Diamond}(A) \overset{nd}{\to}_c \mathbb{B}$ by composing $\forall_{[.]} f$ with Boolean negation.

Compact/overt spaces form a convenient class of spaces over which exhaustive reasoning is possible. The continuous image of a compact/overt space is compact/overt (similar to the fact that the image of a finite set under any map is finite). Like finite subsets, compact/overt subspaces are closed under finitary union, witnessed by a construction

$$\cup : \mathcal{P}_{\Box\Diamond}(A) \times \mathcal{P}_{\Box\Diamond}(A) \to_c \mathcal{P}_{\Box\Diamond}(A),$$

but not necessarily intersection. Naturally, a finite set viewed as a discrete space is compact/overt.

An alternate understanding of binary covers

An alternative interpretation of binary covers helps explain why the two opens in a binary cover get "opposite" treatments in the definitions of conjunction and disjunction. Rather than thinking of a binary cover of A as two open sets $P, Q : \mathcal{O}(A)$ such that (informally) $A \subseteq P \cup Q$, we can instead think of it as an open set P and a *closed* set $\overline{Q}$ which is the "set-theoretic" complement of Q, since the complement of an open set is closed. Then the fact that (P, Q) is a binary cover means that $\overline{Q} \subseteq P$. Then conjunction computes intersections of the open and closed subspaces, while disjunction computes unions. Since we are simply "encoding" closed sets with their open complements, computing the "union" of closed subspaces just corresponds to taking an intersection of their open representatives. Hence, conjunction computes intersections of the second components.

This elicits the view of binary covers as "approximate" predicates, sandwiching a closed subspace inside an open one, with wiggle room for points in between. Any points in $\overline{Q}$ (and thus also P) will definitely compute to true, while any points outside of P (and thus also outside $\overline{Q}$) will definitely compute to false, while in-between points, which are in P but not $\overline{Q}$, are allowed to compute either way.

[3] The algebraic manipulations in this derivation are justified by the corresponding ones on $\mathcal{P}_{\Box\Diamond}(A)$. For instance $\top \leq \Box\top$ implies

$$\lambda(\gamma, s).\ \top \quad \leq \quad \lambda(\gamma, s).\ \forall x \in s.\ \top.$$

5.3 Binary covers on $\mathbb{R}$

Here, we will describe some compact/overt subspaces and binary covers of $\mathbb{R}$ and use these to solve some approximate decision problems on $\mathbb{R}$ with the calculus of binary covers.

There is a construction

$$\mathsf{HeineBorel} : \{(a,b) : \mathbb{R} \times \mathbb{R} \mid a < b\} \to_c \mathcal{P}^+_{\Box\Diamond}(\mathbb{R})$$

that demonstrates for endpoints (a,b) that the closed interval from a to b is compact/overt and inhabited (a constructive analogue of the Heine-Borel theorem) [46].

We can define binary covers that give approximate order comparisons on $\mathbb{R}$ by defining, for each $\varepsilon > 0$,

$$\begin{aligned}
[\cdot < \cdot]_\varepsilon &: \mathbb{R} \times \mathbb{R} \overset{nd}{\to}_c \mathbb{B} \\
[x < y]_\varepsilon &\triangleq \mathsf{case}(x,y) \begin{cases} \iota[\lambda(x,y).\ x < y]\,(_) & \Rightarrow \mathsf{true} \\ \iota[\lambda(x,y).\ x > y - \varepsilon]\,(_) & \Rightarrow \mathsf{false}. \end{cases}
\end{aligned}$$

We can use this to approximately query whether an arbitrary continuous map $f : \mathbb{R} \to_c \mathbb{R}$ has no roots on a compact/overt subspace of $\mathbb{R}$ (such as a closed interval), with the binary cover

$$\begin{aligned}
\mathsf{no_roots}_\varepsilon &: \mathcal{P}_{\Box\Diamond}(\mathbb{R}) \overset{nd}{\to}_c \mathbb{B} \\
\mathsf{no_roots}_\varepsilon(s) &\triangleq \forall x \in s.\ [0 < |f(x)|]_\varepsilon.
\end{aligned}$$

If we observe true of the result on input s, then indeed f has no roots on s, while if we observe false of the result, then there is some $x \in s$ such that $|f(x)| < \varepsilon$.

