Electronic Notes in Theoretical Computer Science 87 (2004)
http://www.elsevier.nl/locate/entcs/volume87.html 150 pages

Synthetic topology
of data types and classical spaces

Martín Escardó

*School of Computer Science, University of Birmingham,
Birmingham B15 2TT, England*

Abstract. Synthetic topology as conceived in this monograph has three fundamental aspects:

1. to explain what has been done in classical topology in conceptual terms,
2. to provide one-line, enlightening proofs of the theorems that constitute the core of the theory, and
3. to smoothly export topological concepts and theorems to unintended situations, keeping the synthetic proofs unmodified.

The unintended situation that we focus on is the theory of computation, in particular regarding programming languages from both operational (Part I) and denotational (Part III) points of view, with emphasis on sequential computation. We are aware of other applications of synthetic topology, e.g. to locales, convergence spaces, sequential spaces, equilogical spaces, and some sheaf and realizability toposes, but this will be reported elsewhere.

Aspects 1 and 2 are the subject of Part II. However, it turns out that it is possible to tackle aspect 3 without previous reference to 1 or 2. In fact, we start by developing synthetic topology of programming-language data types in Part I, without assuming any background in classical topology and without introducing any. Part III combines ideas from Parts I and II, developing non-trivial computational applications. The main new result is a computational version of the Tychonoff theorem. We also review previously known applications and explain how topology and semantics interact in program-correctness proofs.

Although computers are finite, infinity shows up in a number of important situations in the theory of computation, e.g. *infinity in syntax*: loops, recursion; *infinity in time:* non-terminating computations; *infinity of data:* stream computation and higher-type computation; *infinity in precision:* real-number computation; *infinity through abstraction:* probabilistic descriptions.

The first few chapters of Part I explore how the fundamental topological notions of continuous map, open set, closed set, compact set, Hausdorff space, and discrete space reconcile the finite character of computers with the infinite nature of the entities one wishes to calculate with. One of the main contributions of this monograph is to explain the computational nature of the the notion of compactness. Roughly speaking, a set is compact if and only if, given any semidecidable property, one can semidecide whether it holds for *all* elements of the set in finite time. Surprisingly, there are infinite computationally compact sets, for example that of infinite streams of binary digits.

Keywords. Synthetic topology, data type, topological space, Tychonoff theorem, domain theory, computability, recursion theory, λ-calculus, functional programming, programming-language semantics, operational semantics, denotational semantics, Scott model of PCF, equilogical space, cartesian closed extension.

MSC 2000. 54-02, 03B40, 06B35, 68Q10, 68Q55, 68P99, 03D65, 03D75, 03B15, 03B70 , 18B30.

Contents

Chapter 0

Preface

This is a revised and expanded edition of manuscript lecture notes originally written for an advanced course at the Bellairs Research Institute of McGill University based in Barbados, in April 2003. Prakash Panangaden is warmly thanked for inviting me to deliver this course and to subsequently submit the resulting lecture notes for publication. This has forced me to invest time in shaping the presentation of the ideas and finally write them down.

Reinhold Heckmann, Alex Simpson and Paul Taylor kindly proof-read earlier versions of this manuscript and provided useful suggestions. However, I haven't followed all the advice given by them, and the errors and imprecisions that remain are mine. Paul Taylor is also gratefully acknowledged for recent discussions about his abstract Stone duality and its connections with the material presented here (see the entry *Taylor* in the index). I have had many interesting and profitable conversations with Achim Jung and Steve Vickers. Finally, I am grateful for the comments and suggestions given by the anonymous referee and by the overt students of the *Midlands Graduate School for the Foundations of Computing Science* and *Appsem Spring School* held jointly in March-April 2004 at Nottingham University, in particular my students Thomas Anberree, José Raymundo Marcial-Romero, and Ho Weng Kin. There are many more people to acknowledge, and I apologize for stopping here.

0.1 Organization

This set of notes consists of thirteen chapters divided in three parts:

I Topology of data types.

II Topology of classical spaces.

III Domain theory, topology and denotational semantics.

The computational Part I can be read independently of the mathematical Part II. Computer scientists may use Part I as a bridge to reach Part II, and mathematicians may travel in the opposite direction. The central Chapter 3 of Part I parallels the central Chapter 9 of Part II. Part III unifies the parallel computational and mathematical developments of Parts I and II, and concludes with some applications.

Each part starts with a discussion of its own contents and organization. The particular chosen linear sorting of the chapters is to some extent idiosyncratic, and readers are invited to try their own paths, not necessarily linear, probably including cycles.

0.2 Intended audience

Three audiences are expected: researchers who are familiar with the area (and hence know well the mathematics and the computer science involved), mathematicians who are not necessarily familiar with the required computer-science concepts but would like to learn about the applications of topology to the theory of computation, and computer scientists who are not familiar with the applications of topology to computer science.

0.3 Prerequisites and supporting material

The ideal prerequisites are topology, domain theory, recursion theory, and programming language semantics, but it should be possible to cover Chapters 1–5 without them. Many expository texts have been written in the (computer-science and mathematical) literature about such topics. A graduate student may take this set of notes as a guide to such texts, or the other way round. To begin with, a biased selection of (not necessarily expository) supporting texts is the following:

1. Scott's seminal manuscript on a logic of computable functions [113], and papers *continuous lattices* [111] and *data types as lattices* [112].

2. Plotkin's seminal paper on PCF [101] and widely circulated *Pisa notes* [103].

3. Smyth's *topological view of predicate transformers* [121] and handbook chapter on *topology* [122].

4. Abramsky's *logic of observable properties* [1].

5. Vickers' *topology via logic* [140].

6. Abramsky and Jung's handbook chapter on *domains* [3].

7. Amadio and Curien's book on *domain theory and lambda-calculi* [4].

8. Thomas Streicher's course notes on *mathematical foundations of functional programming* [128].

9. Weihrauch's book on *computable analysis* [142].

Possible supporting texts for the above supporting texts include:

1. Rogers' book on *recursion theory* [108].

2. Davey and Priestley's *introduction to lattices and order* [24].

3. The multi-author *compendium of continuous lattices* [54] or, preferably, its recent expanded edition *continuous lattices and domains* [55].

4. Johnstone's book on *Stone spaces* and their descendants [69].

5. Any good text on general topology such as Kelley's [78], Dugundji's [31], Bourbaki's [17]. A reader with no background on the subject may prefer to start with texts such as [117] or [129].

6. For expository reasons, we have avoided the use of category theory. But, inevitably, occasional references are made to it. Of course, check Mac Lane's book [89]. A survey of its uses in functional programming and programming-language semantics is [105].

0.4 Topology of data types

The topological view of computational phenomena has been developed in intuitionistic and constructive mathematics, logic and recursion theory, domain theory, and type-two theory of effectivity. This goes back to Brouwer, who proved that, in his intuitionistic approach to mathematics, all functions $f\colon \mathbb{R} \to \mathbb{R}$ are continuous. In these notes, our mathematics is classical, but his arguments can be exported to the theory of computation as developed in classical mathematics to conclude that computable functions are continuous.

Work by Kleene, Kreisel, Myhill/Shepherdson, Rice/Shapiro, Nerode, Scott, Ershov, Plotkin, Smyth, Abramsky, Vickers, Weihrauch and no doubt many others gradually exhibited the topological character of data types other than the real numbers, emphasizing the fact that computable functions are topologically continuous generalizes to any domain of computation.

0.5 Synthetic topology

We reformulate classical topological concepts as continuity notions with the aid of the *Sierpinski space*, which has one open point $\top$ (true) and one closed point $\perp$ (false) and plays the role of a space of results of *observations* or *semidecisions*. For example, a set is closed iff one can continuously semidecide its complement, a space is Hausdorff iff one can continuously tell distinct points apart, and a subset of a space is compact iff one can continuously universally quantify over it.

Replacing "continuously" by "computationally", computational versions of the topological notions are obtained. Surprisingly, there exist computationally compact sets of infinite cardinality, such as the Cantor space of infinite sequences of binary digits and the closed unit interval of real numbers. These two uncountable spaces behave as finite sets in that they admit universal quantification in finite time for continuous Sierpinski-valued properties defined on them.

Using the lambda-calculus, we can combine the continuous maps that define compactness, Hausdorff separation, closedness etc. to produce new continuous maps. For example, the theorem that a compact subspace Q of a Hausdorff space X is closed has the following computational reading: If we can computationally tell distinct points of X apart and we can computationally quantify over Q, then we can computationally semidecide the complement of Q. The synthetic proof of the topological theorem is a lambda-expression that defines the semidecision function for the complement of Q

from the apartness map of X and the quantifier of Q: A point x of X is not in Q iff it is distinct from all points of Q. Hence the characteristic function of the complement of Q is the lambda-expression $\chi_{X \setminus Q}(x) = (\forall q \ \in Q.x \neq q)$. Because functions that are lambda-constructible from continuous maps are themselves continuous, this is all we need to do. But lambda-definability also preserves computability. Thus, both the formulation of the theorem and its proof are seen to simultaneously have computational and topological content, and synthetic proofs are programs in a literal sense.

(At this point, expert readers will object that the category of continuous maps of topological spaces fails to be cartesian closed and hence doesn't admit an interpretation of the (simply typed) λ-calculus. In order to overcome this obstacle, we formally add, in a standard way, imaginary spaces that implement the exponentials (function spaces) that are missing in the world of topological spaces. For expository reasons, however, we first prove the theorems in less generality than they are known, by requiring the needed exponentials to exist as real spaces and working with the restricted lambda-calculus. Thus, for example, at a first instance the theorem discussed above has the extraneous assumption that the exponential $\mathbb{S}^X$ exists as a real space, where $\mathbb{S}$ is the Sierpinski space. Then, at a second stage, the extraneous assumptions are removed with the aid of imaginary exponentials, but the original proofs are retained.)

The terminology *synthetic* goes back to its use in *synthetic differential geometry* [82] and *synthetic domain theory* [65]. In our case, the point is that the notion of continuity is taken as primitive and that the other topological notions, including that of open set, are derived from it via the use of a space of results observations. Moreover, proofs are obtained by manipulating continuous maps rather than points and open sets. In a computational setting, the intended connotation of the word is that the topology is operationally *extracted* from a programming language as opposed to *imposed* into it via a denotational semantics.

0.6 Synthetic topology of data types

In the synthetic approach to the topology of data types, which is based on the above ideas, we start from computational definitions of topological notions, and at a later stage convince ourselves that the computational notions match the classical topological ones. In fact, in order to stress this point, we develop the synthetic topology topology of data types without assuming any background on classical topology (and without introducing any).

Many of the definitions of classical topology arise as theorems in synthetic topology. Moreover, the topologies that arise are familiar in topology and analysis. For instance, computable functions on infinite sequences of binary digits are continuous with respect to the Cantor topology.

Many applications of the topology of data types are known in the theory of computation. To give a simple example, it follows from the compactness of the Cantor space that equality of integer-valued continuous functions on the Cantor space is computationally decidable. We prove computational compactness of the Cantor space by writing a functional program that implements the *Tychonoff theorem* in the countable case.

Part I

Topology of data types

Contents and organization

The centre of gravity of this part is Chapter 3, for which no previous knowledge of classical topology is required. Although this part is primarily addressed to computer scientists, it has been designed to be readable, at least to some extent, by mathematicians with no previous exposure to computer science, for whom Chapter 2 should serve as an introduction to the relevant prerequisites. However, mathematicians may prefer to start from Part II. In particular, Chapter 7 of that part justifies the synthetic formulations of the topological notions from the point of view of classical topology, and Chapter 9 parallels Section 3.12.

The introductory Chapter 1 briefly discusses an informal dictionary relating topological and computational notions, which is revised and expanded in Chapter 5 in light of the preceding technical development.

Chapter 2 introduces the notion of *data language* for a *programming language*, which allows us to discuss programs that manipulate not-necessarily-computable data without invoking a denotational semantics for the programming language. The data language can, in particular, be taken to be the same as the programming language, in which case all data are computable. The concrete meaning of the synthetic topological notions vary together with the underlying data and programming languages, but the basic theory holds for a variety of pairs of languages, including sequential languages.

Chapter 4 gives an important example in which the synthetic notions of open set and continuous map satisfy the classical topological axioms. Expert readers may object that this example crucially relies on the presence of certain parallel features in the data language, but in any case the development of Chapter 3 shows that, despite the fact that the synthetic notions may fail to satisfy the classical axioms, the classical theorems do hold for them.

Chapter 6 takes a closer look at the classical topologies of some prototypical domains of computation. For this, some basic knowledge of classical topology is required. Mathematicians may prefer to approach Part I starting from this (perhaps after reading the introductory Chapter 1). As in Chapter 4, some of the material relies on, and motivates, Part III.

Chapter 1

Smyth's dictionary

In this introductory chapter we briefly discuss an informal dictionary that relates computational and topological concepts [121, 122]:

Data type $\approx$ *topological space.*

Piece of data $\approx$ *point.*

We shall see many examples supporting this idea.

Semidecidable property $\approx$ *open set.*

(*observable property* – Abramsky [1],
affirmable property – Vickers [140].)

We shall give detailed accounts to this entry of the dictionary in Chapters 3 and 4. Here we briefly tell the traditional story.

Suppose that an observer is watching a black box that outputs decimal digits, one after the other, in a never-ending fashion. The observer can be any physical device, including a person. It may be that, for example, the black box is producing the decimal expansion of π. If the observer were able to see the internal machinery of the black box and maybe get hold of a program that controls its behaviour, then he would perhaps be able to prove or disprove that it will indeed produce the decimal expansion of π. However, because the box is black, the observer doesn't see its inside, and he has to content himself with what he can observe about its external behaviour.

For example, the property that the output is the decimal expansion of π is not observable, because the observer would have to wait until the end of time, or have a crystal ball, in order to be sure. But the negation of this property is observable: An algorithm for computing the decimal expansion of π is known. Hence the observer can run this algorithm, perhaps with the aid of a third physical device, and compare its output to that of the black box. If the black box is not producing the decimal expansion of π, the observer will realize that at some finite stage. This may take a billion zillion years if the black box is computing something very close to π, but the semidecision is possible in principle. If, on the other hand, the black box happens to be computing π, then the observer will be busy forever without ever having a chance of producing an answer.

Thus, by an observable property we mean a property such that, when it holds, one can make sure it does by observation, ignoring practical limits of time and other physical resources. If it doesn't hold, the observer is not obliged to answer; in fact, in this case, as in the example just given, the observer will usually be busy until the end of time trying to verify the property in vain.

Now suppose that $p(x)$ and $q(x)$ are observable properties. Then the conjunction $p(x) \wedge q(x)$ is certainly observable: First observe one of them, and then, if the experiment succeeds, observe the other, and, if the experiment succeeds again, then the fact that $p(x) \wedge q(x)$ holds has been observed.

Given a family $p_i(x)$ of properties, in order to observe that the disjunction $\bigvee_i p_i(x)$ holds it suffices to observe that one of the disjuncts $p_i(x)$ holds. Hence arbitrary disjunctions of observable properties are observable. As it stands, this claim is problematic. It is true that to observe the disjunction $\bigvee_i p_i(x)$ holds it suffices to observe that one disjunct $p_i(x)$ holds. However, how does the observer figure out which? Of course, he can try all in parallel. But how is the collection of all disjuncts $p_i(x)$ presented to him?

So, in summary, observable properties are closed under finite conjunctions and arbitrary disjunctions, where, for the moment, we are not sure what "arbitrary" means. This imprecision is clarified in two different ways in Chapters 3 and 4 with two different conclusions. This leads to a revision of the dictionary, which is summarized in Chapter 5.

Identifying a property with the set of elements that it satisfies, we arrive at the conclusion that observable sets are closed under the formation of finite intersections and arbitrary unions. That is, they satisfy the axioms for the open sets of a topology on the set of data.

Computable function $\approx$ *continuous map.*

As discussed in the preface, this goes back to Brouwer. This entry of the dictionary is the main topic of Chapters 4 and 6. Before that, in Chapter 3, it is made into a synthetic definition of continuity for functions between data types.

> *? ≈ compact set.*

> *"The notion of compactness is a little harder to motivate: but it will have the significance for us of a 'finitarily specifiable' set, or, alternatively, of a set of results attainable by a boundedly non-deterministic process." [121]*

One of the main purposes of these notes is to fill this gap. We also extend the dictionary with computational manifestations of other topological notions, such as those of Hausdorff space, discrete space, and function space.

1.1 Notes

We stress that the material of this chapter should be regarded as motivating the computational and mathematical development that follows rather than asserting any philosophical dogma. In particular, the discussion about observers and observable properties poses more questions than it answers, and is necessarily vague and prone to endless philosophical debate.

However, one can formulate objective, operational notions of observation with mathematical and computational precision in particular circumstances, as is done in Chapter 2 and studied in Chapters 3 and 4. In particular, the difference between observable and semidecidable properties is clarified in Chapters 2 and 4, which leads to a revision of the dictionary. The expansion and revision are summarized in Chapter 5.

Chapter 2

Operational notions of data

In order to be able to rigorously develop the topology of data types as suggested by the informal discussion of the previous chapter, we need precise definitions of notions of observation or semidecision. For this, we in turn need precise notions of data. We obtain suitable definitions by considering *data languages* for a base *programming language*. In an extreme, but familiar, case, the data language is taken to be the same as the programming language. But we can also consider the situation in which programs can manipulate data coming from the environment, which we are not entitled to assume to be necessarily programmable or computable. For the sake of uniformity, the elements of function types are regarded as data, following Strachey's slogan that functions are first-class citizens.

In an attempt to make the topological character of computational phenomena convincing, we reason on purely operational grounds without invoking any topologically motivated denotational semantics for the programming language under consideration (see Section 12.15 of Part III). Moreover, we don't assume that our language includes the so-called parallel-or operation, although we do discuss the topological consequences of extending the language with a weaker variant.

2.1 Computational set-up

For mathematical simplicity, we consider a *functional* programming language [16], but we intend the development to be reasonably self-contained, and hence we explain the language as we proceed. We use Haskell notation, so that the interested reader can try the programs, e.g. using the `hugs` interpreter [73]. For computer scientists, we emphasize that, in our examples, it is crucial to use a call-by-name language — with more effort, and sacrificing clarity, one could use a call-by-value functional language such as ML [99], or even a traditional imperative language.

2.2 Functional programming

In functional programming one defines computable functions by writing down equations that they satisfy. Mathematically, in order to know what function a set of equations defines, we need to solve the system of equations (typically by means of fixed-point techniques, as explained in Chapter 12.14). Computationally, the equations are interpreted as a rewrite system and taken as an algorithm or computer program. An example is the factorial function:

```
f :: Nat -> Nat
f(0) = 1
f(n+1) = (n+1) * f(n)
```

In this simple case, both the mathematical and computational meanings of the system of equations are clear. On operational grounds, any system of equations has a solution, including e.g.

```
g :: Nat -> Nat
g(n) = g(n)+1
```

In this case, the mathematical meaning is not immediately clear, but it is evident that the operational solution is a constantly divergent function. If we regard such implicit definitions of computable functions as the analogue of the differential equations in physics, we can say that in physics one has singularities and in computation one has non-termination.

2.3 The Baire data type

For several reasons, we are interested in programs that manipulate (infinite) sequences of natural numbers. Our programming language has a built-in data type for sequences, but it will be more convenient for our purposes to take the mathematical view, regarding sequences as functions defined on the natural numbers. For simplicity, we shall pretend that the built-in data type `Int` is that of natural numbers, and so we declare:

```
type Nat = Int
```

(A data type of natural numbers without negative integers can be easily defined in our language, but this would be a distraction from our main aims.)

The *Baire data type* is defined by

```
type Baire = Nat -> Nat
```

Intuitively, this is the type of sequences of natural numbers. However, it turns out that, on operational grounds, it also has some extraneous elements (see Section 2.4). The set of sequences of natural numbers will occur as a *subspace*, which we will call the *Baire space*, of the Baire data type (see Section 3.5).

As a warming-up exercise, we consider the function that interleaves two given sequences:

```
interl :: (Baire,Baire) -> Baire
interl(s,t) = \i -> if even(i) then s(i/2) else t((i-1)/2)
```

A bit oddly, double-colon is used to indicate types and the pairing notation to denote cartesian products. Here an expression of the form `\i -> e` denotes the function that maps `i` to `e` (readers who know the λ-calculus should regard the symbol "\" as a one-legged letter λ). The if-then-else construction is (pre)defined by the equations

```
if True  then x else y = x
if False then x else y = y
```

and has type

```
(Bool,a,a) -> a
```

for any type variable `a`, where the type of booleans is (pre)defined by the declaration

```
data Bool = True | False
```

A more contrived definition of the interleaving function is the following. Firstly, define the *head*, *tail* and *cons*(truction) functions by

```
hd :: Baire -> Nat
hd(s) = s(0)

tl :: Baire -> Baire
```

```
tl(s) = \i -> s(i+1)

cons :: (Nat,Baire) -> Baire
cons(n,s) = \i -> if i == 0 then n else s(i-1)
```

Here "`==`" denotes the boolean-valued equality relation (for natural numbers in this context). The head of a sequence is its first term, the tail function decapitates its argument, and the cons function attaches a new head to a given sequence, and hence the following equations, which are not part of the program we are writing, hold:

```
cons(hd(s),tl(s)) = s
hd(cons(n,s)) = n
tl(cons(n,s)) = s
```

With this notation, the readers can convince themselves that the interleaving function explicitly defined above satisfies the equation

```
interl(s,t) = cons(hd(s),cons(hd(t),interl(tl(s),tl(t))))
```

This can, in fact, be regarded as an implicit definition of the interleaving function (c.f. discussion above about solving equations), or as an alternative program for computing it. That is, if one defines

```
interl' :: (Baire,Baire) -> Baire
interl'(s,t) = cons(hd(s),cons(hd(t),interl'(tl(s),tl(t))))
```

then

```
interl' = interl
```

holds.

2.4 Divergence and points at infinity

Of course, the equality relation on the Baire data type is not computationally decidable. However, the apartness relation is computationally *semi*decidable:

```
apart_B :: (Baire,Baire) -> Bool
apart_B(s,t) = apart(0)
               where apart(i) = if s(i) /= t(i)
                                then True else apart(i+1)
```

Here "/=" is the negation of the boolean-valued equality relation. For example, we have that

```
apart_B(\i -> 0,\i -> if i == 2^100 then 1 else 0) = True
```

although it takes a very long time to get the answer, and it takes infinitely long to get the answer when we run the expression

```
apart_B(\i -> 0, \i -> 0)
```

To be accurate, the above program defines the apartness map of the Baire *space*, which occurs as a subspace of the Baire *data type* (see Sections 3.6 and 3.9 below).

Notice that we started by saying that, in functional programming, one implicitly defines functions by equations they satisfy. But, of course, not every equation satisfied by a function uniquely determines it. For example, the interleaving function satisfies the equation

```
f = f
```

but the program

```
divergent_function :: (Baire,Baire) -> Baire
divergent_function = divergent_function
```

certainly doesn't define the interleaving function: When we run it, we get a non-terminating computation.

One attitude towards this phenomenon is to consider that such *divergent* programs fail to compute any entity of the required type, and hence should be ruled out of consideration (one cannot rule them out of existence because the halting problem is not computationally solvable). Another is to regard such kind of computational behaviour as an entity similar to a point at infinity in projective geometry, and think of the non-terminating program as computing such a postulated, intangible entity, which is usually denoted by $\perp$ and called *bottom* (the reason for this terminology is discussed below). This is the point of view that is compatible with the operational semantics of our language. Thus, for any data type `a`, the program `bot` defined below denotes bottom:

```
bot :: a
bot = bot
```

With this convention, one can now write

```
apart_B(\i -> 0,\i -> 0) = bot
```

That bottom should be regarded as a legitimate entity is supported by the fact that, for example, the following equations hold:

```
if True  then x else bot = x
if False then bot else y = y
(\i -> 0)(bot) = 0
```

As we shall see, the topology of data types is intimately related to divergent computations.

2.5 The Sierpinski data type

The *Sierpinski data type* is that of results of *observations* or semidecisions:

```
data S = T
```

By the above discussion, `S` has precisely two elements, namely `T` (pronounced *top* or *true*) and `bot`. Guided by examples such as the apartness map discussed above, we think of `T` as "observable true" and of `bot` as "unobservable false". In fact, notice that the apartness map is better typed using the Sierpinski data type, because the answer (observable) `False` is not possible.

2.1 LEMMA (TURING 1936) *The function* `diverges`: `S` $\to$ `S` *defined by*

```
diverges(T)=bot
diverges(bot)=T
```

is not computable, and hence not definable in our language.

PROOF Otherwise we would be able to solve the halting problem. $\square$

2.6 Internal and external views of data

What are the elements of a data type? There are two operational answers to this question, depending on whether we consider the language as existing in isolation or within an external environment that can supply data which is not necessarily programmable in the language, but which programs in the language can manipulate. We call these the *internal* and *external* views. For many purposes, it doesn't matter which view one takes, and, in fact, the synthetic topology developed below applies to both. But there is one exception: It turns out that, as we shall see, the Cantor space defined below is compact in one view but not the other. The reader can safely skip the material on the external view until this example is reached or studied, but should certainly consult it before Chapter 4 is reached.

Internal view. In the internal view, we take the elements of a data type to be simply the (equivalence classes of) programs of that type. We write $x \in \mathtt{a}$ to indicate that x is an element of the data type a. In particular, if we write $f \in (\mathtt{a} \to \mathtt{b})$, we imply that the function f is programmable.

External view. Programs in our language can exchange sequences of natural numbers with the environment, by communicating terms in succession in a never-ending fashion. For example, one can write a program $\mathtt{f}\colon \mathtt{Baire} \to \mathtt{Baire}$ and run it *interactively*: The computer will alternate between reading some terms of the sequence from the input, performing internal calculations, and writing some terms of the sequence to the output. For instance, if

```
f(s) = \i -> if even(i) then s(i+1) else s(i-1)
```

then, under the interactive regime, the program will wait for two numbers from the input, print them in reverse order, again wait for two more numbers, again print them in reverse order, and so on.

The crucial point here is that the input sequence is not necessarily computable — unless someone proves that the universe in which we live is a big Turing machine.

Now suppose that a program $\mathtt{f}\colon \mathtt{Baire} \to \mathtt{Baire}$ is written as a composition of programs $\mathtt{Baire} \to \mathtt{a}$ and $\mathtt{a} \to \mathtt{Baire}$ for some suitable data type a. If the input from the first is non-computable, then so will its output be in general. Hence we would get into trouble if we took the elements of the data type a to be programs. Moreover, even if the input turns out to be computable, this doesn't help, because an algorithm that generates the input is not disclosed: We only get the input itself.

Data language for the external view. To solve the problem, we define a *data language*, which will accommodate both programmable and external data. We take this to be the extension of our programming language with a constant of type Baire for each sequence of natural numbers. Each such constant represents a particular potential input.

We take the elements of type `a` to be the (equivalence classes of) expressions in the data language. Thus, we have *external data* (expressions in which input constants occur) and *programmable data* (expressions in our programming language). We write $x \in$ `a` to indicate that x is an element of `a`. In particular, if we write $f \in$ (`a` $\to$ `b`), this time we don't imply that the function f is programmable in the language or computable.

Notice that external data cannot occur *within* programs of our language, which remains unchanged. The point is that programs can *manipulate* external data. For example, given a program `f`: `Baire` $\to$ `Baire` and external data $s \in$ `Baire` we get external data `f`$(s) \in$ `Baire`. Although `f`(s) is not necessarily computable, we can certainly evaluate it relatively to the given input s using the computation rules of the language, with the understanding that whenever one attempts to evaluate s, one is actually reading the input.

2.7 Operational equivalence

When we wrote

```
interl = interl'
```

we didn't say what we meant, relying on the reader's intuition.

Two programs, or two pieces of data, of the same type are operationally equivalent if any observer will detect the same properties for them. To make this notion mathematically precise, we define a notion of experiment that an observer can perform.

By an *experiment* of type `a` we mean a function $u \in$ (`a` $\to$ `S`). To observe a given $x \in$ `a`, the observer prepares an experiment $u \in$ (`a` $\to$ `S`) and then runs $u(x)$ and waits for the evaluation to terminate (necessarily with result `T`). If it does, the observation succeeds. Thus, $x, y \in$ `a` are said to be operationally *equivalent* if convergence of $u(x)$ is equivalent to that of $u(y)$ for every experiment $u \in$ (`a` $\to$ `S`). We treat equivalent programs as if they were equal, both conceptually and notationally.

With an official definition of the elements of data types and equality for programs, one can now be more rigorous. The equation

```
cons(hd(s),tl(s)) = s
```

discussed above actually fails. The offending cases are those of the form

```
s = \i -> n
```

with `n` $\neq$ `bot`. In order to see this, for any `s` $\in$ `Baire`, define

```
s' = \i -> if i == 0 then s(0) else s(i)
```

Then

```
cons(hd(s),tl(s)) = s'
```

Because `s'(bot)=bot` and `s'(i)` $=$ `s(i)` for `i` $\neq$ `bot`, the original equation for `cons` holds if and only if `s` $=$ `s'` if and only if `s(bot)` $=$ `bot`.

2.8 Notes

In many examples of interest, it is not so easy to establish or refute equivalence of two given programs. One of the main aims of *programming language semantics* is to develop general techniques for that purpose. See e.g. Gordon [56], Gunter [57], Pitts [100], Plotkin [101, 103], Tennent [135] and Winskel [145]. In the next chapter we rely on operational methods, and in Chapter 13 of Part III, on denotational methods, which are briefly developed in Chapter 12 of the same part.

The notion of computability relative to external inputs is standard in recursion theory, where external inputs are known as *oracles*, which are used to study Turing degrees. They were previously known to Brouwer, in his intuitionistic approach to mathematics, as *lawless sequences*. Our view of external data is closer to Brouwer's than to Turing's, and coincides with that of [142].

Chapter 3

Synthetic topology of data types

In topology one finds notions such as space, continuous map, open set, closed set, discrete space, Hausdorff space, compact space and so on. Partly based on Smyth's dictionary, in this chapter we define computational notions with the same names, and later on convince ourselves that they match the original topological ones, where the match is precise under some natural assumptions on the model of observation (Theorem 4.1). For this chapter, no background in topology is required.

Using computational technology, we prove known theorems such as "every compact subspace of a Hausdorff space is closed". The proofs are programs in the literal sense. Thus, the topological character of data types is explicitly exhibited. In Part II, we turn this programme on its head: We apply the lambda-calculus to cheaply develop the core of classical topology.

In this chapter we work with the programming and data languages discussed in the previous. With the exception of Section 3.11, it doesn't make any difference whether we work with the internal or the external view of data.

3.1 Continuous maps of data types

In the traditional approach to the topology of data types of languages such as the one we are considering, one starts with a partial order on the set of data, then constructs a topology from the order, then defines continuity from the topology, and finally shows that functions that are definable in the language are continuous. With hindsight, we can start from the end and carry on until we reach the beginning (in Chapter 12 of Part III).

We define a function $f : \mathtt{a} \to \mathtt{b}$ of data types to be *continuous* if it is definable in the language. For example, out of the four functions $\mathtt{S} \to \mathtt{S}$, where $\mathtt{S}$ is the Sierpinski data type, the two constant ones and the identity are continuous, but, as we have already seen, the fourth is not (Lemma 2.1).

Notice the carefully chosen word *definable* as opposed to *computable*: By varying the language under consideration, the notion of continuity will vary. In particular, instead of our base programming language we can use the induced data language with respect to the external view, as it is done in Chapter 4, where one gets a perfect match of synthetic continuity with classical continuity (Theorem 4.1). In this chapter, our base language is Haskell, where in two occasions we consider an extension with a certain computable disjunction operation which is not definable in the language.

All other topological notions considered in this chapter are reduced to that of continuity, with the aid of the Sierpinski data type, and hence they vary together with the notion of continuity. When the coincidence of synthetic and classical continuity holds, one gets classical topological notions (Lemmas 7.1–7.4), and hence we don't bother to attach the qualification *computational* to the notions. The ambiguity of the terminology reflects the ambiguity of the approximation sign $\approx$ in Smyth's dictionary (cf. Chapter 5).

3.2 Open and closed subsets of data types

A subset U of a data type $\mathtt{a}$ is called *open* if its *characteristic function* $\chi_U : \mathtt{a} \to \mathtt{S}$ defined by

$$\chi_U(x) = \mathtt{T} \text{ if and only if } x \in U$$

is continuous. A set is called *closed* if its complement is open.

Because our language is Turing-universal as far as definability of functions $\mathtt{Nat} \to \mathtt{Nat}$ is concerned, a subset of non-divergent elements of $\mathtt{Nat}$ is open if and only if it is r.e., if and only if it is semidecidable. As discussed above, this will change in Chapter 4 (Theorem 4.1).

As another example, the subsets $\emptyset$, $\{\mathtt{T}\}$, and $\{\mathtt{bot}, \mathtt{T}\}$ of the Sierpinski data type are open, because their characteristic functions are the two constant maps and the identity, but the set $\{\mathtt{bot}\}$ is not, because its characteristic function is that considered in Lemma 2.1. If the divergent element of a data type $\mathtt{a}$ belongs to an open set, then all elements of the data type belong to the open set. The reason is that any definable function $\mathtt{a} \to \mathtt{S}$ that maps the divergent program to $\mathtt{T}$ has to send all the elements to $\mathtt{T}$, because, on operational grounds, the program has to produce the output `T` without ever looking at the input. In summary, the only open set with `bot` as a member is the whole space of data.

3.1 PROPOSITION *If $f\colon \mathtt{a} \to \mathtt{b}$ is continuous then $f^{-1}(V)$ is open for every open set $V \subseteq \mathtt{b}$.*

PROOF Because $\chi_{f^{-1}(V)} = \chi_V \circ f$ and because the composite $v \circ f\colon \mathtt{a} \to \mathtt{S}$ is definable if $v\colon \mathtt{b} \to \mathtt{S}$ is. □

Moreover, the inverse-image operation itself is definable:

```
type Open a = a -> S

inverse_image :: (a -> b) -> (Open b -> Open a)
inverse_image(f) = \v -> v.f
```

The converse of the proposition is not true, at least not until we reach Chapter 4.

3.3 Digression — the operational preorder

Notice that, by definition, two programs `x` and `y` are operationally equivalent in the internal sense if and only if $\mathtt{x} \in U \iff \mathtt{y} \in U$ for every open set U. The *internal operational preorder* $\mathtt{x} \sqsubseteq \mathtt{y}$ is defined by requiring that $\mathtt{x} \in U \implies \mathtt{y} \in U$ for every open set U. That is, `y` passes every internal observation that `x` does. By the above development, we conclude that $\mathtt{bot} \sqsubseteq \mathtt{x}$ holds for any `x`. In particular, $\mathtt{bot} \sqsubseteq \mathtt{T}$ in the Sierpinski data type, and this is the historical reason for using the terminologies *bottom* and *top*. (Of course, one can also consider an *external operational preorder*, but then open sets have to be defined relative to observers. See Chapter 4.)

3.2 PROPOSITION *If $f\colon \mathtt{a} \to \mathtt{b}$ is continuous and $\mathtt{x} \sqsubseteq \mathtt{y}$ then $f(\mathtt{x}) \sqsubseteq f(\mathtt{y})$.*

PROOF If V is a an open neighbourhood of $f(\mathtt{x})$ then $f^{-1}(V)$ is an open neighbourhood of `x` by Proposition 3.1 and hence of `y` by the assumption that $\mathtt{x} \sqsubseteq \mathtt{y}$, from which we conclude that V is a neighbourhood of $f(\mathtt{y})$, as required to conclude that $f(\mathtt{x}) \sqsubseteq f(\mathtt{y})$. □

This is summarized by saying that continuous functions are *monotone*. But we shall not have occasion to consider this preorder in this chapter (see Chapter 12 of Part III).

3.4 Intersections and unions of open sets

In any data type, the empty set and the whole space of data are open: Just consider the two constant functions into `S`. In order to see that finite intersections of open sets are open, first consider the conjunction operator, which one can write in infix notation:

```
(/\) :: S -> S -> S
T /\ T = T
```

(There is no mistake in the first line, but we won't pause to explain the idiosyncrasies of the programming language.) The other cases

```
T   /\ bot = bot
bot /\ T   = bot
bot /\ bot = bot
```

hold automatically. Notice that they cannot be given explicitly, because we cannot define a function by stipulating what happens at non-terminating arguments, as illustrated in Lemma 2.1. With this, we now program the intersection operator by

```
intersection :: Open a -> Open a -> Open a
u `intersection` v = \x -> u(x) /\ v(x)
```

where the quotes are used in order to indicate that a function is used as an infix operator. This proves

3.3 Proposition *Finite intersections of open sets are open.*

Regarding unions, it is well known that a function

```
(\/) :: S -> S -> S
```

for disjunction is not expressible in the language. In fact, one requires that the equations

```
T   \/ T   = T
T   \/ bot = T
bot \/ T   = T
bot \/ bot = bot
```

hold, but the evaluation mechanism of the language is sequential and in order to evaluate an expression e_1 `\/` e_2 one would have to evaluate the expressions e_1 and e_2 in an interleaved or parallel fashion, until one of them terminates (necessarily with result `T`). This function is certainly computable though, and the language can be extended with it if required. In the extended language, finite unions of open sets are open:

```
union :: Open a -> Open a -> Open a
u `union` v = \x -> u(x) \/ v(x)
```

Moreover, countable unions of open sets are also open — as above, we regard sequences as functions defined on the natural numbers:

```
countable_union :: (Nat -> Open a) -> Open a
countable_union(s) = \x -> exists(0)
                  where exists(i) = s(i)(x) \/ exists(i+1)
```

3.4 Proposition *In the language extended with the disjunction operation, countable unions of open sets are open.*

However, in this chapter we work with the restricted language whenever possible, clearly indicating when the parallel operation is invoked. In fact, it turns out that the synthetic topology developed below doesn't rely on the closure properties for open sets: They are there, but we don't seem to need to explicitly invoke them. But we'll meet the closure properties again in a proposition that generalizes them.

3.5 Spaces

For one reason or another, one frequently considers subsets of data types, even if they are not expressible in the language. For example, every data type has an extraneous divergent element `bot`, but often we are concerned with the set of non-divergent elements, or some more subtly defined subset. By a *space* we mean an arbitrary subset of a data type. (But see Section 3.13.) If X is a subspace of a data type `a`, we also say that the data type `a` is an *environment* for the space X.

For example, for us the *space of natural numbers* is the subspace N of non-divergent elements of the data type of natural numbers, and the *space of booleans* is the subspace T of non-divergent elements of the data type of booleans.

3.6 The Baire and Cantor spaces

We are particularly interested in two subspaces of the Baire data type defined above: The *Baire space* is the subset B of functions that map the divergent element to itself, and non-divergent elements to non-divergent elements (i.e., the Baire space consists of the strict total functions). The *Cantor space* is the subset C of B consisting of functions taking values 0 or 1 on all non-divergent arguments.

3.7 Continuous maps of spaces

Because subspaces of data types are not necessarily data types, we are forced to work with *relative* topological notions as follows (cf. Chapter 12 of Part III). Let X and Y be subspaces of data types `a` and `b`. We say that a function $\phi\colon X \to Y$ is (relatively) continuous if there is at least one continuous function $f\colon \mathtt{a} \to \mathtt{b}$ with $\phi(x) = f(x)$ for every $x \in X$. We don't care how f behaves on elements of `a` which are outside X (cf. Section 3.13).

3.8 Open and closed subsets of spaces

We say that a subset of a space is (relatively) open if its Sierpinski-valued characteristic map is continuous. The following is immediate from the definitions.

3.5 PROPOSITION *For a subspace X of a data type* `a`*, a subset U of X is open in X iff there is an open subset U' of* `a` *such that $X \cap U' = U$.*

Similarly, we define a notion of (relatively) closed subset of a subspace.

3.6 EXERCISE The subset of all sequences s which belong to the Cantor space and satisfy $s(17) = 0$ is not open in the Baire data type, but it is open in the Cantor space.

3.9 Discrete and Hausdorff spaces

We say that a subspace of a data type is (relatively) *discrete* if its Sierpinski-valued equality map is continuous. For example, the data type of natural numbers is *not* discrete, because one has to take into account the divergent element. However, the space of natural numbers is — we just use the predefined boolean-valued equality test:

```
equal_N :: (Nat,Nat) -> S
equal_N(m,n) = if m == n then T else bot
```

As we have seen, the Baire and Cantor spaces are not discrete, for it takes an infinitely long time to check that two infinite sequences are equal.

3.7 Exercise In a discrete space, singletons consisting of programmable elements are open.

We say that a subspace of a data type is (relatively) *Hausdorff* if its Sierpinski-valued apartness map is continuous. Again, because of the presence of the divergent element, no data type is Hausdorff. However, for example, the space of natural numbers is Hausdorff, and, as we have seen in Chapter 2.4, so is the Baire space. It follows that the Cantor space is also Hausdorff, because an apartness program for a space obviously also works for any subspace. That is, subspaces of Hausdorff spaces are Hausdorff. The same argument shows that subspaces of discrete spaces are discrete.

3.8 Exercise In a Hausdorff space, singletons consisting of programmable elements are closed.

3.10 Compact and overt spaces

We call a subspace Q of a data type `a` *compact* if its universal quantification functional $\forall_Q \colon (\mathtt{a} \to \mathtt{S}) \to \mathtt{S}$ defined by $\forall_Q(p) = \mathtt{T}$ iff $p(x) = \mathtt{T}$ for all $x \in Q$ is continuous. The notion of compactness generalizes that of finiteness: Any finite subspace $\{x_1, \dots, x_n\}$ of definable elements of any data type is compact. Its quantification functional is definable as the nameless program

```
\p -> p(x1) /\ ... /\ p(xn)
```

where $\mathtt{x1}, \dots, \mathtt{xn}$ are programs for $x_1, \dots, x_n$. Of course, one needs a different program for each finite set — the above is just a program scheme.

The space of all elements of any data type `a` is compact, but for trivial reasons: A continuous predicate holds for all elements if and only if it holds for the divergent element, as discussed above:

```
\p -> p(bot)
```

A subset of the space of natural numbers is compact if and only if it is finite, for otherwise we would be able to solve the halting problem. The situation changes radically in the case of non-discrete spaces, but it is still not so easy to find non-trivial examples of compact spaces. For example, the Baire space fails to be compact, as we shall see below.

A subspace O of a data type `a` is called *overt* if its existential quantification functional $\exists_O \colon (\mathtt{a} \to \mathtt{S}) \to \mathtt{S}$ defined by $\exists_O(p) = \mathtt{T}$ iff $p(x) = \mathtt{T}$ for some $x \in O$ is continuous. For example, any overt set of natural numbers is r.e., as shown in Proposition 3.13.

3.9 Exercise Any r.e. set of natural numbers is overt in the language extended with the disjunction operation.

As in this example, overtness results typically rely on the existence of parallel features such as the disjunction operation discussed above.

3.10 Exercise Show that the Baire and Cantor spaces are overt using the disjunction operator. In particular, conclude than an overt set doesn't need to be countable. *Hint.* Enumerate infinite sequences whose finite prefixes exhaust all finite sequences, and argue using the modulus of continuity of a predicate at a sequence as defined in Section 3.11 below.

3.11 Compactness of the Cantor space

In classical topology, the Cantor space is one of the simplest non-trivial examples of a compact space. In the synthetic approach we are considering, compactness of the Cantor space holds in the external view of data but fails in the internal.

If $p(s)$ evaluates to `T` for $p \in (\mathtt{Baire} \to \mathtt{S})$ and $s \in \mathtt{Baire}$, then, on operational grounds, we conclude that only finitely many terms of the sequence s can be inspected before the evaluation terminates, because if an answer is ever produced then this has to happen after finitely many applications of the equations that define the program. We refer to the index of the last inspected term plus one as the (operational) *modulus of continuity* of p at s. If the modulus is zero then no term of the sequence is inspected. By the

Cantor tree we mean the infinite binarily branching tree. We think of a point in the Cantor space as an infinite path in the Cantor tree, starting from the root, where a sequence of digits 0 and 1 is interpreted as a sequence of instructions "turn left" and "turn right".

Each predicate $p \in (\texttt{Baire} \to \texttt{S})$ induces a pruning of the tree: For each s in the Cantor space with $p(s) = \texttt{T}$, we prune the path s at level n, where n is the modulus of continuity of p at s. For the external view of data, if p holds for all s in the Cantor space, then all paths of the resulting tree are finite and hence the tree itself is finite by König's lemma. We refer to the height of this tree as the *uniform modulus of continuity* of p (notice that this is defined only for predicates that hold for all points in the Cantor space).

It is crucial in this last argument that all paths are pruned to finite paths. If we take the internal view, then s in the proof ranges over *computable* sequences, and hence only the computable paths are pruned. As a consequence, there may remain infinite paths. However, they must be non-computable. It is clear that the pruned tree is computable, and so one is tempted to think that there cannot be any non-computable path. But this impression is wrong: Such trees, which are called *Kleene trees*, are known to exist [12]. The tree of a predicate that holds for all computable sequences but fails for at least one non-computable sequence must be infinite, and hence it is not possible to completely traverse it in finite time. Hence compactness fails in the internal view.

But, because, in the external view, the tree is finite, we can hope to traverse it in finite time in order to perform the universal quantification. A simple idea is that a predicate holds for all sequences in the Cantor space iff it holds for those that start with a zero and those that start with a one. This corresponds to searching the left and right subtrees of the predicate. More precisely, the left and right subtrees of a predicate p coincide with the trees of the predicates $p_0(s) = p(\mathrm{cons}(0, s))$ and $p_1(s) = p(\mathrm{cons}(1, s))$. Hence if p has uniform modulus $n + 1$ then p_0 and p_1 have uniform modulus n. This argument shows that not only does $\forall_C \colon (\texttt{Baire} \to \texttt{S}) \to \texttt{S}$ satisfy the equation

```
forall_C(p) = forall_C(\s -> p(cons(0,s)))
           /\ forall_C(\s -> p(cons(1,s)))
```

but also that as the equation is unfolded starting with a universally valid predicate, the modulus of the predicate decreases to 0. From that point on, the predicate doesn't look at its argument anymore. However, it is clear that a finite unfolding of the equation never produces the value T and hence evaluation doesn't terminate. But we are in the right track. What we need is to find a way to probe p. If we could ask p whether it looks at its argument (cf. Longley [86]), then we would be done. However, all we can do in our language is to write down equations, and hence we have a harder task ahead.