Perhaps we have two maps $f, g : \mathbb{R} \times \mathbb{R} \to_c \mathbb{R}$ and wish to confirm, for some compact/overt space $s : \mathcal{P}_{\Box\Diamond}(\mathbb{R})$ that there is some $x \in [0,1]$ such that for every $y \in s$, $f(x,y)$ is positive while $g(x,y)$ is negative. Then

$$\begin{aligned}
\mathsf{apart}_\varepsilon &: \mathcal{P}_{\Box\Diamond}(\mathbb{R}) \overset{nd}{\to}_c \mathbb{B} \\
\mathsf{apart}_\varepsilon(s) &\triangleq \exists x \in [0,1].\ \forall y \in s.\ [0 < f(x,y)]_\varepsilon \wedge [g(x,y) < 0]_\varepsilon.
\end{aligned}$$

approximately decides this, where $[0,1]$ is used as shorthand for $\mathsf{HeineBorel}(0,1)$. Like the previous query, this one has only false negatives (and its negation has only false positives). If we observe true, then it is certainly true, while if we observe false, then for every $x \in [0,1]$ there is a $y \in [0,1]$ such that either $f(x,y) < \varepsilon$ or $g(x,y) > -\varepsilon$.

5.4 Approximate root-finding

Recall the approximate root-finding program from the introduction,

$$\begin{aligned}
&\mathsf{roots}_f : * \overset{nd}{\to}_c \{* \mid \forall x \in K.\ f(x) \neq 0\} + \{x : K \mid |f(x)| < \varepsilon\} \\
&\mathsf{roots}_f \triangleq \mathsf{case}(\mathsf{tt}) \begin{cases} \iota[\exists x \in K.\ |f(x)| < \varepsilon]\,(y) & \Rightarrow \mathsf{inr}(\mathsf{simulate}(y)) \\ \iota[\forall x \in K.\ f(x) \neq 0]\,(n) & \Rightarrow \mathsf{inl}(n) \end{cases}.
\end{aligned}$$

Now that we have explained compact/overt spaces and binary covers, we are prepared to understand this code.

We begin by confirming that the two cases cover the entire input space. We can define a binary cover

$$\begin{aligned}
&[\cdot \neq 0]_\varepsilon : \mathbb{R} \overset{nd}{\to}_c \mathbb{B} \\
&[x \neq 0]_\varepsilon \triangleq \mathsf{case}(x) \begin{cases} \iota[\cdot \neq 0]\,(_) & \Rightarrow \mathsf{true} \\ \iota[\lambda x.\ |x| < \varepsilon]\,(_) & \Rightarrow \mathsf{false} \end{cases}
\end{aligned}$$

that approximately determines whether a real number is nonzero.

Then the binary cover

$$\begin{aligned}
&\mathsf{no_roots}'_\varepsilon : * \overset{nd}{\to}_c \mathbb{B} \\
&\mathsf{no_roots}'_\varepsilon \triangleq \forall x \in K.\ [x \neq 0]_\varepsilon
\end{aligned}$$

is defined by the same pair of opens as in the definition of roots_f, and hence those opens cover $*$.