In order to be able to probe p, we consider an "if-then" construction on the Sierpinski data type, without an "else" clause:

```
ifs :: (S,a) -> a
ifs(T,x) = x
```

Then the equation

```
ifs(bot,x) = bot
```

holds automatically. Using this, our program is the following:

```
c :: Baire
c = \i -> 0

forall_C :: (Baire -> S) -> S
forall_C(p) = p(ifs(forall_C(\s -> p(cons(0,s))), c))
           /\ p(ifs(forall_C(\s -> p(cons(1,s))), c))
```

What is important about `c` here is that it is a point of the Cantor space, but the particular choice is irrelevant. In order to argue that it works, it is convenient to rewrite it to name some subexpressions:

```
forall_C(p) = p(t0) /\ p(t1)
    where p0(s) = p(cons(0,s))
          p1(s) = p(cons(1,s))
          t0 = ifs(forall_C(p0), c)
          t1 = ifs(forall_C(p1), c)
```

It is not hard to see that the quantifier does satisfy any of the two equivalent equations by considering the two cases $\forall_C(p) = \mathtt{T}$ and $\forall_C(p) = \mathtt{bot}$. As discussed above, it is a general fact that an implicit definition of a function may have more than one solution. We take the operational solution, which is obtained by repeatedly unfolding the equations until a value is reached, or forever so that bottom is computed. As we shall see in Chapter 12, it coincides with the smallest continuous solution in the operational preorder.

We first show that `forall_C`(p) evaluates to T if $p(s)$ evaluates to T for all s in the Cantor space by induction on the uniform modulus of p. If the modulus is zero, then both arguments of the conjunction operator evaluate to `T`, no matter what t_0 and t_1 are, and hence the conjunction itself evaluates to `T`, as required. If p has modulus $n+1$ then p_0 and p_1 have modulus n, as discussed above, and hence `forall_C`(p_0) and `forall_C`(p_1) evaluate to `T` by the induction hypothesis. It follows that t_0 and t_1 evaluate to points in the Cantor space, and, no matter what they are as long as they are members of the Cantor space, $p(t_0)$ and $p(t_1)$ evaluate to `T` and hence so does their conjunction, which concludes our inductive argument.

To complete the proof, we show that if `forall_C`(p) evaluates to `T` then $p(s)$ evaluates to `T` for all s in the Cantor space. The argument considers the number of unfoldings of the equation that defines `forall_C`(p) performed by the evaluation procedure. If this number is one, then no information about t_0 and t_1 is available and hence $p(t_0) \wedge p(t_1)$ must have evaluated to `T` without p looking at its arguments t_0 or t_1, i.e., p must have modulus of continuity 0. Hence $p(s)$ must evaluate to `T` for every s in the Cantor space, as required. More generally, if `forall_C`(p) evaluates to `T` in 2^n unfoldings or fewer, then p has uniform modulus of continuity n or smaller, and hence must be universally valid. We have provided the base case of the inductive argument. The routine inductive step is left to the reader. We present a complete proof of a generalization of this program in Chapter 13.

In summary:

3.11 PROPOSITION *Compactness of the Cantor space*

1. *holds in the external view of data, but*
2. *fails in the internal view.*

What is going on here is that the definition of the quantification functional is relative to what we mean by an element of a data type, so we end up with two different definitions when we specialize it to the internal and external views. The above program satisfies one of the resulting specifications, but not the other. The statement that a predicate is universally valid with respect to the external view is stronger (but often easier to prove when it holds) than the statement that it is universally valid with respect to the internal view.

3.12 Basic topology

Now that we have plenty of definitions and at least one example of a non-trivial compact space, let's prove some theorems about them. To be precise, let's write some programs. For convenience, we introduce a type for quantifiers:

```
type Quant a = (a -> S) -> S
```

3.12 PROPOSITION *If X is Hausdorff and $Q \subseteq X$ is compact, then Q is closed.*

PROOF Let `a` be an environment for the space X, `apart_X :: (a,a) -> S` be an apartness program for X and `forall_Q :: Quant a` be a quantifying program for Q. Then the characteristic map of the complement of Q can be programmed by

```
complement_Q :: Open a
complement_Q = \x -> forall_Q(\y -> apart_X(x,y))
```

That this performs the required job follows from the fact that $x \notin Q$ if and only if, for all $y \in Q$, $x \neq y$. □

In fact, functional programmers will have already realized that we can say more: Not only is the characteristic map of the complement of Q definable, but also we can construct it, by means of a program, from the quantifier of Q and an apartness map of X. We don't have a good name for the program, but in any case we want to keep it short for layout reasons:

```
c :: (Quant a, (a,a) -> S) -> Open a
c(forall_Q,apart_X) = \x -> forall_Q (\y -> apart_X(x,y))
```

Then

```
complement_Q = c(forall_Q,apart_X)
```

That is, the result holds *uniformly* in the sense of recursion theory [108]. This is also the case for the following propositions, but we omit the routine details.

The following dual proposition with dual proof won't be very exciting to topologists, but it confirms what is expected from a discrete set over which one can existentially quantity in a computational fashion: It must be r.e.

3.13 PROPOSITION *If X is discrete and $O \subseteq X$ is overt, then O is open.*

PROOF Let `a` be an environment for the space X, `equal_X :: (a,a) -> S` be an equality program for X and `exists_O :: Quant a` be a quantifying program for O. Then the characteristic map of O can be programmed by

```
chi_O :: Open a
chi_O = \x -> exists_O(\y -> equals_X(x,y))
```

That this performs the required job follows from the fact that $x \in O$ if and only if there exists $y \in O$ with $x = y$. $\square$

3.14 PROPOSITION *If X is compact and $F \subseteq X$ is closed then F is compact.*

Here we need the disjunction operation discussed above.

PROOF Let `forall_X :: Quant a` be the quantifying program for X, where `a` is an environment for the space X, and `complement_F :: a -> S` be the program for the characteristic map of the complement of F. Then the quantifying program for F is defined by

```
forall_F :: Quant a
forall_F(p) = forall_X(\x -> complement_F(x) \/ p(x))
```

That this performs the required job follows from the fact that $\forall x \in F.p(x)$ iff $\forall x \in X.x \in F \implies p(x)$ iff $\forall x \in X.x \notin F \vee p(x)$. $\square$

3.15 EXERCISE Prove the dual of the above proposition, namely that an open subspace of an overt space is overt. The parallel operation is not needed.

3.16 PROPOSITION *If f:* `a` $\to$ `b` *is a continuous function and $Q \subseteq$* `a` *is compact, then its direct image $f(Q)$ is compact.*

PROOF If `f` is a program for f and `forall_Q` is a program for quantification over Q, then the following program clearly quantifies over $f(Q)$:

```
forall_fQ :: Quant b
forall_fQ(p) = forall_Q(\x -> p(f(x)))
```

$\square$

This can be applied to conclude that the Baire subspace of the Baire data type is not compact, as claimed above. If it were compact, then its direct image under e.g. the continuous map

```
f :: Baire -> Nat
f(s) = s(0)
```

would be compact, but this is absurd because the image is the space of natural numbers, which, as we have seen, is not compact. This holds for both the internal and external views of data. For the same reasons, we conclude that any continuous image of the Cantor space in the space of natural numbers is finite if we take the external view of data. This fails if we take the internal view, with a counter-example again using Kleene trees.

3.17 Exercise A similar proposition for direct images of overt subspaces with a similar proof holds.

3.18 Proposition *A product of two compact spaces is compact.*

Proof If Q and R are compact subspaces of data types `a` and `b` with quantification programs `forall_Q` and `forall_R`, then the following program quantifies over $Q \times R$:

```
forall_QtimesR :: Quant (a,b)
forall_QtimesR(p) = forall_Q(\x -> forall_R(\y -> p(x,y)))
```

That this performs the required job follows from the fact that $\forall z \in Q \times R.p(z)$ iff $\forall x \in Q.\forall y \in R.p(x,y)$. $\square$

3.19 Exercises Similarly, a product of two overt spaces is overt. A product of two discrete spaces is discrete. Assuming the parallel disjunction operation, a product of two Hausdorff spaces is Hausdorff (however, for many particular examples, the disjunction operation is not needed).

How does one observe a continuous function? A simple idea is that we run it for a particular input and then check whether its output lands in a given open set. But we can do better than that:

3.20 Proposition *If $Q \subseteq$ `a` is compact and $V \subseteq$ `b` is open then the set*

$$N(Q,V) = \{f \in (\mathtt{a} \to \mathtt{b}) \mid f(Q) \subseteq V\}$$

is open.

Proof If `forall_Q` is the quantifier of Q and `v` is a program for the characteristic function of V then the following is a program for the characteristic function of the set displayed above:

```
nQV :: (a->b) -> S
nQV(f) = forall_Q(\x -> v(f(x)))
```

That this performs the required job follows from the fact that $f \in N(Q,V)$ if and only if $\forall x \in Q.f(x) \in V$. $\square$

Open sets of this form are known in classical topology: They form the subbase that defines the so-called *compact-open topology* on the set of continuous maps (Chapter 8.5). As we shall see in Chapter 12, one cannot do better than the above proposition, at least when the disjunction operation is available: After taking finite intersections and then unions of the above sets, all observable properties are exhausted.

Proposition 3.21(1) below is perhaps not so familiar to topologists, but it does have a classical topological manifestation (Proposition 9.17). We have seen that finite intersections of open sets are open. This generalizes from finite sets to compact sets. In fact, because `Open a` is a data type like any other, one can speak about its compact subsets, and hence, identifying open sets with their characteristic functions, we can talk about compact sets of open sets.

3.21 Proposition (Closure properties for open sets)

1. *If a set of open sets is compact, then its intersection is open.*
2. *If a set of open sets is overt, then its union is open.*

Proof (1): Let `forall_Q :: Quant (Open a)` be a program for quantifying over a compact set $\mathcal{Q}$ of (characteristic functions of) open sets of a data type `a`. Because $x \in \bigcap \mathcal{Q}$ iff $\forall U \in \mathcal{Q}. x \in U$, the following is a program for the intersection of $\mathcal{Q}$:

```
intersection_of_Q :: Open a
intersection_of_Q = \x -> forall_Q(\u -> u(x))
```

(2): Similar, using the fact that $x \in \bigcup \mathcal{O}$ iff $\exists U \in \mathcal{O}. x \in U$. □

Notice that the proof of the second item *doesn't* need the disjunction operator. However, as we have already mentioned, in order to show that sets of interest are overt, one invariably needs the disjunction operator.

3.13 Revision of the notion of space

Consider the following assertion and proof:

3.22 Proposition *If X is an overt subspace of a data type* `a` *and Y is a Hausdorff subspace of a data type* `b`*, then the space $(X \to Y)$ consisting of the functions* (`a` $\to$ `b`) *that map X into Y is Hausdorff.*

Proof We can program the apartness map `apart_XtoY` of $(X \to Y)$ from the existential quantifier `exists_X` of X and the apartness map `apart_Y` of Y by

```
apart_XtoY :: (a->b, a->b) -> S
apart_XtoY(f,g) = exists_X(\x -> apart_Y(f(x),g(x)))
```

□

This program doesn't quite perform the advertised job. By definition, the space $(X \to Y)$ consists of the functions `a` $\to$ `b` that map X into Y, but there are, in general, different such functions that have the same behaviour on X. The apartness program defined above doesn't distinguish them, as it shouldn't. To fix the above incorrect statement for our correct proof, we can attempt to give $(X \to Y)$ the quotient topology of the subspace topology. However, it seems more reasonable and in line with practice to think that the notion of space is better captured by an equivalence relation on a subset of a data type, rather than just a subset, and hence Scott's *equilogical spaces* are probably the natural tool to apply in this context [114]. In some cases, such as the Baire and Cantor spaces, it is possible to work with canonical representatives of equivalence classes, as we have done above.

We have a dual proposition, with the same revised interpretation of the notion of space (as a subset with an equivalence relation):

3.23 Proposition *If X is a compact subspace of a data type* `a` *and Y is a discrete subspace of a data type* `b` *then the space $(X \to Y)$ is discrete.*

Proof We define the equality map of $(X \to Y)$, with the identifications discussed above, from the universal quantifier of X and the equality map of Y:

```
equal_XtoY :: (a->b, a->b) -> S
equal_XtoY(f,g) = forall_X(\x -> equal_Y(f(x),g(x)))
```

□

Recall that N is the space of natural numbers and T is space of booleans (see Section 3.5). Hence $(N \to T)$ is another manifestation of the Cantor space.

3.24 Corollary *Under the external view of data, $((N \to T) \to N)$ has semidecidable equality and semidecidable apartness.*

Because our language doesn't have a mechanism for gluing semidecision procedures for a set and for its complement, it doesn't immediately follow that this space has decidable equality. However, a proof that it does (which can be read at this point) and other surprising computational facts (which depend on more advanced material) are contained in Chapter 13.

3.14 Notes

It follows easily from what is known about function-space topologies that the classical notion of compactness coincides with the one given here (Lemma 7.4 and Chapter 8). This seems to have been first pointed out and exploited by Taylor [131] and the author independently and from different perspectives.

Dubuc and Penon [30] have an interesting notion of a compact object of a topos, expressed in the internal language, which amounts to a certain Frobenius condition (as known in locale theory) for the universal quantifier (which always exists in topos theory). They also consider other topological notions, for example that of a Hausdorff object. There must be connections of their approach with abstract Stone duality and the ideas reported here, but we haven't discovered them at the time of writing.

The functional program for universal quantification over the Cantor space provided here is due to the author but it is related to a program formerly discovered by Berger [13], which we present in Chapter 13. The fact that compactness of the Cantor space is susceptible to considerations such as the one made here is well known in logic and recursion theory, with Kleene trees playing the same role.

The given formulation of the notion of overt space was discovered by Taylor, but the notion itself was originally introduced in locale theory as developed in arbitrary toposes by Joyal with a different form of definition and under a different name [70, 72]. The notion also occurs in formal topology via positivity predicates [22]. In classical topology, the notion of overt space plays no role at all, and hence classical topologists will necessarily miss the point: Every subspace of any space is overt (Lemma 7.5). As opposed to the topos of sets, classical logic doesn't necessarily hold in an arbitrary topos (i.e. the principle of excluded middle and the general axiom of choice may fail), as is the case for instance for the topos of sheaves on a topological space, and this makes the notion non-vacuous. Under classical logic, which is what we are assuming here, what makes it non-vacuous is the requirement of computability.

The Sierpinski space is a common tool in computer science, arising as the typical space of results of observations, as already emphasized by Smyth. For other kinds of computation, e.g. non-deterministic or probabilistic, one considers different spaces of results of observations [104]. Perhaps, for computational applications, one should develop topology relativized to a given space of results of observations.

The translations of topological notions such as those of discrete and Hausdorff space in terms of the Sierpinski space are obvious, and so are their computational interpretations — at least when one has seen them. But, to the best of our knowledge, they haven't been explicitly formulated or exploited, except in the work by Taylor and by the author.

Taylor formulates the quantifiers by adjoint conditions [131], as it is done in topos theory [72]. He formulates the notions of open, closed, discrete and Hausdorff objects by the existence of certain pullbacks that arise in topos theory, with the subobject classifier of a topos replaced by the Sierpinski object. His abstract Stone duality is based on the discovery that Paré's theorem for toposes and a certain Stone-type duality in topology can be regarded as instances of the same phenomenon. The duality here is that between distributive continuous lattices in the sense of Dana Scott and locally compact sober spaces, which is due to Hofmann and Lawson [62]. Taylor has developed a translation of abstract Stone duality into the logic programming language *Prolog*, which is briefly discussed in the last chapter of the paper [133], but this doesn't seem to be related to the Haskell programs presented here.

The proofs of topological statements via functional programs reported in this chapter (and via the λ-calculus reported in Chapter 9) were discovered by the author, but some constructions in abstract Stone duality can also be regarded as functional programs and there is some overlap.

During a visit of Dana Scott to the University of Birmingham in England in January 2001, the author communicated the approach to topology reported here and in Chapter 9. Scott saw this as an opportunity to exploit his recent *equilogical spaces* and wrote it down together with Andrej Bauer in an unpublished note [11].

Notice that we have taken a purely operational, rather than denotational, view of data and programs in this chapter. This is because we wanted to justify the topological view of data types from first computational principles. The denotational view occurs in the proof of Theorem 4.1 and in Chapter 13 and is briefly introduced in Chapter 12.

Chapter 4

Computability versus classical continuity

In the previous chapter we defined computational versions of topological notions, using topological terminology for the computational concepts. We now revert to the classical topological meanings of the terms, and so we require some rudimentary background on topology (perhaps in the form of domain theory) at this point.

The topology of a data type is somehow induced by its computational structure. With this in mind, it is not entirely surprising that

Computability implies continuity.

Indeed, we made this fact into a sensible definition in the synthetic approach to the topology of data types developed in Chapter 3. For the classical notion of continuity, the converse of the statement fails. We shall exhibit counter-examples in due course, but, for the moment, a cardinality argument suffices: In general, there are uncountably many continuous functions, but only countably many computable functions.

Nevertheless, one is entitled to ask to what extent the converse holds.

4.1 The Myhill–Shepherdson and Rice–Shapiro theorems

One precise answer for the data type $(\mathbb{N} \rightharpoonup \mathbb{N})$ of partial functions on the natural numbers is given by the Myhill–Shepherdson theorem: *Every effectively continuous functional* $(\mathbb{N} \rightharpoonup \mathbb{N}) \to (\mathbb{N} \rightharpoonup \mathbb{N})$ *is computable.* The

Rice–Shapiro theorem is about the extent to which openness implies semidecidability, this time for a different data type: *Every effectively open set of* $\mathcal{P}\,\mathbb{N}$ *is semidecidable*. The precise formulations and proofs of these two theorems can be found in e.g. Rogers' book [108].

4.2 Classical topology of data types

Here we explore a different type of answer. For the sake of mathematical rigour, we consider the programming language PCF and some of its standard extensions (for its call-by-name evaluation strategy). This can be regarded as a subset of the language discussed in the previous chapter, and, in fact, all the programs written in the previous chapter could have been written in PCF instead. Readers who don't know PCF have two options: They can (1) safely rely on the previous chapter, ignoring some technical details, or else (2) pause to read e.g. Streicher's excellent notes on *mathematical foundations of functional programming*. If option (1) is taken, it won't be possible to make sense of the proof of Theorem 4.1 until Chapter 12 is reached — but it should be profitable to just understand its formulation at this point.

In Chapter 2 we introduced a data language, for a given base programming language, in order to make sense of programs computing with data coming from the external environment. Because we have function types in the language, we have a notion of function coming from the environment. We prove that the functions that come from the environment are precisely the classically continuous ones, where the notion of open set is defined relatively to the environment, rather than relatively to the programming language as in the previous chapter. The idea is that the classically open sets are precisely the observable, not necessarily semidecidable, properties. Our base programming language is PCF^{++} and our data language is $\mathrm{PCF}^{++}_{\Omega}$. Human beings write programs in PCF^{++}, or perhaps just PCF, and ideal observers living in the environment prepare their data and experiments in $\mathrm{PCF}^{++}_{\Omega}$, which programs written by human beings can process.

1. $\mathrm{PCF}^{+} = \mathrm{PCF}$ extended with parallel-or.

   (This will implement the requirement that finite unions of open sets be open.)

2. $\mathrm{PCF}^{++} = \mathrm{PCF}^{+}$ extended with the parallel existential quantifier.

   (This will implement the requirement that arbitrary unions of open sets be open.)

3. $\mathrm{PCF}^{++}_{\Omega} = \mathrm{PCF}^{++}$ extended with constants of type $\mathtt{Nat} \to \mathtt{Nat}$, one for each sequence of natural numbers, representing potential inputs provided by external observers.

   (This will implement the requirement that "arbitrary" really means arbitrary in the axiom for closure under unions for open sets — cf. the discussion in Chapter 1 regarding disjunctions of observable properties.)

   The operational semantics of $\mathrm{PCF}^{++}_{\Omega}$ is defined in the same way as for PCF^{++}, with obvious rules for evaluating inputs, and so is operational equivalence.

4. For each type σ, the topological space X_σ is defined as follows:
   (a) Its points are the equivalence classes of $\mathrm{PCF}^{++}_{\Omega}$ programs of type σ.
   (b) A set $U \subseteq X_\sigma$ is called open if the function

$$\begin{array}{rcl} \chi_U \colon X_\sigma & \to & X_{\mathtt{Bool}} \\ x & \mapsto & \begin{cases} \mathrm{true} & \text{if } x \in U, \\ \bot & \text{if } x \notin U \end{cases} \end{array}$$

   is definable in $\mathrm{PCF}^{++}_{\Omega}$. Here "true" is the equivalence class of the term "`True`" and $\bot$ is the equivalence class of divergent terms of type `Bool`. Definability of χ_U amounts to the requirement that there is a term $F\colon \sigma \to \mathtt{Bool}$ such that, for every term M of type σ, one has that $F(M) = \mathtt{True}$ if the equivalence class of M belongs to U and $F(M)$ is a divergent term otherwise. Notice that this function takes values in the Sierpinski subspace $\{\bot, \mathtt{true}\}$ of the boolean data type, which is not directly available as a data type on its own in PCF.

For the expert reader, we remark that we are *not* invoking the operational preorder or any denotational semantics for the language in these definitions. However, they do occur in the proof of the following.

4.1 Theorem

1. *The open sets of X_σ form a topology, that is, they are closed under the formation of finite intersections and arbitrary unions.*

2. *A function $f\colon X_\sigma \to X_\tau$ is definable in PCF_Ω^{++} iff it is continuous.*

Proof (Sketch) Interpret PCF_Ω^{++} in the standard Scott model [113, 101] of PCF^{++}. Replace recursive sequences by inputs, i.e. arbitrary sequences, in Plotkin's proof [101, Theorem 5.1] of Turing-universality of PCF^{++} to prove that every element of D_σ is definable in PCF_Ω^{++}, where D_σ is the interpretation of the type σ in the model. Computational adequacy of the model holds for PCF_Ω^{++} with the same proof as that for PCF^{++} [101, Theorem 3.1]. Hence the domain order of D_σ is isomorphic to the partial-order reflection of the operational preorder on closed terms of type σ. It follows that the open sets of X_σ are the Scott open sets of its operational partial order. This concludes the proof (1). Because $D_{\sigma\to\tau}$ under the Scott topology is homeomorphic to $X_{\sigma\to\tau}$ and because $f\colon X_\sigma \to X_\tau$ is continuous iff $f \in X_{\sigma\to\tau}$, (2) follows. □

Thus, in this setting,

$$\begin{array}{rcl} \text{computable} & \Longrightarrow & \text{continuous,} \\ \text{continuous} & \Longrightarrow & \text{computable relatively to external inputs.} \end{array}$$

For data types D and E, the function type $(D \to E)$ consists of the continuous functions from D to E, rather than *all* functions or just the *computable* ones. Thus, the language articulates a notion of computable function on continuous data. A particular instance of this situation is a functional such as

$$F\colon (C \to D) \to E$$

We are typically interested in the case in which F is computable. However, the above development tells us that it is appropriate to take the input of F to be a continuous, not necessarily computable, function $f\colon C \to D$. If the function f happens to be computable, then so will be $F(f)$, because computable functions preserve computability.