It remains to define $\mathsf{simulate}$, which in this instance is a map

$$\mathsf{simulate} : \{* \mid \exists x \in K.\ |f(x)| < \varepsilon\} \overset{nd}{\to}_c \{x : K \mid |f(x)| < \varepsilon\},$$

but in general, for any overt space A and open $U : \mathcal{O}(A)$, defines a map

$$\mathsf{simulate} : \{* \mid \exists x \in A.\ U(x)\} \overset{nd}{\to}_c \{x : A \mid U(x)\}$$

that, given the existence of some values that satisfy a property U of A, can nondeterministically simulate those values. It is defined by the inverse image map

$$\begin{aligned}
\mathsf{simulate}^* &: \mathcal{O}(\{A \mid U\}) \to \mathcal{O}(\{* \mid \exists_A U\}) \\
\mathsf{simulate}^*(V) &\triangleq \exists_A(V \wedge U).
\end{aligned}$$

Proposition 5.9. *The inverse image map* $\mathsf{simulate}^*$ *preserves joins and* $\top$.

Proof. First, we confirm that the output lies in the open subspace U of A, that is, that $\mathsf{simulate}^*$ is gives equivalent results on inputs V and W satisfying $V \wedge U = W \wedge U$. This is straightforward because $\mathsf{simulate}^*$ immediately applies $\cdot \wedge U$ to its argument.

The map preserves joins since it is the composition $\exists_A \circ (\cdot \wedge U)$, both of which preserve joins. It preserves $\top$ (or equivalently, U), since

$$\mathsf{simulate}^*(\top) = \exists_A(U),$$

which is equal to $\top$ in $\{* \mid \exists_A U\}$. □

The approximate root-finding program roots_f accomplishes the relatively complicated task of approximate root-finding over a very general class of functions with a very short definition that works by composing some constructs from this chapter and the previous one. Most of the real computational work is accomplished by the formal proof that K is compact/overt. This gives evidence that the property of being compact/overt corresponds to a very general and composable computational interface for exhaustive search.[4]

[4]Escardó [10] discusses the computational significance of compactness.

Chapter 6

Related work

6.1 Alternative theories of constructive topology

Synthetic topology gives an alternative computable interpretation of topology that differs subtly from formal topology and locale theory [9, 20]. Synthetic topology gives an alternative intuition about overlapping patterns. In this understanding, data types from a (functional) programming language serve as spaces, and open sets are semidecidable predicates that take values of some type as arguments; for points in the open set, the semidecider must halt, whereas for points in the closed complement, the semidecider should not halt.

In this case, an open cover $U \leq \bigvee_{i:I} V_i$ means that whenever U halts, then some semidecider V_i also halts.

This interpretation is limited to (classically) spatial locales (locales that also make sense as conventional topological spaces) and covers indexed by recursively enumerable sets. But it gives an alternative computational interpretation of overlapping pattern matching: to run a pattern match, run the semideciders for each case in the pattern match *concurrently*. Since the cases must cover the entire space, the semidecider for one case will eventually halt and at that point proceed into the branch corresponding to that case. However, implementing overlapping patterns naively with this computational interpretation does not seem particularly efficient.

Taylor's *abstract stone duality* [36] is a theory/language for locally compact topological spaces. The additional dependency of local compactness means that all spaces are exponentiable, which is convenient as the formulation makes use of such exponentials.

6.2 Related programming formalisms involving continuity, partiality, or nondeterminism

Marcial-Romero and Escardó [22] define a language with real number computation which has a foundational family of functions $\mathrm{rtest}_{a,b}$ which map real numbers to nondeterministic Boolean values. This language also admits nontermination, so it

denotes its types as Hoare powerlocales (which corresponds to $\mathcal{P}_\Diamond$ in this thesis). Establishing totality requires reasoning within their operational model. Our work differs in a few ways: we describe a general principle for constructing nondeterministic maps, and we are able to use the positive Hoare powerlocale, which guarantees totality (at the cost of requiring formal proofs of covering).

Escardó's defines a language "Real PCF" with a denotational semantics in terms of cpo's, in which there is an operation known as a "parallel conditional," which corresponds to the internal "or" operation on the Sierpínski space $\vee_\Sigma : \Sigma \times \Sigma \to_c \Sigma$ in our formalism. Parallel conditionals applied to construct total deterministic functions on the real numbers, which differs from most of our examples, whose computations are total but nondeterministic.