Notice that one way of showing that F is not computable is to prove that it is not continuous. The converse fails in general, but it is a fact of experience that it often holds in practice, which can be used as a guideline to successfully conjecture that certain function(al)s are computable.

Notice also that, because we have encoded open sets as semidecision functions, the above theorem also gives:

$$\begin{aligned}
\text{semidecidable} &\implies \text{open},\\
\text{open} &\implies \text{semidecidable relatively to external inputs.}
\end{aligned}$$

The above proof shows that the topologies that we get are Scott topologies. This is compatible with, and indeed explains, the fact that not all functions are computable relatively to not-necessarily-computable inputs, one example being the function on the booleans that maps $\perp$ to true, and true and false to $\perp$. This is the case despite the fact that e.g. an enumeration of the complement of the halting set is allowed as an input.

4.3 Notes

In summary, Chapter 3 shows that synthetic topology can be developed in a variety of languages, and this one shows that, for a particular language, synthetic topology coincides with classical topology. For the full coincidence of all topological notions discussed in Chapter 3, we further need the results of Chapters 7, 8 and 12. As far as this chapter is concerned, for this coincidence to hold, we need (i) external inputs in the data language, (ii) synthetic topological notions defined relative to the data language rather than to the base programming language, and (iii) parallel features (in the observer's language but not necessarily in the base programming language).

Notice that (ideal or human) observers can externally compute parallel-or by observing (the external effect of) computations of pairs of programs of type `Bool`. This is true for the existential quantifier as well, if we either assume that we are allowed to have access to countably many copies of the PCF black box that computes the input predicate, or else we are allowed to restart and abort computations of the black box. Hence we advocate that it is reasonable to take the observation language (or data language) to be PCF_Ω^{++} even if we choose our base language to be PCF rather than PCF^+ or PCF^{++}. From the point of view of recursion theory, parallel-or and the parallel existential quantifier correspond to *dovetailing* [108], and hence it is natural to include them.

However, there are good reasons to exclude them for certain purposes — cf. Longley's work on computability at higher types [87, 86]. If parallel-or and/or the existential quantifier are not included in the observer's language, the synthetic open sets don't form classical topologies. In any case, as we have seen in Chapter 3, it is possible to develop a good deal of topology even when the open sets don't form topologies in the classical sense, but we haven't explored this avenue in more detail than already reported in Chapter 3.

We remark that, although the topologies that one gets in this chapter *are* closed under the formation of arbitrary unions, and hence are classical as claimed, the methods used here are fundamentally different from the ones used in Chapter 3 to obtain restricted versions of the closure properties. Here we have argued using a domain-theoretic model of the language which is known, by mathematical means, to have this classical closure property, whereas in Chapter 3 we constructed programs to implement the operations in the countable case (Proposition 3.4) and the overt case (Proposition 3.21(2)). But, because in classical topology all sets are overt (Lemma 7.5), the closure property established via the model also holds synthetically within the data language, using Proposition 3.21(2). In order to exploit the closure property implemented by this proposition, a collection of open sets has to be presented via its existential quantifier (rather than via an enumeration as in Proposition 3.4). This gives a possible answer to the question, posed in Chapter 1, of how a truly arbitrary collection of observable properties can be presented to an observer. One way of presenting a not-necessarily-countable subset of a data type is via a search method for it, which is precisely what an existential quantifier is. Here we apply this idea to the data type of observable properties (function type with values in the Sierpinski space).

As discussed above, in the absence of parallel features, the scope of synthetic topology remains to be investigated, not only at the level of (programming and data) languages, but also at the level of their mathematical models. However, notice that because, for instance, various categories of games [2, 66] are models of PCF, the synthetic topology developed in Chapter 3 applies to them. Just as the notion of overt space, which has no counterpart in classical topology (Lemma 7.5), emerges in Chapter 3 for computational reasons, other classically invisible topological notions are likely to emerge for sequentiality reasons in the investigation of the synthetic topology of such models.

Chapter 5

Revised and expanded edition of Smyth's dictionary

Assuming that observers live in an external environment which is not necessarily restricted to the laws of Turing computation, Chapters 2–4 elucidate the distinction between the notions of semidecidable and observable property, and their relationship to that of topologically open set, at least if one believes that the given mathematical definition of observation is reasonable. In summary, a property of elements of a data type is *semidecidable* iff its Sierpinski-valued characteristic function is definable in the programming language, and it is *observable* iff its characteristic function is definable in the data language.

If, in addition, one assumes that the data language includes the parallel constructs discussed in Chapter 4, some occurrences of the approximation sign $\approx$ in Smyth's dictionary become equalities and others implications as follows:

> *Data type* $=$ *topological space (of a certain kind).*
>
> *Piece of data* $=$ *point (computable or not).*
>
> *Semidecidable property* $\Rightarrow$ *observable property* $=$ *open set.*
>
> *Computable map* $\Rightarrow$ *map computable by observer* $=$ *continuous map.*

If the data language is sequential, then the above equality signs can be taken as synthetic formulations of "sequential" topological notions, but we don't pursue this subject here. However, to be consistent, for sequential programming and data languages, one should speak of *sequentially* semidecidable and observable properties in the above entries.

Also based on the previous development, but depending on topological material developed in Parts II and III, we include:

Subspace of data type with semidecidable equality $\Rightarrow$ *discrete space.*

Subspace of data type with semidecidable apartness $\Rightarrow$ *Hausdorff space.*

Computationally universally quantifiable set
$\Rightarrow$ *continuously universally quantifiable set* $=$ *compact set.*

Function type $=$ *function space.*

The last entry is based on Proposition 3.20 and on Chapter 8 of Part II. But, before taking care of the unresolved entries of the dictionary, we pause to address some concrete aspects of the topology of computation.

Chapter 6

Computationally induced classical topologies

We mentioned in Chapter 4 that it is not terribly surprising that computable functions are continuous with respect to computationally induced topologies. What *is* surprising is that these topologies are familiar. In fact, this is what justifies the terminologies e.g. *Cantor space* and *Baire space* from classical topology that were adopted in Chapter 3 to designate (certain subsets of) well-known domains of computation. In this chapter we consider the classical topology of these and other domains in a programming-language-independent fashion.

6.1 The Cantor space

Let $2 = \{0, 1\}$ be the set of bits (binary digits) and consider computations of functions $f \colon 2^\omega \to 2^\omega$, where 2^ω denotes the set of infinite sequences of bits:

$$\xrightarrow{\quad s_0 s_1 s_2 \cdots \quad} \boxed{\; f \;} \xrightarrow{\quad t_0 t_1 t_2 \cdots \quad}$$

$$s = s_0 s_1 s_2 \cdots \qquad t = t_0 t_1 t_2 \cdots$$
$$f(s) = t.$$

Think of the sequences s and t as the complete histories of the input and output, including the future. The black box alternates between reading some digits from the input, performing some internal computations and writing some digits to the output. Bad input suppliers will provide a finite sequence of digits and then give up — these are ruled out of consideration for the moment. Bad black boxes will engage into infinite internal computations at some point, neglecting the output forever — these are also ruled out for the moment.

6.1 Example $h\colon 2^\omega \to 2^\omega$ defined by

$$h(s) = t \text{ where } t_i = \bar{s_i} \text{ (digit negation).}$$

This is clearly computable. We emphasize again that there is no need to restrict attention to computable inputs.

6.2 Counter-Example $f\colon 2^\omega \to 2^\omega$ defined by

$$\begin{aligned} f(0^{k_0}10^{k_1}10^{k_2}\cdots 0^{k_n}10^\omega) &= 0^{k_0}h_0 0^{k_1}h_1 0^{k_2}\cdots 0^{k_n}h_n 0^\omega \\ f(0^{k_0}10^{k_1}10^{k_2}\cdots 0^{k_n}1\cdots) &= 0^{k_0}h_0 0^{k_1}h_1 0^{k_2}\cdots 0^{k_n}h_n \cdots \end{aligned}$$

where

$$h_n = \begin{cases} 1 & \text{if } k_n \text{ belongs to the halting set,} \\ 0 & \text{otherwise,} \end{cases}$$

is *not* computable.

6.3 Counter-Example $g\colon 2^\omega \to 2^\omega$ defined by

$$g(s) = \begin{cases} 10^\omega & \text{if } \forall i \in \omega,\ s_i = 0, \\ 0^\omega & \text{otherwise,} \end{cases}$$

is *not* computable either.

But the reasons are fundamentally different:

1. The halting set is undecidable.
2. The first digit of the output depends on infinitely many digits of the input.

A black box could compute f if antiprotonic computers were built in order to decide the halting set (of Turing machines — that of antiprotonic computers would require a further technological development, as Turing's self-referential argument for undecidability of the halting problem is bound to apply to antiprotonic computers as well). However, to compute g, a black box would have to be in possession of a crystal ball, because supplying, and hence reading, the whole input takes forever. In any case, based on what went wrong with g, we can say

> *If a function $f\colon 2^\omega \to 2^\omega$ is computable, then finite parts of its output must depend only on finite parts of its input.*

As Counter-example 6.2 shows, this necessary condition is not sufficient to characterize computability.

More formally, define

$$s =_n t \iff \forall i < n,\, s_i = t_i.$$

Then the condition amounts to

$$\forall s \in 2^\omega\, \forall \epsilon \in \omega\, \exists \delta \in \omega\, \forall t \in 2^\omega,\; s =_\delta t \implies f(s) =_\epsilon f(t).$$

We say that f is of *finite character*.

6.4 Proposition *Endow* 2 *with the discrete topology and* 2^ω *with the product topology. Then* $f\colon 2^\omega \to 2^\omega$ *is of finite character iff it is continuous.*

Proof This readily follows from the definition of product topology.

But it may be helpful to consider a more complicated proof. Define

$$d(s,t) = \inf\{2^{-n} \mid s =_n t\}.$$

Then, as is well known [122], d is a metric that induces the topological product on the set-theoretical product, and it is clear that the ϵ–δ definition of continuity w.r.t. d coincides with the ϵ–δ definition of the notion of finitary character. □

This topology is called the *Cantor topology* because it makes the product space homeomorphic to the Cantor middle-third set of the closed unit interval $[0,1]$ with the relative topology. For us, it has computational significance:

$$U \subseteq 2^\omega \text{ is open} \iff \forall s \in U\, \exists n\, \forall t =_n s,\, t \in U.$$

That is, if s passes a test U, then it has a finite part such that every t sharing this part also passes the test. In this sense, the property of belonging to U is an observable one, albeit not necessarily a semidecidable one, as it may be necessary to perform a non-computable operation on the finite part of s in order to check whether s indeed passes the test.

6.2 The Kahn domain

We have ruled out bad input suppliers and bad black boxes. Let's now allow them. Then a black box of the kind we are considering is best modelled by a function

$$f\colon 2^\infty \to 2^\infty,$$

where

$$2^\infty = 2^* \cup 2^\omega$$

and 2^* is the set of finite (possibly empty) sequences of bits.

Because outputs, once written out, cannot be retracted, computable functions have to be *monotone*:

$$s \text{ is a prefix of } s' \implies f(s) \text{ is a prefix of } f(s').$$

Again, such functions have to be of finite character — but this time we don't need the relations $=_n$ to express the condition:

> *If t is a finite prefix of $f(s)$, then there is a finite prefix s' of s such that t is already a prefix of $f(s')$.*

6.5 Proposition *$f\colon 2^\infty \to 2^\infty$ is monotone and of finite character iff it is continuous w.r.t. the Scott topology of the prefix order of 2^∞.*

Proof See Chapter 12 . $\square$

In general, the Scott topology of a directed complete poset (dcpo) is defined by saying that a set U is open iff it is an upper set and every directed set with join in U actually intersects U (see Chapter 12). In this case (and more generally in algebraic dcpos), the second condition can be simplified to

$$\forall s \in U \exists \text{ a finite prefix } s' \in U \text{ of } s \text{ s.t. } \forall t \text{ with } s' \text{ as a prefix}, t \in U.$$

We leave the proof of the following as an exercise.

6.6 Proposition *The Cantor topology of 2^ω coincides with the subspace topology of the Scott topology of 2^∞.*

6.7 Corollary *Suppose a potentially bad function $g\colon 2^\infty \to 2^\infty$ turns out to be good i.e.*

$$\begin{tikzcd}
2^\omega \arrow[r, "f"] \arrow[d, hook] & 2^\omega \arrow[d, hook] \\
2^\infty \arrow[r, "g"'] & 2^\infty
\end{tikzcd}$$

for some (necessarily unique) $f\colon 2^\omega \to 2^\omega$. If g is continuous then so is f.

Proof This is a standard property of subspace topologies. □

Computationally, it is clear that whenever we implement a black box $f\colon 2^\omega \to 2^\omega$ we are in reality implementing a black box $g\colon 2^\infty \to 2^\infty$ such that the above diagram commutes. Topologically, we have:

6.8 Proposition *Every continuous $f\colon 2^\omega \to 2^\omega$ extends to at least one continuous function $g\colon 2^\infty \to 2^\infty$ (in the sense of the above diagram).*

Proof A direct proof of this fact is not difficult and is an interesting exercise. A more abstract proof that uses (and partly motivates) the material developed in Chapter 12 has the advantage of applying to many similar situations encountered here and elsewhere: The embedding $2^\omega \hookrightarrow 2^\infty$ is dense (because Scott closed sets are lower sets), and 2^∞, being a Scott domain under the Scott topology, is injective over dense topological embeddings. □

Notice that the space 2^ω is Hausdorff but 2^∞ isn't. As we have already seen, this is typical of data types: They are usually non-Hausdorff (in fact typically domains under the Scott topology), but we are actually interested in distinguished Hausdorff subspaces. That is, a domain serves as an *environment* for the Hausdorff space we wish to compute with. In this, and many other but not all examples, the Hausdorff space is that of maximal elements of the domain. A counter-example occurs in Chapter 3: It turned out to be convenient to work with the Cantor space using the Baire data type as a computational environment.

6.3 The real line

For simplicity, we consider the unit interval $[0,1]$, to begin with, and then the interval $[-1,1]$. There are many approaches. We consider three, of which the first is flawed.

We may compute with reals via their binary expansions (as Turing [139] did):

$$\begin{array}{rcl} [\![-]\!]\colon 2^\omega & \twoheadrightarrow & [0,1] \\ s & \mapsto & \sum_{i\in\omega} s_i 2^{-i-1}. \end{array}$$

Think of s as $0.s_0s_1s_2\cdots$. Notice that dyadic numbers (i.e. numbers of the form $m/2^n \in [0,1]$ with m and n integer) have two binary expansions.

6.9 Proposition *The quotient topology on $[0,1]$ induced by this surjection is the usual Hausdorff topology.*

PROOF With this topology, it is easy to check that the map is continuous. But continuous surjections of compact Hausdorff spaces are always quotient maps. □

6.10 COROLLARY *In a situation*

$$\begin{array}{ccc} 2^\omega & \xrightarrow{f} & 2^\omega \\ \downarrow & & \downarrow \\ [0,1] & \xrightarrow[g]{} & [0,1], \end{array}$$

if f is continuous, then so is g.

PROOF This is a standard property of quotient topologies. □

We refer to a map f such as that in the above diagram as a *realizer* of g, and we say that g is *computable* with respect to binary notation if it has at least one computable realizer.

6.11 COROLLARY *Digitally computable functions on* $[0,1]$ *are continuous.*

The converse of Corollary 6.10 fails badly. Because we have ten fingers, we illustrate this using decimal notation, but the readers should convince themselves that the choice of base is unimportant, as long as it is an integer bigger than 1:

$$(10) = \{0,1,2,\ldots,9\}, \qquad \begin{array}{ccc} (10)^\omega & \xrightarrow{f} & (10)^\omega \\ \downarrow & & \downarrow \\ [0,1] & \xrightarrow[g]{} & [0,1]. \end{array}$$

The bad news is that most continuous functions g don't have continuous realizers f, e.g.

6.12 PROPOSITION *The function $g(x) = 3x/10$ has no continuous realizer f.*

PROOF (Brouwer 1920) If f were a realizer, the first digit of $f(3^n 2\cdots)$ would have to be 0 and that of $f(3^n 4\cdots)$ would have to be 1. On the other hand, that of $f(3^\omega)$ can be either 0 or else 1, because $[\![10^\omega]\!] = [\![09^\omega]\!] = 0.1 = 3[\![3^\omega]\!]/10$. But, by continuity, it can be neither. □

The good news is that there are other realizations or representations of real numbers that overcome the problem, as already discovered by Brouwer. His solution was to work with the non-integral base 2/3 and still with digits 0 and 1 — Turing [138] adopted this solution.

Here we consider an equivalent, well known solution which is perhaps more intuitive (see e.g. [144], and the introduction of [44] for some history). We keep the base 2 but allow negative digits:

$$\begin{array}{rcl} [\![-]\!]\colon 3^\omega & \twoheadrightarrow & [-1,1] \\ s & \mapsto & \sum_{i\in\omega} s_i 2^{-i-1} \end{array} \qquad \text{where } 3 = \{-1,0,1\}.$$

We refer to the members of the source of the quotient as *realizers* of the members of the target. With this terminology, each of the numbers -1 and 1 has exactly one realizer, the other dyadic numbers each have countably many realizers, and each non-dyadic number has uncountably many realizers. Intuitively, the problem identified in the above proof disappears, because when one is not so sure about two choices, either will do — if one makes a "mistake", it can be corrected at a later stage via the use of a negative digit.

More formally, the problem disappears as follows. The above realization function is a topological quotient map for the same reasons, and the same corollaries follow, with the bad news overcome:

6.13 Proposition *For every continuous $g\colon [-1,1] \to [-1,1]$ there is at least one continuous realizer $f\colon 3^\omega \to 3^\omega$.*

And, in fact, in general there are uncountably many realizers.

Proof For a full proof see e.g. [143]. First show that every continuous map $\phi\colon 3^\omega \to [-1,1]$ lifts to at least one continuous map $f\colon 3^\omega \to 3^\omega$ as in the diagram

$$\begin{tikzcd} 3^\omega \arrow[r, dashed, "f"] \arrow[dr, "\phi"'] & 3^\omega \arrow[d] \\ & {[-1,1].} \end{tikzcd}$$

(In categorical language, 3^ω (in the left top corner) is projective over the down quotient.) To conclude, apply this to the map $\phi(s) = g([\![s]\!])$. $\square$

The space 3^ω is homeomorphic to any countable product P of finite discrete spaces of cardinality at least 2 (in fact computationally so if the function that sends a natural number n to the size of the nth factor of the product is computable). Hence the (projectivity) assertion of the proof implies that for any quotient realization $\phi\colon P \twoheadrightarrow [-1,1]$ of the unit interval there is a continuous translation $f\colon P \to 3^\omega$ from P-notation to signed-digit notation:

$$\begin{tikzcd} P \arrow[r, dashed, "f"] \arrow[dr, "\phi"'] & 3^\omega \arrow[d] \\ & {[-1,1].} \end{tikzcd}$$

In this sense, signed-digit representation is characterized, up to continuous translation, as the *maximal* quotient realization using spaces of the form P. It can be shown that maximality still holds when one generalizes P to any subspace of the Baire space (the topological product $\mathbb{N}^\omega$, where $\mathbb{N}$ is the countable discrete space). Such a quotient realization of a space is called an *admissible representation* in Weihrauch's school of computability [142].

The above proposition holds with "continuous" replaced by "computable", for any of the many equivalent definitions of the notion of computability for functions over the reals that can be found in the literature:

> *A function $g\colon [-1,1] \to [-1,1]$ is computable iff it has at least one computable realizer $f\colon 3^\omega \to 3^\omega$.*

Thus, this can be taken as a formulation of the notion, assuming that the notion of computability over 3^ω is understood.

6.4 The interval domain

Very briefly, we consider the analogue of the situation

$$\begin{tikzcd} 2^\omega \arrow[r] \arrow[d, hook] & 2^\omega \arrow[d, hook] \\ 2^\infty \arrow[r] & 2^\infty \end{tikzcd}$$

with the Cantor space 2^ω and the "partialized" Cantor space 2^∞ replaced by the unit interval $[-1,1]$ and the "partialized" unit interval $I[-1,1]$:

$$I[-1,1] \quad = \quad \text{closed subintervals of } [-1,1] \text{ under the Scott topology of the reverse-inclusion order on intervals.}$$

We have a topological embedding

$$\begin{array}{rcl} [-1,1] & \hookrightarrow & I[-1,1] \\ x & \mapsto & [x,x] = \{x\}. \end{array}$$

Notice that this is an embedding onto the maximal elements of the interval domain. Thus, what this says is that the relative Scott topology on the maximal elements is (homeomorphic to) the usual Hausdorff topology on the closed interval $[-1,1]$. That is, yet again a computational topology induces a familiar topology. Hence in a situation

$$\begin{tikzcd} {[-1,1]} \arrow[r, "f"] \arrow[d, hook] & {[-1,1]} \arrow[d, hook] \\ I[-1,1] \arrow[r, "g"'] & I[-1,1], \end{tikzcd}$$

if g is continuous then so is f.

Moreover, for any continuous f there is at least one continuous g s.t. the above diagram commutes. As for Proposition 6.8, we sketch two proofs, referring the reader to Chapter 12 of Part II for domain-theoretic details.

First proof. Take $g(x)$ to be $\{f(r) \mid r \in x\}$. Because a subset of the unit interval is a closed interval if and only if it is compact and connected, and because continuous maps take compact sets to compact sets and connected sets to connected sets, $g(x)$ is a closed interval if x is, and hence g is well-defined, and it is clearly an extension of f is the sense of the above diagram. Moreover, direct-image formation is easily seen to preserve filtered intersections (i.e. directed joins in the interval domain), and hence g is Scott continuous.

Second proof. The space $I[-1,1]$, being a continuous Scott domain under the Scott topology, is densely injective and the embedding $[-1,1] \hookrightarrow I[-1,1]$ is dense.

Computability via the interval domain (using its standard effective presentation that enumerates rational intervals [120]) coincides with computability via signed-digit realizers, at least as far as second-order types are concerned (where the ground type of real numbers is taken to have order zero): What happens at third-order types and beyond is an open question, which in turn relies on an open problem in topology [10, 98].

6.5 Notes

Most of the material of this chapter is folklore, and some references have been given above. Regarding the interval domain, see Edalat's work [32] or e.g. [43].

Part II

Topology of classical spaces

Contents and organization

The foci of this mathematical part are Chapters 7 and 9, which parallel the central Chapter 3 of Part I from the point of view of classical topology. We first develop synthetic formulations of classical topological notions in a series of lemmas in Chapter 7. For the proof of the lemma that takes care of the notion of compactness, we need to pause to develop some material on function spaces, which is the topic of Chapter 8. This chapter also introduces the λ-calculus, which is the main tool in the synthetic development of topology.