Similarly, Tsuiki's work on computation with Gray-code based real numbers [38] is based on "indeterministic" computation, where potentially nonterminating computations must be interleaved, and those that terminate must agree in their answers.

6.3 Overlapping pattern matching

We are unaware of any other notion of pattern matching that permits patterns where determining membership is undecidable (without jeopardizing totality).

Coquand [6] gives a topologically motivated explanation of pattern matching for dependently typed functional programming, describing patterns as (disjoint) partitions of a space.

Müller [23] describes a system for exact real arithmetic that has a datatype of "lazy Booleans" analogous to our $\mathbb{B}_\perp$, as well as an n-ary `choose` operation on lazy Booleans that would be analogous in our language to

$$\mathsf{choose}_n : \prod_{i:\mathsf{Fin}(n)} \mathbb{B}_\perp \overset{nd,p}{\to}_c \mathsf{Fin}(n)$$

$$\mathsf{choose}_n(b) \triangleq \mathsf{case}(b)\left\{[i : \mathsf{Fin}(n)] \quad \iota[[i \mapsto \cdot = \mathsf{true}]]\,(\_) \quad \Rightarrow i\ .\right.$$

Dijkstra [8] introduces "guarded commands," a language construct for imperative programming languages, where a branch can be chosen nondeterministically from a list of statements each guarded by a Boolean expression.

6.4 Binary covers

Some of the modified Boolean spaces give topological interpretations of well-known fuzzy logics: $\mathbb{B}_\perp$ corresponds to Kleene's three-valued logic [17], $\mathcal{P}^+_\Diamond(\mathbb{B})$ to Priest's logic of paradox [29], and $\mathcal{P}_\Diamond(\mathbb{B})$ to Dunn/Belnap's four-valued logic [13].

dReal is a tool that allows computation of approximate truth values over the real numbers [12], allowing order comparisons and bounded quantifiers. The calculus of binary covers presented here, when restricted to $\mathbb{R}$, provides similar computational

abilities but with a different foundational framework. In a sense, it shows how it is possible to generalize the theory behind dReal to spaces other than $\mathbb{R}$.

6.5 Implementation

Using the theory of abstract stone duality, Bauer and Taylor have developed a computer implementation, *Marshall*, that supports exact real arithmetic computation based on the Dedekind real numbers [3].

Some aspects of of formal topology have been formalized within the Matita theorem prover (another theorem prover implementing Martin-Löf type theory) [1, 5]. Their library is not as concrete as ours; as far as we are aware, they do not construct any spaces (in an empty context), nor define any continuous maps other than identity maps and compositions of continuous maps, focusing instead on the relation between various notions of spaces and maps.

Chapter 7

Discussion and conclusion

The programs described in this thesis conceivably support reasonably efficient implementations, thanks to their use of formal topology and (predicative) locale theory. Formal proofs that inverse image maps preserve meets and joins not only confirm that they indeed define continuous maps but also provide computational content.

Overlapping pattern matches seem useful for a variety of purposes but in particular highlight the importance of nondeterminism in constructing programs that manipulate spaces. The approximate root-finding procedure in the introduction and the case study of binary covers demonstrate useful nondeterministic programs that can be constructed with overlapping patterns.

7.1 Coq implementation

We developed a library for formal topology in the predicative fragment of Coq (i.e., without use of the `Prop` universe) [32]. The library includes the following constructions:

- the one-point space $*$ and the unique maps $A \to_c *$
- discrete spaces and a construction that creates a continuous map $f : A \to_c B$ from a discrete space A to any space B, given for each $a : A$ a point $f_a : * \to_c B$
- arbitrary type-indexed sum spaces, their injections, and their universal property
- arbitrary type-indexed product spaces, their projections, and their universal property
- metric "completion" spaces of metric sets, and a construction for lifting Lipschitz continuous functions from metric sets to metric completions
- lifted spaces (e.g., $A_\perp$), with implementations of $\perp : * \to_c A_\perp$ and $\mathsf{up} : A \to_c A_\perp$ as continuous maps
- definitions regarding subspaces, and some particular facts about open and closed subspaces