As discussed in Chapter 8, it is sometimes possible to topologize the set of continuous maps from a space X to a space Y so that a function space Y^X that obeys the laws of exponentiation is obtained. The synthetic formulation of the notion of compactness makes use of the case in which Y is the Sierpinski space. As a result, the synthetic proof of e.g. the fact that a product of two compact spaces is again compact, provided in Chapter 9, has the extraneous assumption that the two spaces can serve as exponents for the Sierpinski space, which is not always the case, as explained in Chapter 8. The purpose of Chapter 10 is to show how one can easily circumvent this obstacle by considering generalized topological spaces that act as "imaginary" exponentials, very much like the imaginary number i acts as an exponential $(-1)^{\frac{1}{2}}$ of the two real numbers -1 and $1/2$ outside the real-number system.

Chapter 11 formulates representation theorems for compact and closed sets as universal and existential functionals, which are analogous to the Riesz representation theorem for measures as linear functionals.

Chapter 7

Synthetic formulation of classical topological notions

In this chapter we formulate some basic topological concepts as continuity notions with the aid of the *Sierpinski space*. This is the space $\mathbb{S}$ that has two points $\top$ (true) and $\perp$ (false), and three open sets $\emptyset, \{\top\}$ and $\{\perp, \top\}$. Equivalently, $\top$ is open but not closed, and $\perp$ is closed but not open.

7.1 Open subspaces

The following well known (and easy) lemma was the implicit reason for defining open subsets of data types in the way we did in Chapters 3 and 4.

7.1 Lemma *A subset* U *of a topological space* X *is* open *iff its* characteristic function

$$\begin{aligned} \chi_U : X &\to \mathbb{S} \\ x &\mapsto [\![x \in U]\!] = \begin{cases} \top & \text{if } x \in U, \\ \perp & \text{if } x \notin U, \end{cases} \end{aligned}$$

is continuous.

What is perhaps not so well known is that, like the notion of open subspace, those of Hausdorff, discrete, and compact space can also be reduced to continuity of certain maps involving the Sierpinski space.

7.2 Hausdorff spaces

A space is Hausdorff if any two distinct points can be separated by disjoint neighbourhoods, and one quickly learns that this is equivalent to saying that its diagonal is closed in the product topology. But the diagonal is closed iff its complement is open. This proves:

7.2 Lemma *A space X is Hausdorff iff its* apartness map

$$\begin{array}{rcl}(\neq)\colon X \times X & \to & \mathbb{S} \\ (x,y) & \mapsto & [\![x \neq y]\!]\end{array}$$

is continuous.

7.3 Discrete spaces

A space is discrete if every singleton, and hence every set of points, is open, but it is probably not so well known that this is equivalent to saying that its diagonal is open, which is an easy exercise:

7.3 Lemma *A space X is discrete iff its* equality map

$$\begin{array}{rcl}(=)\colon X \times X & \to & \mathbb{S} \\ (x,y) & \mapsto & [\![x = y]\!]\end{array}$$

is continuous.

7.4 Compact subspaces

In the next chapter we shall see how to topologize the set of continuous maps from a topological space X to a topological space Y, obtaining a natural function space $(X \to Y)$. The following remarkable fact is a reformulation of a well known property of function-space topologies.

7.4 Lemma *A subset Q of a topological space X is* compact *iff its universal-quantification functional*

$$\begin{array}{rccl}\forall_Q\colon & (X \to \mathbb{S}) & \to & \mathbb{S} \\ & p & \mapsto & [\![\forall x \in Q. p(x) = \top]\!]\end{array}$$

is continuous.

Proof Provided in Chapter 8. □

7.5 A classically invisible notion

We have seen in e.g. Proposition 3.13 that the existential-quantification functional may fail to be computable. But, in classical topology,

7.5 Lemma *For any $F \subseteq X$, the existential quantification functional*

$$\begin{array}{rccl} \exists_F : & (X \to \mathbb{S}) & \to & \mathbb{S} \\ & p & \mapsto & [\![\exists x \in F.p(x) = \top]\!] \end{array}$$

is always continuous.

Proof Provided in Chapter 8. □

We shall apply the above five lemmas in Chapter 9 to easily develop basic topology and extract constructive content from the theorems, performing the task of the computational Chapter 3 from the point of view of classical topology. In order to carry out this programme, it is necessary to pause to define and study the natural function space $(X \to Y)$ that occurs in the formulation of the last two lemmas.

7.6 Notes

For more notes about the material developed in this chapter, in particular the relationship to Taylor's abstract stone duality, see Section 3.14.

A logical presentation of the material of these notes would assume familiarity with classical topology, as this chapter does, and would have this chapter as the starting point. However, because the synthetic formulations presented in this chapter are appealing and stand on their own, they can be taken as a starting point for the synthetic topology of data types, as we have done in Chapter 3. Moreover, in that computational context, the λ-calculus is a familiar tool, which, in the form of a programming language, can be used in a natural way to prove topological theorems in a transparent way and obtain interesting, unexpected computational conclusions, as we have also done in Chapter 3.

In the classical formulation of topology, the notion of open set is taken as primitive, in the sense that all other topological notions are reduced to it. In the synthetic formulation developed in this chapter, the primitive notions are those of Sierpinski space of truth values and of continuity of maps. The fruitfulness of this change of perspective is illustrated in Chapters 3 and 4 of Part I, where the synthetic notion of continuity naturally varies in interesting ways. The reason this works is the striking fact that the

Sierpinski space has a direct computational interpretation as a space of results of observations or semidecisions. The asymmetry of the topology of the Sierpinski space precisely matches the asymmetry of the computational notion of semidecision.

In the same way as the present chapter gives input to Chapter 3, by providing synthetic formulations of classical topological notions, that chapter in turn gives input to Chapter 9, which develops the core of classical topology via the λ-calculus. Thus, the interaction between topology and computation goes both ways. In this part, the highlight is the input of computational ideas into topology. However, this has to be taken with a pinch of salt: In the computational setting, the function spaces required for the synthetic formulation of the notion of compactness exist by fiat, but in the topological setting we have to work hard to reach them (Chapters 8 and 10).

In Part III, where the highlight is again the input of topological (and also order-theoretical) ideas into the theory of computation, we unify the developments of this and the previous parts, where once more non-trivial computational conclusions are derived from topological theorems.

Chapter 8

Function spaces in classical topology

The previous chapter reduces some fundamental topological concepts to the notion of continuity with the aid of the Sierpinski space. For the notion of compactness (Lemma 7.4), we invoked function spaces, which we now develop. In Sections 8.1 and 8.2 we discuss exponentiation of spaces and its laws, and in Section 8.3 we introduce λ-notation. These tools are applied in Chapter 9 to easily develop basic topology parallelling the development of Chapter 3.12.

For an expository account of function spaces in topology with full proofs, together with credits and references to original sources, see [50]. In this chapter we summarize the development of that paper to the extent that is needed for our purposes. After reading Sections 8.1–8.3, it is possible to proceed directly to Chapter 9 provided Lemmas 7.4 and 7.5, which are proved in Section 8.4, are taken on faith. Section 8.5 formulates some characterizations of exponentiable spaces and exponential topologies, which are partially proved in Chapter 9.

8.1 Exponentials and natural function spaces

For topological spaces X and Y, we denote by $\mathrm{C}(X,Y)$ the *set* of continuous maps from X to Y. The *transpose* $\overline{g}\colon A \to \mathrm{C}(X,Y)$ of a continuous map $g\colon A \times X \to Y$ is defined by

$$\overline{g}(a) = g_a, \qquad \text{where } g_a \in \mathrm{C}(X,Y) \text{ is given by } g_a(x) = g(a,x).$$

More concisely, we write the definition of the transpose as

$$\overline{g}(a) = (x \mapsto g(a,x)) \qquad \text{or} \qquad \overline{g}(a)(x) = g(a,x).$$

A topology on the set $\mathrm{C}(X,Y)$ is called *exponential* if continuity of a function $g\colon A \times X \to Y$ is equivalent to that of its transpose $\overline{g}\colon A \to \mathrm{C}(X,Y)$. As we shall see soon, there is at most one such topology. If it exists, the set $\mathrm{C}(X,Y)$ endowed with this topology is referred to as an *exponential* and is denoted by

$$Y^X.$$

For example, if the exponential exists and we take A to be the closed unit interval $I = [0,1]$, then a homotopy $h\colon I \times X \to Y$ of continuous functions $f, g\colon X \to Y$ is essentially the same thing as a path $\overline{h}\colon I \to Y^X$ from f to g in the function space Y^X.

8.1 Remark For readers who know the general definition of an exponential Y^X of two objects X and Y of a category (briefly: the contravariant set-valued functor $\hom(- \times X, Y)$ is representable by Y^X), which we are not assuming, we emphasize that, because our category is well pointed, the categorical notion is equivalent to that defined below in our special case.

Unfortunately, there isn't in general an exponential topology (Theorem 8.24) and hence we aren't always entitled to write Y^X. In other words, the category of topological spaces fails to be *cartesian closed*. But there is always a canonical candidate for the exponential topology, which will crucially come to our rescue (Lemma 10.1 and Corollary 10.2).

8.2 Lemma (Natural topology) *There is a largest topology on* $\mathrm{C}(X,Y)$ *such that, for all spaces* A*, continuity of a function* $g\colon A \times X \to Y$ *implies that of its transpose* $\overline{g}\colon A \to \mathrm{C}(X,Y)$*, known as the* natural topology.

Proof Declare a set $N \subseteq \mathrm{C}(X,Y)$ to be open if and only if $\overline{g}^{-1}(N)$ is open for every continuous map $g\colon A \times X \to Y$. These sets are easily seen to form a topology, which, by construction, satisfies the required property. □

8.3 Remark One may wonder whether it would perhaps be sensible to take the smallest topology for which the converse holds. However, this topology doesn't always exist — see Remark 8.19 below.

The set $\mathrm{C}(X,Y)$ endowed with the natural topology is denoted by

$$(X \to Y)$$

and referred to as the *natural function space*. Lemmas 8.4 and 8.5 below are elaborated in Section 8.4.

8.4 Lemma *If the exponential Y^X exists then it coincides with the natural function space $(X \to Y)$.*

Lemmas 7.4 and 7.5 (proved in Section 8.4) hold whether or not the topology of the natural function space $(X \to \mathbb{S})$ is exponential. But, in order to be able to apply them in Chapter 9, we need it to be exponential — or else use the technology developed in Chapter 10.

8.5 Lemma *If the exponential $\mathbb{S}^X$ exists then so does the exponential Y^X for every topological space Y.*

8.2 Exponential laws

The above facts about exponentials are particular to the category of topological spaces. The following three lemmas easily follow from the general categorical definition or the equivalent one given above. The (external) definition of an exponential Y^X says that transposition is a bijection from continuous maps $A \times X \to Y$ to continuous maps $A \to Y^X$.

8.6 Lemma (Internal exponential law) *Let A, X and Y be topological spaces and assume that the exponential Y^X exists. If either of the exponentials $Y^{A\times X}$ and $(Y^X)^A$ exists then so does the other, and they are homeomorphic via transposition:*

$$\begin{array}{ccc} Y^{A\times X} & \cong & (Y^X)^A \\ g & \mapsto & \overline{g}. \end{array}$$

This and the following two lemmas play an important role in the applications developed in Chapter 9.

8.7 Lemma *If the exponential Y^X exists then the evaluation map*

$$\begin{array}{rccc} \varepsilon_{X,Y}\colon & Y^X \times X & \to & Y \\ & (f,x) & \mapsto & f(x) \end{array}$$

is continuous.

Proof It has the identity map $Y^X \to Y^X$ as its transpose. $\square$

8.8 LEMMA *If $f\colon Y \to Z$ and $h\colon W \to X$ are continuous maps of topological spaces then the functionals*

$$\begin{array}{rcl} f^X\colon Y^X & \to & Z^X \\ g & \mapsto & f \circ g \end{array} \qquad\qquad \begin{array}{rcl} Y^h\colon Y^X & \to & Y^W \\ g & \mapsto & g \circ h \end{array}$$

are continuous, provided the involved exponentials exist.

8.3 The restricted, simply typed λ-calculus

In our context, the λ-calculus is a labour-saving device for manipulating the exponential laws discussed in the two previous sections. However, because in Chapter 9 we include the direct manipulations of function spaces corresponding to the λ-calculations that we provide, readers may safely take a casual look at the development of this section.

The expression $x+y$ of the real variables x and y can be regarded either as a real number, as function of x, as a function of y, or as a function of both. In order to make the distinction explicit, one can write:

$$x+y, \qquad x \mapsto x+y, \qquad y \mapsto x+y, \qquad (x,y) \mapsto x+y.$$

A fifth way of interpreting the expression $x+y$ is as a function that, for each given x, produces the function $y \mapsto x+y$. In this case one can write

$$x \mapsto (y \mapsto x+y).$$

In the λ-calculus, one uses λ-notation rather than the mathematically more familiar $\mapsto$-notation. For instance, some of the above examples are written

$$x+y, \qquad \lambda x.x+y, \qquad \lambda y.x+y, \qquad \lambda x.\lambda y.x+y.$$

(This is awkward when we use the λ-calculus to do e.g. linear algebra, measure theory and integration, where λ's traditionally play the role of scalars. See the example towards the end of this section.)

Often the λ-calculus is taken as a symbol-pushing, formal system, or even programming language, without any *a priori* mathematical interpretation. Here we use the *restricted* simply typed λ-calculus as a device for automatically constructing continuous maps out of given ones (generalizing the fact that compositions of continuous maps are automatically continuous), and we deliberately omit syntactic details that are irrelevant for our present purposes (but that are crucial for some calculational aspects regarding the development of Chapter 2 and hence 3). The restriction, discussed below, comes from the fact that not all exponentials exist in the world of topological spaces.

In the above examples, some variables are *free* and others are *bound*. For instance, in $x + y$ both variables are free, in $\lambda x.x + y$ the variable x is bound and the variable y is free, and in $\lambda x.\lambda y.x + y$ both variables are bound. Notice that bound variables can be safely renamed provided we are consistent. For example, there is no difference between $\lambda x.\lambda y.x + y$ and $\lambda y.\lambda z.y + z$ other than the accidental choice of names of variables.

The above examples have different *types*, which are topological spaces where their values live. For example, $x + 1$ has type $\mathbb{R}$ and $\lambda x.x + 1$ has type $\mathbb{R}^{\mathbb{R}}$. Thus, we have to know that the exponential exists before being able to write $\lambda x.x + 1$. This is the restriction alluded to above.

The λ-*polynomials* (also known as λ-*expressions* or λ-*terms*) are inductively defined, together with their free variables and types, as follows:

(λ_0) Every variable x that ranges over a space X is a polynomial of type X, with just one free variable x.

(λ_1) If the exponential Y^X exists, x is a variable that ranges over the space X and t is a polynomial of type Y, then $\lambda x.t$ is a polynomial of type Y^X, with the same free variables as t except x.

Notice that we don't require that x occurs as a free variable of t — consider a constant function.

(λ_2) If $f : X_1 \times \cdots \times X_n \to Y$ is a continuous map and $t_1, \ldots, t_n$ are polynomials of types $X_1, \ldots, X_n$, then $f(t_1, \ldots, t_n)$ is a polynomial of type Y, with free variables those of $t_1, \ldots, t_n$.

This clause includes the possibility $n = 0$, in which case the product $X_1 \times \cdots \times X_n$ is the one-point space and hence f picks a point of Y. To avoid the detour via the one-point space, we may safely agree that if y_0 is a point of Y then y_0 is a polynomial of type Y with no free variables.

Choosing $Y = X_1 \times \cdots \times X_n$ and f the as identity map in the last clause, we see that if $t_1, \ldots, t_n$ are polynomials of types $X_1, \ldots, X_n$ then $(t_1, \ldots, t_n)$ is a polynomial of type $X_1 \times \cdots \times X_n$. Using the same clause again, we conclude that if the exponential Y^X exists and if t and t' are polynomials of type Y^X and X (respectively, of course), then $\varepsilon(t, t')$ is a polynomial of type Y, where $\varepsilon\colon Y^X \times X \to Y$ is the evaluation map. This polynomial $\varepsilon(t, t')$ is abbreviated as $t(t')$. For future reference, we summarize these two derived clauses:

(λ_3) If $t_1, \ldots, t_n$ are polynomials of types $X_1, \ldots, X_n$ then $(t_1, \ldots, t_n)$ is a polynomial of type $X_1 \times \cdots \times X_n$ with free variables those of $t_1, \ldots, t_n$.

(λ_4) If the exponential Y^X exists and t and t' are polynomials of types Y^X and X, then $t(t')$ is a polynomial of type Y with free variables those of t and t'.

A *polynomial function*, or λ-*definable function*, is one that is obtained by evaluating a polynomial. Such a function will be continuous by construction. In order to evaluate a polynomial, we have to assign values to its free variables. More precisely and more generally, if a polynomial t of type X has free variables that are among those in the list of variables $a_1, \ldots, a_k$ (without repetitions) of type $A_1, \ldots, A_k$ (possibly with repetitions), then t defines a continuous map $A_1 \times \cdots \times A_k \to X$. We refer to such a list of variables and types for the polynomial t as a *context*, and we abbreviate the type information by writing $a_1, \ldots, a_k\colon A_1, \ldots, A_k$ for contexts and $t : X$ for polynomials.

The continuous maps *defined* by polynomials are inductively constructed as follows:

(λ_0) A variable a_i in the context $a_1, \ldots, a_k\colon A_1, \ldots, A_k$ defines the projection

$$\pi_i\colon A_1 \times \cdots \times A_k \to A_i.$$

(λ_1) If the exponential Y^X exists and the polynomial $t\colon Y$ in the context $a_1, \ldots, a_k, x\colon A_1, \ldots, A_k, X$ defines the continuous map

$$g\colon A_1 \times \cdots \times A_k \times X \to Y,$$

then the polynomial $\lambda x.t\colon Y^X$ in the context $a_1, \ldots, a_k$ defines its exponential transpose

$$\overline{g}\colon A_1 \times \cdots \times A_k \to Y^X.$$

(Notice that the variable x is not among $a_1, \ldots, a_n$ because, by definition of context, $a_1, \ldots, a_n, x$ doesn't contain repetitions.)

(λ_2) If $f\colon X_1 \times \cdots \times X_n \to Y$ is a continuous map and the polynomials $t_i\colon X_i$ in the context $a_1, \dots, a_k\colon A_1, \dots, A_k$ define continuous maps

$$g_i\colon A_1 \times \cdots \times A_k \to X_i,$$

then the polynomial $f(t_1, \dots, t_n)\colon Y$ defines the composite

$$f \circ (g_1, \dots, g_n)\colon A_1 \times \cdots \times A_k \to Y.$$

Thus, by construction, λ-definable functions are continuous. Because they are constructed from continuous functions by applications of the continuous-maps clause (λ_2) with the aid of the variables clause (λ_0) and the lambda clause (λ_1), we can say, more memorably:

> *Functions that are λ-definable from continuous maps are themselves continuous.*

It is now clear that definitions using λ-notation amount to sequences of transpositions and compositions of continuous maps. Although it may not be immediately apparent, such calculations occur often in mathematics, at least implicitly. For example, a particular case of *Fubini's rule* for integration says that, in order to integrate a continuous map of two variables, we can integrate over one variable and then over the other in an iterated fashion:

$$\int_{X \times Y} f = \int_X \left(\int_Y f(x,y)dy \right) dx.$$

If we regard the integration signs as standing for continuous functionals

$$\int_{X \times Y}\colon \mathbb{R}^{X \times Y} \to \mathbb{R}, \qquad \int_X\colon \mathbb{R}^X \to \mathbb{R}, \qquad \int_Y\colon \mathbb{R}^Y \to \mathbb{R},$$

then the right-hand side of the above equation can be equivalently written

$$\int_X \lambda x. \int_Y \lambda y. f(x,y).$$

Unraveling the definitions, and using the notation of the previous section, we see that this polynomial (with the free functional variable f of type $\mathbb{R}^{X \times Y}$) defines the composite

$$\mathbb{R}^{X \times Y} \xrightarrow{\cong} (\mathbb{R}^Y)^X \xrightarrow{(\int_Y)^X} \mathbb{R}^X \xrightarrow{\int_X} \mathbb{R}.$$

Thus, what the equation says is that this is the same as $\int_{X \times Y}\colon \mathbb{R}^{X \times Y} \to \mathbb{R}$.

For us, the point of using the λ-calculus is that we automatically conclude that the functional defined by such a polynomial is continuous, because it is λ-defined from continuous maps (in this case the functionals $\int_X \colon \mathbb{R}^X \to \mathbb{R}$ and $\int_Y \colon \mathbb{R}^X \to \mathbb{R}$). Moreover, in practice, there is no need to unravel the continuous map defined by the polynomial in order to know that it is continuous, because this is so by construction, as we have seen. In our applications, we consider functionals of the same type, typically with $\mathbb{R}$ replaced by the Sierpinski space $\mathbb{S}$, e.g. the universal and existential quantification functionals $\forall \colon \mathbb{S}^X \to \mathbb{S}$ and $\exists \colon \mathbb{S}^X \to \mathbb{S}$ of Lemmas 7.4 and 7.5 (see also Chapter 11).

8.9 Remark It is possible to remove the assumption of existence of the exponential Y^X in clause (λ_2) of the definition of polynomials by replacing Y^X by the natural function space $(X \to Y)$, which always exists. The construction of continuous maps defined by polynomials still works with this modification, because, by definition, the natural function space allows transposition in the required direction. Given that we know that if the exponential exists then it coincides with the natural function space, this is a sensible thing to do. But notice that the derived clause (λ_4) still needs the proviso that the exponential exists. The reason is that the natural topology is exponential if and only if it makes the evaluation map continuous, as we shall see in the next section. By virtue of the results of Chapter 10, which allow us to work with exponentials constructed outside the world of topological spaces, we don't take the troublesome route of working with this generalization. However, in some interesting examples, working with it allows us to sidestep the machinery developed in Chapter 10 — see Remark 9.2.

8.10 Question Is there an analogue of the natural function space in the category of locales? Is the natural topology characterized by a universal property that can be formulated in arbitrary categories with finite products?

At this point, as discussed above, if Lemmas 7.4 and 7.5 are taken on faith, it is possible proceed directly to Chapter 9.

8.4 Exponentiable spaces

In order to discuss existence of exponentials Y^X, we introduce the following terminology. A topology on the set $\mathrm{C}(X, Y)$ is called

1. *splitting* if continuity of $g\colon A \times X \to Y$ implies that of $\overline{g}\colon A \to \mathrm{C}(X, Y)$,
2. *conjoining* if continuity of $\overline{g}\colon A \to \mathrm{C}(X, Y)$ implies that of $g\colon A \times X \to Y$.

With this terminology, the topology is exponential if it is both splitting and conjoining.