- definition of a predicative analogue of locales, and facts relating them to formal spaces
- a simplified overlapping pattern matching construction, where patterns must correspond to opens of the space of the scrutinee

7.2 Programming with a theorem prover

A continuous map $f : A \to_c B$ is defined by an inverse image map $f^* : \mathcal{O}(B) \to \mathcal{O}(A)$ that preserves joins, $\top$, and binary meets. We can view f^* as a complete *specification* of the behavior of the behavior of f, and it is the formal (and constructive) proofs that f^* preserves the relevant structure that serve as the *implementation* of the computational behavior. It is not enough only to know informally that f^* preserves the relevant structure. At the same time, different formal proofs still give equivalent continuous maps, so (disregarding matters such as computational efficiency) it does not matter which formal proof is given.

We believe that theorem provers such as Coq are well-suited for programming in this manner. One can define in Gallina the inverse image map f^*, and then use the theorem-proving interface to prove that f^* preserves the relevant structure. For instance, in our Coq development, we made heavy use of "proof-relevant" typeclass-based rewriting[1]. Without tools such as this, defining the formal proofs may be potentially tedious. Since the formal proofs are irrelevant to reasoning about programs (beyond the fact that they exist), their definitions can be left opaque (until it is time to run/extract programs).

It is possible to avoid the need for proofs by restricting to the language defined in Section 2.4, which basically only allows composition of existing maps. There are also no proof obligations in the pattern matching construction for partial and non-deterministic maps, but there are proof obligations for constructing either total or deterministic maps. The proof obligations needed for pattern families is always to show that they belong to a particular Grothendieck pretopology. Given the compositional structure of Grothendieck pretopologies, it should be convenient to use a theorem prover to derive pattern families. This overlapping case splitting would be similar to using the `destruct` tactic in Coq to split a term in an inductive type into possible cases. Viewing pattern families in this way, the obligation that they belong to a particular Grothendieck pretopology is more like a part of the program structure, rather than a side condition that must be verified independently of the program syntax.

7.3 Future work

We are broadly interested in understanding how constructive topology can inform the design of programming languages for continuous systems.

[1] That is, rewriting of terms in `Type` rather that `Prop`. This feature was added to Coq in version 8.5.

Dependent types

For verification of systems, it is desirable to have a full dependently typed language for expressing programming tasks. For instance, reconsider the problem of an autonomous car that must decide whether or not to stop at a traffic light that has just turned yellow. We can model the problem as such: we have spaces $\mathsf{state\_before}$, $\mathsf{state\_after}$ and $\mathsf{decision}$ that describe the possible states of the car when the light turns yellow and when it turns red, respectively, and the decisions that can be made when the light turns yellow; a dynamics function $\mathsf{dynamics} : \mathsf{state\_before} \times \mathsf{decision} \to_c \mathsf{state\_after}$; and an open $\mathsf{safe} : \mathcal{O}(\mathsf{state\_after})$ that indicates if a given state is outside of the intersection, and thus safe. We want to ensure that for every possible car state $s : \mathsf{state\_before}$ there is a decision $d : \mathsf{decision}$ such that $\mathsf{safe}(\mathsf{dynamics}(s, d))$, but the decision must be allowed to be made nondeterministically. With dependent types, we could imagine encoding this task as

$$\prod_{s:\mathsf{state\_before}} \mathcal{P}^+_\Diamond \left(\sum_{d:\mathsf{decision}} \mathsf{safe}(\mathsf{dynamics}(s, d)) \right).$$

It ought to be possible to accomplish this by taking the gros topos of sheaves over the site generated by the open cover topology [11]. We do not yet understand how to extend the partiality and nondeterminism monads to the entire topos, but believe that approaches from synthetic topology might apply.

Probabilistic programming

Constructive topology provides a good setting for phrasing probability theory in a constructive manner, and so it should inform the theory and semantics of probabilistic programming. From any formal space A one can construct a space $\mathcal{R}(A)$ of probability distributions on A [44, 47].

Simpson [33] describes "random locales," which are formal spaces which are not spatial, meaning that they have no analogue in classical topology. These random locales characterize the properties that occur with probability 1. For instance, Simpson describes the sublocale Ran of $\prod_{\_:\mathbb{N}} \mathbb{B}$ of random sequences of Boolean values. On this sublocale, the pattern match

$$\begin{aligned} &\mathsf{geometric} : \mathsf{Ran} \to_c \mathbb{N} \\ &\mathsf{geometric}(x) \triangleq \mathsf{case}(x) \left\{ [n : \mathbb{N}] \quad \iota[[n \mapsto \cdot = \mathsf{true}] \wedge \textstyle\bigwedge_{k<n} [k \mapsto \cdot = \mathsf{false}]] \, (\_) \quad \Rightarrow n \right. \end{aligned}$$

which finds the first occurrence of a true in the sequence defines a valid continuous map (it would not if the codomain were $\prod_{\_:\mathbb{N}} \mathbb{B}$). Although Ran has no points $* \to_c \mathsf{Ran}$, so this $\mathsf{geometric}$ cannot be run on any deterministic inputs, Ran *does* have nondeterministic points $* \xrightarrow{nd}_c \mathsf{Ran}$, so $\mathsf{geometric}$ can be run on nondeterministic values.

Bibliography

[1] Andrea Asperti, Maria Emilia Maietti, Claudio Sacerdoti Coen, Giovanni Sambin, and Silvio Valentini. Formalization of formal topology by means of the interactive theorem prover Matita. In *International Conference on Intelligent Computer Mathematics*, pages 278–280. Springer, 2011.

[2] Andrej Bauer. Realizability as the connection between computable and constructive mathematics. In *Lecture notes for a tutorial at a satellite seminar of CCA 2004*, 2005.

[3] Andrej Bauer. Efficient computation with Dedekind reals. In *Fifth International Conference on Computability and Complexity in Analysis, pages i–vi, Hagen, Germany*, 2008.

[4] Francesco Ciraulo, Maria Emilia Maietti, and Giovanni Sambin. Convergence in formal topology: a unifying notion. *Journal of Logic and Analysis*, 5, 2013.

[5] Claudio Sacerdoti Coen and Enrico Tassi. Formalising overlap algebras in Matita. *Mathematical Structures in Computer Science*, 21(04):763–793, 2011.

[6] Thierry Coquand. Pattern matching with dependent types. In *Proceedings of the Workshop on Types for Proofs and Program*, pages 66–79, 1992.

[7] Thierry Coquand, Giovanni Sambin, Jan Smith, and Silvio Valentini. Inductively generated formal topologies. *Annals of Pure and Applied Logic*, 124(1):71–106, 2003.

[8] Edsger W. Dijkstra. Nondeterminacy and formal derivation of programs. *Comm. ACM*, 18(8):453–457, 1975.

[9] Martín Escardó. Synthetic topology: of data types and classical spaces. *Electronic Notes in Theoretical Computer Science*, 87:21–156, 2004.

[10] Martín Escardó. Infinite sets that admit fast exhaustive search. In *22nd Annual IEEE Symposium on Logic in Computer Science (LICS 2007)*, pages 443–452. IEEE, 2007.

[11] Michael P Fourman. Continuous truth I: Non-constructive objects. *Studies in Logic and the Foundations of Mathematics*, 112:161–180, 1984.

[12] Sicun Gao, Jeremy Avigad, and Edmund M Clarke. Delta-decidability over the reals. In *Logic in Computer Science (LICS), 2012 27th Annual IEEE Symposium on*, pages 305–314. IEEE, 2012.

[13] Siegfried Gottwald. Many-valued logic. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, spring 2015 edition, 2015.

[14] Peter T Johnstone. *Topos theory*. Academic Press, 1977.