8.11 Lemma *A topology on* $\mathrm{C}(X, Y)$ *is conjoining if and only if it makes the* evaluation map $\varepsilon_{X,Y}\colon \mathrm{C}(X, Y) \times X \to Y$ *into a continuous function.*

Using this, one easily proves the following lemma, which can be read as saying that the splitting and conjoining topologies form a sort of *Dedekind cut*. Recall that a topology T on a given set is *coarser* than another topology T' on the same set if $T \subseteq T'$. In this case one also says that the topology T' is *finer* than T.

8.12 Lemma

1. *The indiscrete topology is splitting and the discrete is conjoining.*
2. *Any topology coarser than a splitting topology is also splitting.*
3. *Any topology finer than a conjoining topology is also conjoining.*
4. *Any splitting topology is coarser than any conjoining topology.*

8.13 Corollary *There is at most one exponential topology; when it exists, it is the coarsest conjoining topology, or, equivalently, the finest splitting topology.*

Because the natural topology is the finest splitting topology by construction, the above establishes Lemma 8.4. In general, however, there isn't a coarsest conjoining topology unless the natural topology is conjoining (see Remark 8.19).

8.14 Definition (Exponentiable space) A space X is called *exponentiable* if the set $\mathrm{C}(X, Y)$ admits an exponential topology for every space Y.

We have just concluded that a space X is exponentiable if and only if the natural function space $(X \to Y)$ is exponential for every space Y. Our next goal is to formulate an intrinsic characterization of exponentiable spaces and of exponential topology.

Firstly, one can avoid quantification over all spaces Y in the definition of exponentiability of X: As stated in Lemma 8.5, it turns out that a topological space X is exponentiable if and only if the single exponential $\mathbb{S}^X$ exists. Moreover, in this case, the topology of Y^X, for any space Y, is determined by the topologies of $\mathbb{S}^X$ and Y as follows. For any topological space, let

$$\mathcal{O}\,X$$

denote its set of open sets.

8.15 Definition (Induced function-space topology) The topology on the set $\mathrm{C}(X, Y)$ *induced* by a topology T on the set $\mathcal{O}\,X$ is that generated by the subbasic open sets

$$N(H, V) = \{f \in \mathrm{C}(X, Y) \mid f^{-1}(V) \in H\},$$

where H ranges over T and V ranges over $\mathcal{O}\,Y$.

We have seen that there is a bijection from the lattice of open sets $\mathcal{O}\,X$ of X to the set $\mathrm{C}(X, \mathbb{S})$ that sends an open set to its characteristic map. We transfer names of properties of topologies on $\mathrm{C}(X, \mathbb{S})$ to $\mathcal{O}\,X$ via the bijection. So, for example, a topology on $\mathcal{O}\,X$ is called exponential if the corresponding topology on $\mathrm{C}(X, \mathbb{S})$ is exponential.

8.16 Lemma *A topology on $\mathcal{O}\,X$ is splitting (resp. conjoining) iff it induces a splitting (resp. conjoining) topology on $\mathrm{C}(X, Y)$ for every space Y.*

8.17 Corollary *A space X is exponentiable if and only if $\mathcal{O}\,X$ has an exponential topology. In this case, the exponential topology of $\mathrm{C}(X, Y)$ is that induced by the exponential topology of $\mathcal{O}\,X$.*

A set $H \subseteq \mathcal{O}\,X$ is called *Alexandroff open* if the conditions $U \in H$ and $U \subseteq U' \in \mathcal{O}\,X$ together imply that $U' \in H$, and it is called *Scott open* if in addition every open cover of a member of H has a finite subcover of a member of H. (Because $\mathcal{O}\,X$ is a complete lattice, the latter is equivalent to saying that every directed open cover of a member of H intersects H.) Thus, for example, for any subset Q of X, the Alexandroff open set $\{U \in \mathcal{O}\,X \mid Q \subseteq U\}$ is Scott open if and only if Q is compact. We observe that this example is the only place where the open-cover definition of compactness occurs in Part II. All other uses of compactness are reduced to it, via the proof of Lemma 7.4 that we are about to give.

8.18 Lemma (Naturality of the Scott topology.)

1. *The Scott topology of $\mathcal{O}\,X$ is always splitting.*

2. *The Scott topology of $\mathcal{O}\,X$ is an intersection of conjoining topologies.*

Hence the Scott topology is the natural topology.

8.19 Remark By the second part of this lemma, it follows that there isn't in general a coarsest conjoining topology, unless the natural topology is conjoining, in which case the coarsest conjoining topology coincides with the finest splitting topology, i.e. the space is exponentiable — cf. Remark 8.3.

We can now fill two gaps.

Proof of Lemma 7.4. By the corollary, the natural function space $(X \to \mathbb{S})$ is homeomorphic to the set $\mathcal{O}\,X$ under the Scott topology via the bijection $\mathcal{O}\,X \cong \mathrm{C}(X,\mathbb{S})$, and the universal-quantification functional can be regarded as a map $\forall_Q \colon \mathcal{O}\,X \to \mathbb{S}$ with $\forall_Q(U) = \top$ iff $\forall x \in Q.\chi_U(x) = \top$. But $\forall x \in Q.\chi_U(x) = \top$ iff $\forall x \in Q.x \in U$ iff $Q \subseteq U$. Hence $\forall_Q^{-1}(\top) = \{U \in \mathcal{O}\,X \mid Q \subseteq U\}$, which, as we have observed above, is Scott open if and only if Q is compact. □

Proof of Lemma 7.5. Via the bijection $\mathrm{C}(X,\mathbb{S}) \cong \mathcal{O}\,X$, the existential-quantification functional can be regarded as a map $\exists_F \colon \mathcal{O}\,X \to \mathbb{S}$ with $\exists_F(U) = \top$ iff $F \cap U \neq \emptyset$. An easy verification shows that the set $\exists_F^{-1}(\top) = \{U \in \mathcal{O}\,X \mid F \cap U \neq \emptyset\}$ is Scott open. □

8.5 Characterization of exponentiable spaces

In order to summarize the results of Section 8.4, we name a special instance of the induced topology:

8.20 Definition (Isbell topology) The topology on $\mathrm{C}(X,Y)$ induced by the Scott topology of $\mathcal{O}\,X$ is known as the *Isbell topology*.

8.21 Theorem *A space is exponentiable if and only if the Scott topology of its lattice of open sets is conjoining. Moreover, for X exponentiable, the topology of an exponential Y^X is the Isbell topology.*

8.22 Lemma *The Scott topology of $\mathcal{O}\,X$ is conjoining if and only if $\mathcal{O}\,X$ is a continuous lattice in the sense of Dana Scott.*

Regarding continuous lattices, see Chapter 12 and, in connection with function spaces, Section 12.11 in particular. An equivalent topological formulation of the lattice-theoretic condition is that X be core-compact:

8.23 Definition (Core-compact space) A topological space X is called *core-compact* if every open neighbourhood V of a point x of X contains an open neighbourhood U of x such that every open cover of V has a finite subcover of U.

Hence every locally compact space is core-compact. Moreover, among Hausdorff spaces (and more generally sober spaces), core-compactness coincides with local compactness. As it is well known, a careful formulation of the notion of local compactness is needed in the absense of the Hausdorff separation axiom: We mean that every point has a base of compact (not necessarily open) neighbourhoods. The following is an immediate corollary.

8.24 Theorem *A topological space is exponentiable if and only if it is core-compact.*

We finish by remarking that if X is locally compact, then the topology of the exponential Y^X also coincides with the *compact-open topology*. This is generated by the subbasic Isbell open sets of the form $N(H, V)$ with $H = \{U \in \mathcal{O}\, X \mid Q \subseteq U\}$ for $Q \subseteq X$ compact. These are precisely the sets of the form $\{f \in Y^X \mid f(Q) \subseteq V\}$ which occur in the usual formulation of the compact-open topology (cf. Propositions 3.20 and 9.11).

8.6 Notes

For the long and interesting history of the subject of function spaces, see Isbell [68]. The only thing to be added is that the terminology *natural topology* used in this chapter is taken from an unpublished manuscript by Eilenberg [36], which was kindly supplied by Fred Linton to the author — but of course the concept was known long before that manuscript.

The interpretation of the simply-typed λ-calculus in cartesian closed categories is a familiar theme in categorical logic — see e.g. [23, 72, 84, 90]. Here we have rehearsed this in the particular case of the category of continuous maps of topological spaces, taking care of the (rather annoying) fact that it is not cartesian closed, i.e. fails to have all exponentials.

Chapter 9

Classical topology via the λ-calculus

We have seen in Chapter 7 that some topological notions, such as those of open and closed subspace, and those of Hausdorff, discrete and compact space, can be expressed as continuity of certain maps involving the Sierpinski space. Using the function-space machinery of the previous chapter, we can combine these maps in order to produce new continuous maps and hence easily prove known propositions about topological spaces. Equivalently, we can combine them using the λ-calculus. In any case, the point is that one automatically gets continuous functions out of given continuous functions. For the benefit of readers who are not acquainted with the λ-calculus, or who feel in shaky grounds given the fact that not all exponentials exist in the world of topological spaces, we include both the proofs via the λ-calculus and those via direct manipulation of function spaces.

To be able to take full advantage of the function-space machinery or the λ-calculus, we would need the world of topological spaces to have exponentials Y^X for all topological spaces X and Y, that is, to form a *cartesian closed* category, which it doesn't (see Chapter 8 and Remark 9.9). One can take the further step of formally adding the missing exponentials when one needs them, very much like one adds the missing exponential $i = (-1)^{\frac{1}{2}}$ to the reals obtaining the complex numbers, and we indeed proceed in this way in Chapter 10. But, to begin with, we content ourselves with working within the world of "real" topological spaces, explicitly assuming existence of exponentials within the world when necessary, and hence the propositions are not formulated in the full generality they are known. However, in order to be able to reuse both the formulations and the proofs provided here to regain full generality, we flag such extraneous existence assumptions as *preliminary*. Notice that there are some occurrences of existence assumptions to which we don't attach the label: This is the case when function spaces occur in the formulation of a proposition. In the flagged cases, the exponentials are needed only in the proofs.

9.1 PROPOSITION *If X is Hausdorff and $Q \subseteq X$ is compact, then Q is closed.*

Preliminary assumption. The exponential $\mathbb{S}^X$ exists.

PROOF By Lemma 7.1 and Section 8.3, it is enough to λ-define the complement of the characteristic function of Q from continuous maps. Because $x \notin Q \iff \forall y \in Q. x \neq y$, we conclude that $\chi_{X\setminus Q}(x) = \forall_Q(\lambda y. x \neq y)$. The result then follows from the fact that $\forall_Q \colon \mathbb{S}^X \to \mathbb{S}$ and $(\neq) \colon X \times X \to \mathbb{S}$ are continuous by the assumptions that Q is compact and X is Hausdorff, using the synthetic formulations of the topological notions given in Lemmas 7.4 and 7.2. $\square$

The continuous map defined by the λ-expression of the above proof is obtained as follows. Using the exponential law, we get the continuous map $\overline{(\neq)} \colon X \to \mathbb{S}^X$ from the continuous map $(\neq) \colon X \times X \to \mathbb{S}$, and, composing with the continuous map $\forall_Q \colon \mathbb{S}^X \to \mathbb{S}$, we get the continuous map

$$X \xrightarrow{\overline{(\neq)}} \mathbb{S}^X \xrightarrow{\forall_Q} \mathbb{S},$$

which gives the characteristic function of the complement of Q.

A constructive reading of the proposition is that if we can tell points of X apart and we can quantify over Q, then we can semidecide the complement of Q. Algorithms for the first two tasks give an algorithm for the third — see Chapter 3. This is what the λ-expression amounts to in a computational setting. Thus, both the *formulation* of the classical proposition and its *proof* are seen to have computational content, and synthetic proofs are programs in a literal sense.

9.2 REMARK By Remark 8.9, we see that in this example we can interpret the λ-expression using the natural function space $(X \to \mathbb{S})$ and hence the preliminary assumption that $\mathbb{S}^X$ exists is not really necessary. The same applies to other examples in this chapter, but the exercise of discovering them (and of showing that the others don't qualify) is left to the interested reader.

9.3 PROPOSITION *If X is compact and $F \subseteq X$ is closed then F is compact.*

Preliminary assumption. The exponential $\mathbb{S}^X$ exists.

PROOF We λ-define $\forall_F : \mathbb{S}^X \to \mathbb{S}$ from continuous maps. Notice that $\forall x \in F.p(x)$ iff $\forall x \in X.x \in F \implies p(x)$ iff $\forall x \in X.x \notin F \vee p(x)$. Hence $\forall_F(p) = \forall_X(\lambda x.\chi_{X \setminus F}(x) \vee p(x))$, where $(- \vee -) : \mathbb{S} \times \mathbb{S} \to \mathbb{S}$ is the evident (continuous) disjunction map. $\square$

The λ-expression defines a continuous map as follows. Firstly, we have a composition of continuous maps

$$\mathbb{S}^X \times X \xrightarrow{\mathrm{id} \times \Delta} \mathbb{S}^X \times (X \times X) \xrightarrow{\cong} (\mathbb{S}^X \times X) \times X \xrightarrow{\varepsilon \times \chi_{X \setminus F}} \mathbb{S} \times \mathbb{S} \xrightarrow{\vee} \mathbb{S},$$

where id is the identity map of $\mathbb{S}^X$, the map Δ is the diagonal of X, the symbol $\cong$ denotes the evident homeomorphism, and ε is the evaluation map of the exponential $\mathbb{S}^X$. Transposing this composite we get a continuous map $\mathbb{S}^X \to \mathbb{S}^X$, which composed with $\forall_X : \mathbb{S}^X \to \mathbb{S}$ yields $\forall_F : \mathbb{S}^X \to \mathbb{S}$.

This illustrates the typical phenomenon that λ-expressions are easier to deal with than the corresponding sequences of compositions and transpositions. However, because our category doesn't have all exponentials, it is a good idea to explicitly write down the translations of our λ-expressions in order to be clear about exactly which exponentials are needed.

Generalizing the proof of the above proposition, we get that if F is a closed subspace of a space X and Q is a compact subspace of X, then $F \cap Q$ is compact, because $\forall x \in F \cap Q.p(x)$ iff $\forall x \in Q.x \notin F \vee p(x)$, with the same preliminary assumption.

9.4 PROPOSITION *If $f : X \to Y$ is continuous and $Q \subseteq X$ is compact then $f(Q)$ is compact.*

Preliminary assumption. The exponentials $\mathbb{S}^X$ and $\mathbb{S}^Y$ exist.

PROOF $\forall y \in f(Q).p(y)$ iff $\forall x \in Q.p(f(x))$. $\square$

Here the continuous map defined by the implicitly given λ-expression in the above proof is the composite

$$\mathbb{S}^Y \xrightarrow{\mathbb{S}^f} \mathbb{S}^X \xrightarrow{\forall_Q} \mathbb{S},$$

which gives the quantifier of $f(Q)$.

9.5 PROPOSITION *If X and Y are compact then so is $X \times Y$.*

Preliminary assumption. The exponentials $\mathbb{S}^X$ and $\mathbb{S}^Y$ exist.

PROOF $\forall z \in X \times Y.p(z)$ iff $\forall x \in X.\forall y \in Y.p(x, y)$. $\square$

Cf. Fubini's rule for integration over a product of two spaces. The continuous map defined by the implicitly given λ-expression is the composite

$$\mathbb{S}^{X\times Y} \xrightarrow{\cong} (\mathbb{S}^Y)^X \xrightarrow{(\forall_Y)^X} \mathbb{S}^X \xrightarrow{\forall_X} \mathbb{S}.$$

Notice that, under the assumption of existence of $\mathbb{S}^X$ and $\mathbb{S}^Y$ we get the existence of $(\mathbb{S}^Y)^X$ by Lemma 8.5 and hence that of $\mathbb{S}^{X\times Y}$ by Lemma 8.6, which also gives the homeomorphism.

9.6 Remark The above proof looks unbelievably easy compared to the classical proofs. However, the law of conservation of proofs is not violated: An interesting exercise reveals that one of the classical proofs is obtained from the above by a routine unwinding of the proofs of the various lemmas on which it relies. For example, if we start with the proofs given in [50] for exponentials, then an unwinding of above proof produces essentially the proof given in [129, Theorem 5.6.2]. One notices that the major work is performed by the proof of [50, Lemma 4.2] (formulated as Lemma 8.18(1) here). The other lemmas perform bookkeeping only. Indeed, the proof of [129, Theorem 5.6.2] looks as a nesting of two copies of the proof of [50, Lemma 4.2]. The pasting of the two copies is performed by the composition $(\mathbb{S}^Y)^X \xrightarrow{(\forall_Y)^X} \mathbb{S}^X \xrightarrow{\forall_X} \mathbb{S}$, and further composition with the homeomorphism $\mathbb{S}^{X\times Y} \cong (\mathbb{S}^Y)^X$ produces the nesting. A similar examination of the other proofs regarding compactness provided in this chapter reveals that the classical proofs include special instances of the proof of [50, Lemma 4.2]. Thus, that proof can be regarded as the "generic argument involving compactness": The λ-calculus machinery implicitly produces the familiar known instances. These remarks still apply when we eliminate the preliminary assumptions via the techniques of Chapter 10. The λ-calculus (or the function-space machinery) can thus be regarded as an "automatic bookkeeping device" for writing high-level versions of the classical proofs — and, in our case, also for extracting computational content from them.

9.7 Proposition *If Y is Hausdorff then so is the exponential Y^X if it exists.*

Preliminary assumption. The exponential $\mathbb{S}^X$ exists.

Proof $f \neq g$ iff $\exists x \in X. f(x) \neq g(x)$ and Lemma 7.5. □

The continuous map defined by the λ-expression implicitly given in the proof of the above proposition gives the apartness map of Y^X from that of Y as the composite

$$Y^X \times Y^X \xrightarrow{\cong} (Y \times Y)^X \xrightarrow{(\neq_Y)^X} \mathbb{S}^X \xrightarrow{\exists_X} \mathbb{S}.$$

The λ-proof of the above proposition suggests the following dual version. We start from the argument and then try to figure out what it proves: $f = g$ iff $\forall x \in X. f(x) = g(x)$. For this to λ-define a continuous map of the pair (f, g), we need continuously semidecidable equality on Y and continuous universal quantification over X. Thus, under the same preliminary assumption:

9.8 PROPOSITION *If X is compact and Y is discrete, then the exponential Y^X is discrete if it exists.*

Notice how the proofs of two seemingly unrelated propositions have the same shape:

$$Y^X \times Y^X \xrightarrow{\cong} (Y \times Y)^X \xrightarrow{(=_Y)^X} \mathbb{S}^X \xrightarrow{\forall_X} \mathbb{S}.$$

9.9 REMARK Because the Cantor space is compact Hausdorff, it is locally compact and hence the exponential 2^{2^ω} exists by Chapter 8 and is discrete by the above proposition. In fact, it is homeomorphic to the discrete space $\mathbb{N}$, because 2^ω has countably many clopens (sets which are both open and closed), and because the two-point discrete space 2 classifies clopen subspaces. (Cf. Proposition 3.23 and Chapters 13.2–13.3.) A routine verification shows that the exponential Y^X exists if X is discrete, and coincides with the topological product of X-many copies of Y. In particular, the Cantor space 2^ω is the same as the exponential $2^{\mathbb{N}}$, and the exponential $\mathbb{N}^{\mathbb{N}}$ exists and is the Baire space. Hence, using the homeomorphism $\mathbb{N} \cong 2^{2^{\mathbb{N}}}$ and the exponential law, we conclude that $\mathbb{N}^{\mathbb{N}} \cong (2^{2^{\mathbb{N}}})^{\mathbb{N}} \cong 2^{(\mathbb{N} \times 2^{\mathbb{N}})} \cong 2^{(2^{\mathbb{N}} \times \mathbb{N})} \cong (2^{\mathbb{N}})^{(2^{\mathbb{N}})}$. That is, we obtain the curious fact that the Cantor space elevated to the power Cantor space (exists and) is the Baire space [67]. The general criterion for exponentiability given in Chapter 8 reveals that the Cantor space is exponentiable but that the Baire space isn't. Thus, this gives a simple example of a space X which is exponentiable but X^X isn't. The same holds for e.g. $X = [0, 1]$ with the usual Hausdorff topology, but one needs to use tools such as the Ascoli–Arzela theorem in order to argue that the Hausdorff space X^X is not locally compact and hence not exponentiable.

9.10 Remark The Tychonoff theorem gives:

$$Y \text{ compact}, \quad X \text{ discrete} \implies Y^X \text{ compact}.$$

We have just proved a symmetric consequence, assuming that Y^X exists:

$$Y \text{ discrete}, \quad X \text{ compact} \implies Y^X \text{ discrete}.$$

It would be interesting to formulate this as a consequence of a "dual" Tychonoff theorem, but we don't know what such a theorem would state.

Notice that we didn't need to know what the topology of Y^X is in the proofs of the above two propositions. Everything is encapsulated in the proofs of continuity of $\forall$ and $\exists$. In fact, ignoring what we already know (on faith) from Chapters 8.4 and 8.5, we can easily construct plenty of open sets of Y^X using λ-technology:

9.11 Proposition *If $Q \subseteq X$ is compact and $V \subseteq Y$ is open then the set*

$$N(Q,V) = \{f \in Y^X \mid f(Q) \subseteq V\}$$

is open in Y^X provided the exponential exists.

Preliminary assumption: The exponential $\mathbb{S}^X$ exists.

Proof $\chi_{N(Q,V)}(f) = \forall_Q(\lambda x.\chi_V(f(x)))$ because $f \in N(Q,V)$ if and only if $\forall x \in Q. f(x) \in V$. □

9.12 Remark We have seen in Chapter 8.1 that if the exponential $\mathbb{S}^X$ exists, then so does the exponential Y^X for every Y. But if the exponential Y^X exists for a particular Y, there is no reason why the exponential $\mathbb{S}^X$ should exist. To give a trivial counter-example, consider the case in which Y is the two-point discrete space and X is connected. Then the exponential exists, even if X is not exponentiable, and is a two-point discrete space. So, in fact, applying the techniques of Chapter 10 to remove the preliminary assumption from the above proposition, we learn something that we didn't know from Chapter 8: Even when X is not exponentiable, if the exponential Y^X exists for a particular Y then its topology is finer than the compact-open topology. In fact, we can easily improve this bound on the topology of Y^X. The continuous map defined by the λ-expression of the proof of the above proposition is the composite

$$Y^X \xrightarrow{(\chi_V)^X} \mathbb{S}^X \xrightarrow{\forall_Q} \mathbb{S}.$$

Replacing the quantifier by any continuous map $h\colon \mathbb{S}^X \to \mathbb{S}$ whatsoever, one obtains the characteristic functions of the subbasic open sets that define the Isbell topology, and we conclude that if the exponential exists then it has a topology finer than the Isbell topology. As discussed in Chapter 8.5, this is as fine as we can get when X is exponentiable. However, we can carry on ignoring the compact-open and Isbell topologies.

The map considered in the following proposition is well defined because continuous functions map compact sets to compact sets and because the non-empty compact sets of $\mathbb{R}$ are bounded by the Heine–Borel theorem and hence the set $f(X)$ has a supremum in $\mathbb{R}$.

9.13 PROPOSITION *If X is compact and non-empty and the exponential $\mathbb{R}^X$ exists, where $\mathbb{R}$ is endowed with its usual Hausdorff topology, then the functional*

$$\begin{array}{rcl} \sup\colon \mathbb{R}^X & \to & \mathbb{R} \\ f & \mapsto & \sup f(X) \end{array}$$

is continuous.

Preliminary assumption. The exponential $\mathbb{S}^X$ exists.

PROOF Because the open sets of the form (a,∞) and $(-\infty,b)$ form a subbase of the topology of $\mathbb{R}$, it suffices to show that the sets $N_a = \sup^{-1}(a,\infty)$ and $N_b = \sup^{-1}(-\infty,b)$ are open in $\mathbb{R}^X$. We have that $f \in N_a$ iff $a < \sup(f)$ iff $\exists x \in X.a < f(x)$ iff $\exists x \in X.f(x) \in (a,\infty)$. Hence $\chi_{N_a}(f) = \exists_X(\lambda x.\chi_{(a,\infty)}(f(x)))$. Dually, we have that $f \in N_b$ iff $\forall x \in X.\sup(f) < b$ iff $\forall x \in X.f(x) < b$ iff $\forall x \in X.f(x) \in (-\infty,b)$. Hence $\chi_{N_b}(f) = \forall_X(\lambda x.\chi_{(-\infty,b)}(f(x)))$. $\square$

The continuous maps defined by the λ-expressions that give the characteristic functions of $N_a = \sup^{-1}(a,\infty)$ and $N_b = \sup^{-1}(-\infty,b)$ are

$$\mathbb{R}^X \xrightarrow{(\chi_{(a,\infty)})^X} \mathbb{S}^X \xrightarrow{\exists_X} \mathbb{S}, \qquad \mathbb{R}^X \xrightarrow{(\chi_{(-\infty,b)})^X} \mathbb{S}^X \xrightarrow{\forall_X} \mathbb{S}.$$

Notice that both sets are open in the Isbell topology and the second is open in the compact-open topology as well.

9.14 REMARK This generalizes to other topological lattices, such as e.g. continuous lattices under the Lawson topology, with the same proof (and with one half of the proof if one considers the Scott topology and the other if one considered the dual topology). Unfortunately, we won't have the opportunity of discussing the Lawson topology or its computational content in these notes [27].

Recall that a function is called *closed* if it maps closed sets to closed sets.

9.15 THEOREM *A space X is compact iff, for every space Y, the projection $\pi\colon X \times Y \to Y$ is closed.*

Preliminary assumption. The exponential $\mathbb{S}^X$ exists.

PROOF π is closed iff $W \in \mathcal{O}(X \times Y)$ implies $Y \setminus \pi(X \times Y \setminus W) \in \mathcal{O}\,Y$. But $Y \setminus \pi(X \times Y \setminus W) = \{y \in Y \mid \forall x \in X.(x, y) \in W\}$. This immediately gives ($\Rightarrow$). To prove ($\Leftarrow$), choose $Y = \mathbb{S}^X$ and $W = \{(x, p) \in X \times \mathbb{S}^X \mid p(x) = \top\}$ to get $Y \setminus \pi(X \times Y \setminus W) = \{p \in \mathbb{S}^X \mid \forall x \in X.p(x) = \top\} = (\forall_X)^{-1}(\top)$, which shows that $\forall_X\colon \mathbb{S}^X \to \mathbb{S}$ is continuous. □

9.16 EXERCISE Generalize the above proof to get the following well known characterization of proper maps (closed maps with compact fibres): A continuous map $f\colon X \to Z$ is proper iff for every continuous map $g\colon Y \to Z$, the pullback $f^*(g)$ is a closed map:

$$\begin{tikzcd} X \times_Z Y \arrow[r, "f^*(g)"] \arrow[d, "g^*(f)"'] \arrow[dr, phantom, "\lrcorner"] & Y \arrow[d, "g"] \\ X \arrow[r, "f"'] & Z. \end{tikzcd}$$

That is, $X \times_Z Y$ is the subspace $\{(x, y) \in X \times Y \mid f(x) = g(x)\}$ of the product, and the maps emanating from it are the (restrictions of) the projections. *Hint.* The theorem is this with $Z = 1$, the one-point space.

Vickers' book [140] tempts us to observe that what we have developed here is topology via first-order logic rather than just propositional logic. If we regard open sets as properties, second-order quantification occurs in the proof of our last proposition of this chapter.

If $\mathbb{S}^X$ exists, then its topology induces one in $\mathcal{O}\,X$ via the bijection of $\mathbb{S}$-valued continuous maps with open sets. We refer to this as the exponential topology of $\mathcal{O}\,X$ (as we did in Chapter 8.4). The following proposition generalizes the fact that finite intersections of open sets are open.

9.17 PROPOSITION *If X is exponentiable and $\mathcal{Q} \subseteq \mathcal{O}\,X$ is compact in the exponential topology of $\mathcal{O}\,X$, then $\bigcap \mathcal{Q}$ is open.*

PROOF $x \in \bigcap \mathcal{Q}$ iff $\forall U \in \mathcal{Q}.x \in U$. □

It might be possible to remove the exponentiability assumption from the above proposition using the techniques of Chapter 10, but not routinely so as far as we can see, and hence we don't label it as preliminary.

9.1 Notes

See Chapter 3.14. We have made some progress in combining the ideas reported here with those of Taylor's on abstract Stone duality and Vickers and Townsend on double powerlocales and exponentiation [141, 136] in order to tackle locales in arbitrary toposes, but this will be reported elsewhere, were we shall also address synthetic topology in toposes using dominances. We just mention that (1) the λ-expressions of the proofs are same, but one needs to argue in a different way that they perform the required jobs, (2) overtness assumptions have to be added whenever the existential quantifier is invoked in a proof and (3) we don't know how to tackle Theorem 9.15, but we believe that sheaf-topos extensions (gros toposes) of the category of locales will routinely do the job. In particular, in the localic setting, our Proposition 9.7 amounts to [70, Proposition 3.2], where what we call overt locales are called open locales, and for this case the topos machinery is not needed.

Chapter 10

Imaginary exponentials

Because certain exponentials don't exist in the world of real numbers, we pass to the world of complex numbers by adding imaginary exponentials, starting from $i = (-1)^{\frac{1}{2}}$. This extension is *conservative*: Every fact about real numbers deduced in the new world holds in the old world, provided it can be expressed there. In particular, functions and operations are generalized in such a way that they coincide with those of real numbers in the extended world. So, for instance, if y^x is an exponential of real numbers calculated in the world of complex numbers which happens to be real, then it coincides with the exponential calculated in the world of real numbers. An interesting example of an application of the conservativity of the extension is the extraction of roots of cubics: There is a formula for obtaining the three roots of a real polynomial of degree three with real roots, but the formula explicitly manipulates imaginary numbers on its way to its final real result.

Similarly, one can create imaginary exponentials of topological spaces in order to remove the exponentiability assumptions labelled "preliminary" in the previous chapter: In fact, the required exponentials occur in the proofs of the propositions but not in their formulations, so we don't care what they really are.

10.1 Generalized topological spaces

For the purposes of the previous chapter, we only need imaginary exponentials where the base is the Sierpinski space, but it is easier to create all missing exponentials in a single step.

In the following lemma, we refer to topological spaces as *real topological spaces* for emphasis.

10.1 Lemma (Generalized topological spaces) *There exists a category of* generalized topological spaces, *consisting of sets endowed with* generalized topologies, *such that*

1. *The category of real topological spaces lives inside that of generalized spaces: Every real topology can be regarded as a generalized topology, and the notion of continuity w.r.t. generalized topologies subsumes that of continuity w.r.t. real topologies.*

2. *The category of generalized spaces has finite products, which extend products of real spaces, and exponentials, which extend existing real exponentials.*

3. *The generalized topology of any generalized space A can be collapsed to a real topology, giving rise to a real space $\Re A$ with the same set of points as A, its* real part, *in such a way that, for any real space Z, continuity of a map $A \to Z$ is equivalent to that of the map $\Re A \to Z$ that has the same effect on points.*

4. *For any two real spaces X and Y, we have that $\Re(Y^X) = (X \to Y)$, the natural function space in the category of topological spaces.*

Proof See Section 10.2. □

In categorical language, the lemma says that there exists a well pointed category $\widehat{\mathrm{Top}}$ containing the category Top of topological spaces as a fully embedded subcategory, with embedding denoted by $(f : X \to Y) \mapsto (\hat{f} : \hat{X} \to \hat{Y})$, subject to the following properties: (1) $\widehat{\mathrm{Top}}$ is cartesian closed. (2) The embedding $\mathrm{Top} \hookrightarrow \widehat{\mathrm{Top}}$ preserves finite products and any exponential that exists in Top. (3) It has a left adjoint $\Re : \widehat{\mathrm{Top}} \twoheadrightarrow \mathrm{Top}$, a reflection. (4) For any $X, Y \in \mathrm{Top}$ the reflection of the exponential $\hat{Y}^{\hat{X}}$ in $\widehat{\mathrm{Top}}$ is the natural function space $(X \to Y)$ in Top. The notation $\widehat{\mathrm{Top}}$ is pronounced *top hat*. Just as a magician takes rabbits out of his, we take exponentials of topological spaces out of ours.

For the purpose of removing the preliminary exponentiability assumptions from the propositions of the previous chapter, we fix any such category of generalized spaces — it doesn't matter which. In view of the above lemma, we can unambiguously write Y^X for any two real spaces X and Y: The exponential always exists in the category of generalized spaces, and if it exists in the category of real spaces then both coincide. Hence we can now remove the preliminary assumptions of Chapter 9 in view of the following:

10.2 Corollary

1. *For real spaces X, Y, Z, continuity of a functional $Y^X \to Z$ is equivalent to that of the functional $(X \to Y) \to Z$ that has the same effect on points.*

2. *A subset Q of a topological space X is compact if and only if its universal quantification functional $\forall_Q \colon \mathbb{S}^X \to \mathbb{S}$ is continuous.*

3. *For any subset F of a topological space X, its existential quantification functional $\exists_F \colon \mathbb{S}^X \to \mathbb{S}$ is continuous.*

Proof For the first part, take $A = Y^X$ in the lemma and use the fact that $\Re(Y^X) = (X \to Y)$. For the second, take $Y = Z = \mathbb{S}$ in the first and apply Lemma 7.4. For the third, proceed as in the second and apply Lemma 7.5 instead. □

10.2 Examples of categories of generalized spaces

The remainder of this chapter can be safely omitted if the results of the previous section are taken on faith.

Plenty of supercategories of Top satisfying the conditions of Lemma 10.1 are known — we name a few: quasi-topological spaces [29], convergence spaces [64] (also known as filter spaces) and Dana Scott's equilogical spaces [9]. However, there is essentially only one known such category. It has been observed that the known examples are all full subcategories of the category of presheaves of topological spaces, and that the embeddings of Top into each of them land in the same portion of the category of presheaves [29, 109, 110]. The category of presheaves itself is not cartesian closed for size problems: Its homs are classes which fail to be (indexed by) sets in general and hence fail to form (set-valued) presheaves. However, each of the examples of subcategories mentioned above are exponential ideals closed under finite products.

What doesn't seem to have been observed in the literature is that the real part of an imaginary exponential of topological spaces is the natural function space, and hence we now develop some proofs. We briefly discuss convergence and equilogical spaces, and then we develop a full proof for quasi-topological spaces.

Convergence spaces. A *convergence space* is a set (of points) together with a relation between filters (of sets of points) and points, postulating which filters converge to which points, subject to suitable axioms — see e.g. [64]. The continuous maps are those that preserve the convergence relation. This category is known to satisfy the conditions of Lemma 10.1, except perhaps for item (4).

To see that item (4) holds, we consider a standard topology on the set of continuous maps. A filter of sets Φ on $\mathrm{C}(X,Y)$ is said to *converge continuously* to a function $f_0 \in \mathrm{C}(X,Y)$ if for any filter Γ converging to a point x of X, the filter generated by the filter base consisting of the sets $\{f(G) \mid f \in F\}$, for $F \in \Phi$ and $G \in \Gamma$, converges to $f_0(x)$. The *topology of continuous convergence* is obtained in the standard fashion whenever one is given a family of convergent filters: A set $N \subseteq \mathrm{C}(X,Y)$ is open if whenever any of the given filters converges to a member of N, the filter has N as a member. It is proved in [52] that the topology of continuous convergence coincides with the natural topology. Lemma 10.1(4) immediately follows from this and standard facts about the category of convergence spaces.

Equilogical spaces. A proof for equilogical spaces has been produced by Andrej Bauer after the first version of these notes was advertised [7]. Alternatively, as Alex Simpson pointed out to the author, the results for equilogical spaces and convergence space follow immediately from Rosolini [109] and the result for quasi-spaces given below.

Quasi-topological spaces. For the sake of completeness, we include a complete proof of Lemma 10.1 using quasi-spaces. An advantage of quasi-spaces is that they simplify the unwinding process described in Remark 9.6. Moreover, the proof of Lemma 10.1 becomes a triviality once the definitions and constructions are formulated. A disadvantage of quasi-spaces is that quasi-topologies are proper classes rather than sets.

To construct a quasi-space, we start with a set B of points, and, for each topological space X, we choose which functions from points of X to B we want to be continuous. But the chosen continuous maps have to interact with the existing continuous maps of topological spaces in the expected way. The details are as follows.

A *quasi-topology* on a set B consists of, for each topological space X, a collection of designated functions $s\colon X \to B$, called the continuous maps from X into B, such that (i) all constant maps are continuous and (ii) if $t\colon X \to Y$ is a continuous map of topological spaces then the composite $t \circ s\colon X \to B$ is continuous for every continuous $s\colon Y \to B$. A *quasi-space* is a set endowed with a quasi-topology. A *continuous map of quasi-spaces* is a function $f\colon B \to C$ such that the composite $f \circ s\colon X \to C$ is continuous for every continuous map $s\colon X \to B$.

Notice that we are using the word *continuous* for three purposes: (1) to name the well known concept for maps of topological spaces, (2) to name the members of a designated collection of maps from a topological space to a quasi-space, and (3) to name a defined notion of continuous map of quasi-spaces. The continuous maps of type (2) constitute the structure of a quasi-space. Not every collection of maps from topological spaces into a given set qualifies as a quasi-topology on the set: We require a compatibility condition with the continuous maps of type (1). Finally, the continuous maps of type (2) are used in order to define the continuous maps of type (3). For the moment, in order to avoid potential ambiguities, we use the letters X, Y, Z to range over topological spaces and the letters B, C, D to range over quasi-spaces. However, as shown in Lemma 10.5 below, there is no real danger of ambiguity. But let's first observe that we have a category.

10.3 Lemma *Continuous maps of quasi-spaces form a category under ordinary function composition.*

Proof It doesn't harm to include the routine verification: The identity function of any quasi-space is clearly continuous. Let $f\colon A \to B$ and $g\colon B \to C$ be continuous maps of quasi-spaces. In order to show that $g \circ f\colon A \to C$ is continuous, let $s\colon X \to A$ be a continuous map. Because f is a continuous map, the composite $f \circ s\colon X \to B$ is a continuous map, and, because g is a continuous map, so is $g \circ (f \circ s) = (g \circ f) \circ s$, as required. □

10.4 Definition (The quasi-topology of a topological space.) Each topological space X can be regarded as a quasi-topological space: Let the designated continuous maps into X be the topologically continuous ones. The space X regarded as a quasi-space in this way is officially denoted by $\hat{X}$.

The first part of the following lemma shows that type (3) continuity subsumes type (2), and the second that (3) subsumes (1).

10.5 Lemma

> **(Yoneda lemma)** *Continuity of a function $X \to B$ from a topological space X to a quasi-space B is equivalent to that of the function $\hat{X} \to B$ that has the same effect on points.*
>
> **(Yoneda embedding)** *Continuity of a function $X \to Y$ of topological spaces is equivalent to that of the function $\hat{X} \to \hat{Y}$ of quasi-spaces that has the same effect on points.*

Proof (Yoneda lemma) ($\Rightarrow$): Assume that $s\colon X \to B$ is continuous. In order to show that $s\colon \hat{X} \to B$ is continuous, we have to show that $t \circ s\colon X' \to B$ is continuous for any given continuous $t\colon X' \to \hat{X}$, i.e. any given continuous map $t\colon X' \to X$. But this follows from axiom (ii) of the definition of quasi-topology on B.

($\Leftarrow$): If $s\colon \hat{X} \to B$ is continuous, then so is $s\colon X \to B$ because the identity map $X \to \hat{X}$ is continuous by definition of the quasi-topology of $\hat{X}$, and the definition of continuity of $s\colon \hat{X} \to B$ requires that its composition with any map $X \to \hat{X}$ be continuous.

(Yoneda embedding): Taking $B = \hat{Y}$ in the Yoneda lemma, we conclude that a continuous map $X \to \hat{Y}$ is the same thing as a continuous map $\hat{X} \to \hat{Y}$. So it suffices to show that a continuous map $X \to \hat{Y}$ is the same thing as a continuous map $X \to Y$. But this is the definition of the quasi-topology of $\hat{Y}$. $\square$

We can define a continuous map of type $B \to X$ to be a continuous map $B \to \hat{X}$, obtaining a fourth type of continuous map. Compositions of continuous maps produce continuous maps, for any combination of the four types of maps, using the axioms for designated continuous maps, those for continuous maps of quasi-spaces, and Lemma 10.3. In view of these facts and conventions, we can generally allow topological spaces not to wear their quasi-space hats without any danger of confusion.

In categorical terminology, the second part of the above lemma shows that the category of topological spaces is fully embedded into that of quasi-spaces. We now prove that the latter is cartesian closed.

10.6 Lemma *The category of quasi-spaces has finite products.*

Construction: For quasi-spaces B_0 and B_1, take the underlying set of the product as the set-theoretical product $B_0 \times B_1$. As the continuous maps from a space X, take the functions $X \to B_0 \times B_1$ whose composition with the projections $\pi_i\colon B_0 \times B_1 \to B_i$ are continuous maps $X \to B_i$. These are just the pairings $(p_0, p_1)\colon X \to B_0 \times B_1$ of the continuous maps $p_0\colon X \to B_0$ and $p_1\colon X \to B_1$.

Proof It is clear that the designated continuous maps satisfy the required axioms. By construction, the projections $\pi_i\colon B_0 \times B_1 \to B_i$ are continuous maps. In order to verify the universal property, let C be a quasi-space, let $f_i\colon C \to B_i$ be a continuous map for each index i, and let $f\colon C \to B_0 \times B_1$ be the unique set-theoretical function with $f_i = \pi_i \circ f$. To show that it is a continuous map, let $s\colon X \to C$ be a continuous map. By construction of the continuous maps from topological spaces into the product, showing that the composite $f \circ s\colon X \to B_0 \times B_1$ is a continuous map is equivalent to showing that the composite $\pi_i \circ f \circ s$ is a continuous map for each index i. By construction of f, this is the same as $f_i \circ s$, which is a continuous map because f_i and s are. □

(Essentially the same construction and proof actually show that *all* limits exist.) It is immediate that the Yoneda embedding preserves finite products:

10.7 LEMMA $\hat{X} \times \hat{Y} = \widehat{X \times Y}$.

We now consider exponentials.

10.8 LEMMA *If $f\colon B \times C \to D$ is a continuous map of quasi-spaces, then for each $b \in B$, the function $f_b\colon C \to D$ defined by $f_b(c) = f(b, c)$ is continuous.*

PROOF Let $s\colon X \to C$ be a continuous map and $t\colon X \to B$ be the constant continuous map with value b. Because $(t, s)\colon X \to B \times C$ is a continuous map, so is $f \circ (t, s) = f_b \circ s$, as required. □

We thus have a function $\overline{f}\colon B \to D^C$ defined by $\overline{f}(b) = f_b$, where D^C is the set of continuous maps from C to D, called the transpose of f.

10.9 LEMMA *The category of quasi-spaces has all exponentials.*

Construction: Given quasi-spaces C and D, let the underlying set of the exponential D^C consist of the continuous maps from C to D, and let the continuous maps from a topological space X be the functions $u\colon X \to D^C$ such that for every continuous map $t\colon X \to C$, the composite $\varepsilon \circ (u, t)\colon X \to D$ is a continuous map, where $\varepsilon\colon D^C \times C \to D$ is the set-theoretical evaluation map.

PROOF It is easy to see that such designated continuous functions satisfy the required axioms. By construction, they make the evaluation function continuous. Hence if $\overline{f}\colon B \to D^C$ is a continuous map then so is $f\colon B \times C \to D$, because $f = \varepsilon \circ (\overline{f} \times \mathrm{id}_C)$ where $\mathrm{id}_C\colon C \to C$ is the identity. Conversely, assume that $f\colon B \times C \to D$ is a continuous map. In order to show that $\overline{f}\colon B \to D^C$ is also a continuous map, we have to show that if $s\colon X \to B$ is a continuous map then so is the composite $\overline{f} \circ s\colon X \to D^C$. By definition, this amounts to showing that $\varepsilon \circ (\overline{f} \circ s, t)$ is a continuous map for any given continuous map $t\colon X \to C$. But $\varepsilon \circ (\overline{f} \circ s, t) = f \circ (s, t)$, which, being a composition of continuous map, is itself continuous. □

10.10 Corollary *The designated continuous maps from a space X to D^C are precisely the functions of the form $\overline{f} \circ s\colon X \to D^C$ with $f\colon B \times C \to D$ a continuous map and $s\colon X \to B$ a designated continuous map.*

Proof (⇒): Because $\overline{f}\colon B \to D^C$ is a continuous map if $f\colon B \times C \to D$ is, we conclude that $\overline{f} \circ s\colon X \to D^C$ is a continuous map if the function $s\colon X \to B$ is. (⇐): If $u\colon X \to D^C$ is a continuous map, then, considering $B = D^C$, we have that $u = \overline{\varepsilon} \circ u$ where $\varepsilon\colon D^C \times C \to D$ is the evaluation map, because $\overline{\varepsilon}\colon D^C \to D^C$ is the identity. □

10.11 Lemma (The topology of a quasi-topological space) *Any quasi-space B has a unique topology, giving rise to a space $\Re B$ with the same sets of points as B, such that continuity of a map $\Re B \to Y$ into a topological space Y is equivalent to that of the map $B \to \hat{Y}$ with the same set of points.*

Proof Declare a subset U of B to be open if for every continuous map $s\colon X \to B$ from a topological space, the set $s^{-1}(U)$ is open. □

In categorical terminology, the above says that $\Re$ is the functor part of the left adjoint of the inclusion of the category of topological spaces into that of quasi-spaces, i.e. a reflection. Hence the inclusion preserves limits, which gives a categorical proof of Lemma 10.7. The following is an immediate consequence of Lemma 10.11 and Corollary 10.10.

10.12 Corollary *For any two real spaces X and Y, we have that $\Re(Y^X) = (X \to Y)$, the natural function space in the category of topological spaces.*

In particular, if the exponential Y^X exists in the category of topological spaces then it coincides with that calculated in the category of quasi-spaces.

10.3 Notes

Reinhold Heckmann has recently obtained a similar extension of the category of locales, but this is still unpublished (cf. Chapter 9.1).