[15] Peter T Johnstone. *Sketches of an elephant: A topos theory compendium*, volume 2. Oxford University Press, 2002.

[16] Tatsuji Kawai. Localic completion of uniform spaces, 2017.

[17] Stephen Cole Kleene. *Introduction to metamathematics*. North-Holland, 1952.

[18] Leslie Lamport. Buridan's principle. *Foundations of Physics*, 42/8:1056–1066, October 1984.

[19] S.M. Lane and I. Moerdijk. *Sheaves in Geometry and Logic: A First Introduction to Topos Theory*. Hochschultext / Universitext. Springer-Verlag, 1992.

[20] Davorin Lešnik. Synthetic topology and constructive metric spaces. *Diss. University of Ljubljana*, 2010.

[21] Maria Emilia Maietti and Giovanni Sambin. Why topology in the minimalist foundation must be pointfree. *Logic and Logical Philosophy*, 22(2):167–199, 2013.

[22] J Raymundo Marcial-Romero and Martín H Escardó. Semantics of a sequential language for exact real-number computation. *Theoretical Computer Science*, 379(1-2):120–141, 2007.

[23] Norbert Th Müller. Enhancing imperative exact real arithmetic with functions and logic. In *Computability and Complexity in Analysis, Ljubljana*, 2009.

[24] Russell O'Connor. Certified exact transcendental real number computation in Coq. In *Theorem Proving in Higher Order Logics*, pages 246–261. Springer, 2008.

[25] Erik Palmgren. Predicativity problems in point-free topology. In *Proceedings of the Annual European Summer Meeting of the Association for Symbolic Logic, held in Helsinki, Finland*, pages 221–231, 2003.

[26] Erik Palmgren. A constructive and functorial embedding of locally compact metric spaces into locales. *Topology and its Applications*, 154(9):1854–1880, 2007.

[27] Pavel Panchekha, Alex Sanchez-Stern, James R Wilcox, and Zachary Tatlock. Automatically improving accuracy for floating point expressions. *ACM SIGPLAN Notices*, 50(6):1–11, 2015.

[28] Maria Cristina Pedicchio and Walter Tholen. *Categorical foundations: special topics in order, topology, algebra, and sheaf theory*, volume 97. Cambridge University Press, 2004.

[29] Graham Priest. The logic of paradox. *Journal of Philosophical logic*, 8(1):219–241, 1979.

[30] Giovanni Sambin and Silvio Valentini. Building up a toolbox for Martin-Löf intuitionistic type theory: subset theory. In Giovanni Sambin and Jan Smith, editors, *Twenty-five years of Constructive Type Theory*, pages 221–244. Oxford Logic Guides, 1998.

[31] Robert Sedgewick and Kevin Wayne. Introduction to programming in Java: An interdsiciplinary approach.

[32] Benjamin Sherman, Luke Sciarappa, and Clément Pit-Claudel. Coq library for formal topology, May 2017. Available at `https://doi.org/10.5281/zenodo.581231`.

[33] Alex Simpson. Measure, randomness and sublocales. *Annals of Pure and Applied Logic*, 163(11):1642–1659, 2012.

[34] Alex K Simpson. Lazy functional algorithms for exact real functionals. In *International Symposium on Mathematical Foundations of Computer Science*, pages 456–464. Springer, 1998.

[35] Bas Spitters. Locatedness and overt sublocales. *Annals of Pure and Applied Logic*, 162(1):36–54, 2010.

[36] Paul Taylor. A lambda calculus for real analysis. *Journal of Logic and Analysis*, 2, 2010.

[37] Christopher F Townsend. On the parallel between the suplattice and preframe approaches to locale theory. *Annals of Pure and Applied Logic*, 137(1-3):391–412, 2006.

[38] Hideki Tsuiki. Real number computation through gray code embedding. *Theor. Comput. Sci.*, 284(2):467–485, July 2002.

[39] Steven Vickers. *Topology via logic*. Cambridge University Press, 1989.

[40] Steven Vickers. The double powerlocale and exponentiation: a case study in geometric logic. *Theory and Applications of Categories*, 12(13):372–422, 2004.

[41] Steven Vickers. Localic completion of generalized metric spaces I. *Theory and Applications of Categories*, 14(15):328–356, 2005.

[42] Steven Vickers. Some constructive roads to Tychonoff. *From Sets and Types to Topology and Analysis: Towards Practicable Foundations for Constructive Mathematics*, 48:223, 2005.

[43] Steven Vickers. Sublocales in formal topology. *The Journal of Symbolic Logic*, 72(02):463–482, 2007.

[44] Steven Vickers. A localic theory of lower and upper integrals. *Mathematical Logic Quarterly*, 54(1):109–123, 2008.

[45] Steven Vickers. The connected Vietoris powerlocale. *Topology and its Applications*, 156(11):1886–1910, 2009.

[46] Steven Vickers. Localic completion of generalized metric spaces II: Powerlocales. *Journal of Logic and Analysis*, 1, 2009.

[47] Steven Vickers. A monad of valuation locales. *Preprint at* `http://www.cs.bham.ac.uk/~sjv/Riesz.pdf`, 2011.

[48] Frank Waaldijk. On the foundations of constructive mathematics–especially in relation to the theory of continuous functions. *Foundations of Science*, 10(3):249–324, 2005.

[49] K. Weihrauch. *A simple introduction to computable analysis*. Informatik-Berichte. Fernuniv., Fachbereich Informatik, 1995.

List of symbols

$\cdot_\perp$ The lifting monad on spaces. 44, 53, 54, 59, 60, 67, 76

$A \overset{nd,p}{\to}_c B$ The set of nondeterministic and partial maps from a space A to a space B. 47, 50, 53, 61, 62, 70, 92

$A \cong B$ Objects A and B in a category are isomorphic (usually, either two sets are isomorphic or two spaces are homeomorphic). 23, 25, 26, 29, 30, 34, 35, 51, 62, 67, 73, 78, 85, 104

$A \overset{nd}{\to}_c B$ The set of nondeterministic maps from a space A to a space B. 44–46, 57, 59, 63, 73, 80–88, 97, 103

$A \overset{p}{\to}_c B$ The set of partial maps from a space A to a space B. 39–41, 43, 44, 53, 59, 60, 64, 70, 72, 75, 76

$A \to_c B$ The set of continuous maps from a space A to a space B. 11, 12, 15, 23, 25, 26, 28–32, 34, 35, 37, 38, 40, 41, 43, 45–53, 55, 58–63, 69–71, 73, 74, 76–78, 84–87, 92, 95–97

$a \downarrow b$ The meet (intersection) of two basic opens a and b of a space A (which is an open of A). 21–24, 33, 34, 38, 53

$a \equiv b$ Intensional equality of a and b (which must have the same type). 16, 21, 29–32, 42, 64–66

$a \triangleleft U$ The basic open a of a space A is covered by the open U of A. 20–23, 28, 30, 32–34, 38, 42, 45, 53

$f \leqq g$ The truth order implication f implies g for binary covers $f, g : \Gamma \overset{nd}{\to}_c \mathbb{B}$. 83, 84

$f \otimes g$ The parallel composition $f \otimes g : A \times X \to_c B \times Y$ of $f : A \to_c B$ and $g : X \to_c Y$. 49, 52, 62, 63, 67, 74, 75

f^* The inverse image map $f^* : \mathcal{O}(B) \to \mathcal{O}(A)$ of a map of spaces $f : A \to_? B$. 25

$f_!$ The direct image map $f_! : \mathcal{O}(A) \to \mathcal{O}(B)$ of an open map $f : A \to_o B$. 48

$p \Rightarrow e$ A branch in a pattern match, where p is a pattern such that, if matched, the expression e results. 57–61, 68, 70, 74–78, 85, 87, 88, 92, 97