Chapter 11

The Hofmann–Mislove representation theorem

If Q is a compact subset of a topological space X, then its universal quantification functional $A = \forall_Q \colon (X \to \mathbb{S}) \to \mathbb{S}$ is not only continuous but also *meet-linear*:

$$A(\top) = \top, \qquad A(p \wedge q) = A(p) \wedge A(q).$$

Here the left-hand occurrence of $\top$ is the constant map $X \to \mathbb{S}$ with value $\top \in \mathbb{S}$, the operation $(- \wedge -) \colon \mathbb{S} \times \mathbb{S} \to \mathbb{S}$ is the evident (continuous) conjunction map, and $p \wedge q$ is defined pointwise. The Hofmann–Mislove representation theorem can be read as saying that, under favourable circumstances, the converse holds: Every continuous meet-linear functional $A \colon (X \to \mathbb{S}) \to \mathbb{S}$ is the universal quantifier of some compact set. This is analogous to the Riesz representation theorem, which formulates a bijection from measures on a space X to linear functionals $F \colon (X \to \mathbb{R}) \to \mathbb{R}$, sending a measure μ to the linear functional $F = \int_\mu$ i.e. $F(f) = \int f d\mu$.

11.1 Compact saturated sets

To get a bijection between compact sets and continuous meet-linear functionals, one needs to carefully analyse the situation. In the absence of the T_1 separation axiom, one can find distinct compact sets Q and R with $\forall_Q = \forall_R$, as we shall see shortly. But we don't wish to postulate the T_1 separation axiom, because e.g. the Scott topology is T_1 only in the trivial case in which it is discrete.

For a space failing to satisfy the T_1 separation axiom, it proves useful to consider the *specialization order* on its set of points: $x \sqsubseteq y$ iff every neighbourhood of x is a neighbourhood of y. It is clear from the form of definition that this relation is reflexive and transitive, i.e. it is a preorder. Computationally, this occurs as the so-called *operational preorder*: y passes every observation that x does (cf. Section 3.3). The T_0 separation axiom is precisely the requirement that this preorder be antisymmetric, that is, a partial order, and the T_1 separation axiom amounts to the requirement that it be the identity relation. By construction, open sets are upper sets in the specialization order. By definition of closure, the relation $x \sqsubseteq y$ is equivalent to saying that x belongs to the closure of $\{y\}$. Hence continuous maps preserve the specialization order. Thus, if a space X is not T_1 then there exist distinct points x and y with $x \sqsubseteq y$, and we have distinct compact sets $Q = \{x\}$ and $R = \{x, y\}$ such that $\forall_Q(p) = \forall_R(p)$ for every $p \in (X \to \mathbb{S})$, as claimed above.

The *saturation* of a subset S of a topological space is its upper closure in the specialization order, denoted by $\uparrow S$, and S is called saturated if $S = \uparrow S$. Because this is the same as the intersection of the neighbourhoods of S, it has the same neighbourhoods as S, and hence S is compact if and only if its saturation is compact. It is now clear that

11.1 Proposition $\forall_Q = \forall_{\uparrow Q}$.

11.2 Sobriety

In order to formulate the Hofmann–Mislove theorem, we need a further notion. Notice that the filter ϕ of open neighbourhoods of a point of a space X is *completely prime*: Whenever $\bigcup_i U_i \in \phi$ for a family U_i of opens of X, there is some i with $U_i \in \phi$. Computationally, it is clear that when we observe a point of, say, the Cantor space, we don't actually see the point itself but rather (an enumeration of a base of) its open-neighbourhood filter. A space is called *sober* if it is T_0 i.e. no two distinct points share the same neighbourhoods (we don't see double) and every completely prime filter of opens is the open-neighbourhood filter of at least one point (what we see is there). It is the second part that is relevant, for the first can be cheaply enforced, by identifying points that share the same neighbourhoods (which is precisely what we did in Chapter 3 when we defined operational equivalence).

Hausdorff spaces are sober. This can be seen using the fact that a space is sober iff every irreducible closed set is the closure of a unique point. Here a non-empty closed set is called irreducible if it is not the union of two strictly smaller non-empty closed sets. To obtain an irreducible closed set from a completely prime filter of opens, take the union of the opens which are not in the filter. By the requirement of complete primeness, the union itself cannot be in the filter. The complement of the resulting open is the desired irreducible closed set. We leave the details to the interested readers.

An example of a non-sober space is the subspace 2^* of the Kahn domain 2^∞ discussed in Chapter 6.2: For each natural number n, the set $U_n \subseteq 2^*$ of finite sequences s which have the sequence 0^n as a prefix is open. An easy argument shows that the filter $\{U \in \mathcal{O}\, 2^* \mid U_n \subseteq U \text{ for some } n\}$ is completely prime but is not the open-neighbourhood filter of any point. It would be the open-neighbourhood filter of the non-existing point 0^ω if that point were added to the subspace. Every space has a sobrification, which collapses points that share the same neighbourhoods and adds any missing point. Moreover, the lattice of open sets of any space is isomorphic to that of its sobrification. In summary, a space is sober if it has as many points as its lattice of open sets allows it to have without violating the T_0 separation axiom. In computational terms, the set of data is uniquely determined by the set of observable properties. The sobrification of 2^* is 2^∞, and, more generally, any domain-base of a continuous dcpo under the Scott topology regarded as a subspace with the relative topology is known to have the whole domain as its sobrification. In particular, the sobrification of a poset under the Alexandroff topology is homeomorphic to the ideal completion of the poset under the Scott topology. For these and other facts concerning sobriety, see [55, 54, 69].

11.3 A representation theorem for continuous universal quantifiers

11.2 THEOREM (HOFMANN AND MISLOVE) *The following are equivalent for any topological space X.*

1. *X is sober.*
2. *The map that sends a set $Q \subseteq X$ to the filter $\{U \in \mathcal{O}\, X \mid Q \subseteq U\}$ is an order-reversing bijection from compact saturated sets to Scott open filters of open sets, with inverse given by $\phi \mapsto \bigcap \phi$.*
3. *The map that sends a set $Q \subseteq X$ to the functional $\forall_Q \colon (X \to \mathbb{S}) \to \mathbb{S}$ is an order reversing bijection from compact saturated sets to continuous meet-linear functionals.*

PROOF The original version [63] of the theorem is the implication (1) $\Rightarrow$ (2). The simplest proof we know is due to Keimel and Paseka [77]. The implication (2) $\Rightarrow$ (1) is folklore and easy: Use the fact that completely prime filters are Scott open. The equivalence (2) $\Leftrightarrow$ (3) follows immediately from the fact that $(X \to \mathbb{S})$ is homeomorphic to $\mathcal{O}\,X$ under the Scott topology as discussed in Chapter 8.4: Under this translation, a meet-linear map $A\colon \mathcal{O}\,X \to \mathbb{S}$ is one that transforms finite intersections into finite conjunctions. Continuity of A is equivalent to the requirement that the set $A^{-1}(\top)$ be Scott open, and meet-linearity to the requirement that it be a filter. □

Technically, one of the main uses of the Hofmann–Mislove representation theorem is the generalization of a well known property of Hausdorff spaces to sober spaces. We say that a collection $\mathcal{Q}$ of compact subsets of a topological space is *nested* if it is non-empty and for any two sets $R, S \in \mathcal{Q}$ there is $Q \in \mathcal{Q}$ with $Q \subseteq R$ and $Q \subseteq S$.

11.3 COROLLARY

1. *In a sober space, the intersection of any nested collection of compact saturated sets is compact (i.e. compact saturated sets form a dcpo under the reverse-inclusion order).*
2. *If $\bigcap \mathcal{Q} \subseteq U$ for a nested collection $\mathcal{Q}$ of compact saturated subsets of a sober space X and $U \in \mathcal{O}\,X$, then already $Q \subseteq U$ for some $Q \in \mathcal{Q}$.*

In particular, considering the empty open set, we see that, in a sober space, a nested collection of non-empty compact sets has non-empty intersection.

PROOF Use the easily established fact that Scott open filters are closed under the formation of directed unions. □

It is an interesting exercise to express this corollary in terms of universal quantifiers.

11.4 A representation theorem for continuous existential quantifiers

It is natural to wonder if there is a corresponding representation theorem for continuous existential quantifiers. It is clear that an existential quantification functional $E = \exists_F\colon (X \to \mathbb{S}) \to \mathbb{S}$ is *join-linear*, in the sense that

$$E(\bot) = \bot, \qquad E(p \vee q) = E(p) \vee E(q).$$

We denote the closure of a subset S of a topological space by S^-.

11.4 PROPOSITION $\exists_F = \exists_{F^-}$.

PROOF We have seen that via the homeomorphism of $(X \to \mathbb{S})$ and $\mathcal{O}\,X$ under the Scott topology, the existential quantifier of F becomes the continuous map $\exists_F \colon \mathcal{O}\,X \to \mathbb{S}$ such that $\exists_F(U) = \top$ iff $F \cap U \neq \emptyset$. But F meets $U \in \mathcal{O}\,X$ iff F^- meets $U \in \mathcal{O}\,X$ because, by definition of closure, every neighbourhood of a point of F^- meets F. $\square$

The representation theorem turns out to be easy and doesn't depend on any assumption on the space.

11.5 PROPOSITION *The following hold for any topological space X.*

1. *The map that sends $F \subseteq X$ to the set $\varphi(F) = \{U \in \mathcal{O}\,X \mid F \cap U \neq \emptyset\}$ is an order preserving bijection from closed sets to completely prime upper sets (i.e. upper sets which are inaccessible by arbitrary joins) with inverse given by $\psi(\mathcal{U}) = X \setminus \bigcup(\mathcal{O}\,X \setminus \mathcal{U})$.*

2. *The map that sends a set $F \subseteq X$ to the functional $\exists_F \colon (X \to \mathbb{S}) \to \mathbb{S}$ is an order-preserving bijection from closed sets to continuous join-linear functionals.*

PROOF (1): A routine verification shows that $\varphi(F)$ is indeed a completely prime upper set. Because $\mathcal{O}\,X \setminus \varphi(F)$ is the set of open sets disjoint from F, we see that $\psi(\varphi(F))$ is the complement of the largest open set disjoint from F, and this coincides with F if (and only if) F is closed. For $\mathcal{U} \subseteq \mathcal{O}\,X$, we have that $U \in \varphi(\psi(\mathcal{U}))$ iff $\psi(\mathcal{U}) \cap U \neq \emptyset$ iff $U \not\subseteq X \setminus \psi(\mathcal{U}) = \bigcup(\mathcal{O}\,X \setminus \mathcal{U})$. Hence, to conclude that $\varphi(\psi(\mathcal{U})) = \mathcal{U}$ if $\mathcal{U}$ is a completely prime upper set, we have to show that $U \not\subseteq \bigcup(\mathcal{O}\,X \setminus \mathcal{U})$ iff $U \in \mathcal{U}$. ($\Rightarrow$): For the sake of contradiction, assume that $U \notin \mathcal{U}$, i.e. $U \in \mathcal{O}\,X \setminus \mathcal{U}$. Then $U \subseteq \bigcup(\mathcal{O}\,X \setminus \mathcal{U})$, which contradicts the premise. ($\Leftarrow$). For the sake of contradiction, assume that $U \subseteq \bigcup(\mathcal{O}\,X \setminus \mathcal{U})$. Then the premise and the fact that $\mathcal{U}$ is an upper set gives $\bigcup(\mathcal{O}\,X \setminus \mathcal{U}) \in \mathcal{U}$, and completely primeness shows that some member of $\mathcal{O}\,X \setminus \mathcal{U}$ belongs to $\mathcal{U}$, which gives the desired contradiction.

(2): Again considering the homeomorphism $(X \to \mathbb{S}) \cong \mathcal{O}\,X$ for $\mathcal{O}\,X$ endowed with the Scott topology, a map $E \colon \mathcal{O}\,X \to \mathbb{S}$ is join-linear iff it transforms finite unions into finite disjunctions. But this is equivalent to saying that the set $E^{-1}(\top)$ is prime, or inaccessible by finite joins. And, as before, Scott continuity of E is equivalent to Scott openness of $E^{-1}(\top)$. But a set is inaccessible by all joins if and only if it is inaccessible by finite and directed joins. Hence E is continuous and join-linear iff $E^{-1}(\top)$ is a completely prime upper set. The result then follows from (1), because we know that $\exists_F(U) = \top$ iff $F \cap U \neq \emptyset$. $\square$

11.5 Notes

Heckmann has considered representations of power domains in terms of the functionals discussed here [58]. See also Vickers and Townsend [136].

Combining the above two representation theorems, we conclude that the meet-join-linear continuous functionals are in bijection with the completely prime filters of opens, and are precisely the continuous functionals that are both universal and existential quantifiers. Hence they coincide with the evaluation functionals of the form $F(p) = p(x)$, for x a point of the space, if and only if the space is sober. This is the concrete idea behind Taylor's approach to sobriety in abstract Stone duality [133].

One model of abstract Stone duality is the category of locally compact sober spaces. Because such spaces fail to be closed under the formation of compact saturated subspaces, the Hofmann–Mislove theorem doesn't hold in this model. However, in an arbitrary model, inspired by the Hofmann–Mislove theorem, one can regard meet-linear continuous functionals as articulating the notion of compact saturated set. This is precisely what Taylor profitably does in order to prove a version of the Baire category theorem [132].

Part III

Domain theory, topology and denotational semantics

Contents and organization

In the traditional approach to the topology of data types of languages such as the ones considered in Chapters 2–4, one starts with a partial order on the set of data, then constructs a topology from the order, then defines continuity from the topology, and finally shows that the functions that are definable in the language are continuous. This assignment of topological spaces to data types and of continuous maps to programs is known as the *Scott model* of the language, and constitutes an example of a *denotational semantics* of the language. From a computational point of view, this is the main topic of Chapter 12. In particular, we show that the data types of the programming language considered in Part I are densely injective spaces, and from this we conclude that the computationally defined function types of Part I coincide with the topologically defined exponentials of Part II.

Chapter 13 gives some applications to program development and correctnees proofs. In particular, a computational version of the Tychonoff theorem in the countable case is developed. In order to show that the resulting program has the correct termination properties, the classical Tychonoff theorem, with the aid of denotational semantics, is invoked. At the time of writing we don't know of any operational proof. But notice that an operational proof for a particular case of this program is given in Chapter 3.11. Also, we discuss some programs for exact real-number computation.

Chapter 12

Injective spaces, domains and function spaces

This chapter studies the combination of certain order-theoretic and topological ideas, eventually culminating in their application to programming-language semantics in Section 12.15 and to program development and correctness proofs in Chapter 13. We begin by formulating some basic applications.

12.1 Introduction

As in Chapter 4, in order to be able to be rigorous, we consider the programming language PCF rather than the equivalent subset of the language considered in Chapters 2 and 3.

The developments of Chapter 3 and of Chapters 7 and 9 are analogous, but, as the attentive reader probably realized, a potential difference arises in the construction of function spaces. Among other things, in this chapter we show that the computationally defined function types of Chapter 4 coincide with the topologically defined function spaces of Chapter 8:

12.1 Theorem *Topologize the PCF types as in Section 4.2. Then any PCF type X_σ is an exponentiable topological space, and $X_{\sigma\to\tau} = X_\tau^{X_\sigma}$.*

The second thing that the attentive reader will have noticed is that we worked with non-standard, relative notions of Hausdorff and discrete space in Chapter 3, but that we switched back to the standard (absolute) notions in Chapters 7 and 9. We officially formulate the topological relative notion of Hausdorff space, leaving the discrete case to the reader.

We say that a subspace X of a space $\bar{X}$ is *relatively Hausdorff* in $\bar{X}$ if the diagonal of X is relatively closed in $\bar{X} \times \bar{X}$. Clearly, (1) the relative notion implies the absolute one, (2) a closed subspace X of $\bar{X}$ is Hausdorff if and only if it is relatively Hausdorff, and (3) a subspace of a Hausdorff space is relatively Hausdorff. In terms of the Sierpinski space, X is relatively Hausdorff in $\bar{X}$ if there is at least one continuous map $(\neq_X)\colon \bar{X} \times \bar{X} \to \mathbb{S}$ such that for all $(x, y) \in X \times X$, we have that $(x \neq_X y) = \top$ if and only if $x \neq y$. There can be more than one such continuous map because we don't care what the value of $(x \neq_X y)$ is for $(x, y) \in \bar{X} \times \bar{X} \setminus X \times X$.

In this chapter we also prove:

12.2 Proposition *A dense subspace of a PCF type is relatively Hausdorff (resp. discrete) if and only if it is absolutely Hausdorff (resp. discrete).*

A third thing that the attentive reader will have noticed is that, in Chapter 3, what we want are spaces $X, Y, \ldots$ (e.g. the Baire and Cantor spaces) but what we get are larger spaces $\bar{X}, \bar{Y}, \ldots$ containing extraneous points. These are divergent points (at ground types), functions that map divergent to convergent or convergent to divergent points (at first-order types), and (at second and higher types) functionals of much more complicated behaviour combining divergence and convergence. In fact, this is the reason why we were forced to work with relative notions of Hausdorff and discrete space.

In order to analyse the situation, we introduce the following terminology: By an *environment* for a topological space X we mean a superspace $\bar{X}$. In general, X will be homeomorphically embedded into $\bar{X}$, but we often work as if X were a subset of $\bar{X}$. By a PCF-*environment* we mean an environment $\bar{X}$ with $\bar{X}$ a PCF type.

Because of the above phenomenon, we worked with a relative notion of continuous function in Chapter 3: We say that a function $f\colon X \to Y$ of topological spaces is *relatively continuous* with respect to environments $\bar{X}$ and $\bar{Y}$ for X and Y if there is at least one continuous function $\bar{f}\colon \bar{X} \to \bar{Y}$ with $\bar{f}(x) = f(x)$ for all $x \in X$. It is easy to see that every relatively continuous map is continuous.

We also prove in this chapter:

12.3 Theorem *Let X and Y be topological spaces with PCF environments $\bar{X}$ and $\bar{Y}$ such that X is dense in $\bar{X}$. Then every continuous map $X \to Y$ is relatively continuous with respect to the environments $\bar{X}$ and $\bar{Y}$.*

Notice that the above proposition follows from this theorem. It is natural to define f to be relatively PCF-definable if there is at least one definable extension $\bar{f}$. Different notions of relative definability are obtained for different extensions of the language. In view of Theorem 4.1, what the above theorem says is that, for the extension $\mathrm{PCF}_{\Omega}^{++}$, continuity of $f\colon X \to Y$ coincides with relative definability. Hence the above theorem roughly says that if a topological space arises as a dense subspace of a PCF type then we can (attempt to) compute with it in PCF. However, given the restricted nature of PCF types, density is too much to ask for. But one has:

12.4 THEOREM *Every subspace of a PCF type is second countable. Conversely, every second countable space X has a PCF-environment $\bar{X}$ such that for any space Y with a PCF-environment $\bar{Y}$, continuity of a function $X \to Y$ is equivalent to relative continuity.*

A proof of this can be found in [102], where it is shown that, in fact, there is a single PCF type which serves as an environment for all second-countable spaces.

A fourth thing that the attentive reader will have realized is that all functions definable in the languages considered in Chapter 3 and Chapter 4 have fixed points, because arbitrary equations are allowed in implicit definitions of functions, but, on the other hand, most topological spaces that one meets in practice fail to have this strong fixed-point property.

A fifth issue that doesn't arise in our considerations is the untyped λ-calculus, for which one needs an exponentiable space D homeomorphic to the exponential D^D. Moreover, related to this, even in the typed case one is interested in "domain equations" (which make a brief appearance in Chapter 13).

The remarkable paper *continuous lattices* by Dana Scott identifies a class of spaces that simultaneously takes care of the above five issues. These are the *injective spaces*, which, by a sixth insight of Scott, were found to have a purely order-theoretic characterization as the continuous lattices. For computational purposes, the top element of a continuous lattice is a nuisance, and soon afterwards Scott proposed to work with the densely injective spaces, which, using current terminology, he showed to coincide with the *continuous Scott domains* under the Scott topology.

12.2 Densely injective spaces

We call an environment $\bar{X}$ for a space X *tight* if X is dense in $\bar{X}$. A topological space D is called *densely injective* if every continuous map $f\colon X \to D$ extends to a continuous map $\bar{f}\colon \bar{X} \to D$ for every tight environment $\bar{X}$ of X.

For example, the real line with its usual Hausdorff topology is not injective: e.g. the continuous map $(x \mapsto 1/x) \colon \mathbb{R} \setminus \{0\} \to \mathbb{R}$ has no continuous extension to $\mathbb{R}$. In fact, as we shall see later, the only densely injective T_1 spaces are the empty space and the one-point space. We shall see that every PCF type is a densely injective space.

12.3 Densely injective spaces and function spaces

Before investigating what densely injective spaces look like, one can easily relate them to function spaces. For this, one doesn't need to know any internal characterization of the exponentiable spaces — one can work just with the definitions:

12.5 Theorem *If D is a densely injective space and Y is an exponentiable space then D^Y is a densely injective space.*

Proof Let $f \colon X \to D^Y$ be continuous and $\bar{X}$ be a tight environment for X. Then f is the transpose of a continuous map $g \colon X \times Y \to D$ and $\bar{X} \times Y$ is a tight environment for $X \times Y$. Hence, because D is densely injective, g has a continuous extension $\bar{g} \colon \bar{X} \times Y \to D$, which has a continuous transpose $\bar{f} \colon \bar{X} \to D^Y$. A routine calculation shows that $\bar{f}$ extends f, which shows that D^Y is densely injective. □

A similar kind of proof shows that

12.6 Proposition *The densely injective spaces are closed under the formation of products.*

In order to show that the densely injective spaces are exponentiable, and hence conclude that they form a cartesian closed category of spaces, we need to develop an internal characterization for them. We do this in several steps.

12.4 Topology from order and conversely

We have already briefly met the operational preorder of a data type (Chapter 3.3) and its topological manifestation, the specialization order (Chapter 11). Dana Scott discovered that, for certain data types, the topology is uniquely determined by the operational order, and, conversely, the order is uniquely determined by the topology. Such data types arise as *domains*. Thus, domains can be seen either as special kinds of partially ordered sets, or as special kinds of topological spaces. To move from the order-theoretic view to the topological view, one takes the Scott topology of an order, and, in the other direction, one takes the specialization order of a topology. We have already met some examples, such as the PCF types, the Kahn domain and the interval domain. We now look at these and other examples in more detail.

In programming-language semantics, the order-theoretic view is emphasized. For the applications we have in mind (e.g. Chapter 13), the topological view is crucial. In any case, what makes domain theory a rich subject is the interplay between order-theoretic and topological notions.

When we think of an ordering $x \sqsubseteq y$ in computational terms, we say that y is more defined than x, that y has more information content than x, or that y is better than x.

12.5 Directed complete posets

A subset S of a poset D is called *directed* if it is non-empty and for any two $s, t \in S$ there is some $u \in S$ with $s \sqsubseteq u$ and $t \sqsubseteq u$. For example, any chain (linearly ordered subset) is a directed set. In our applications, we think of a directed subset S of D as an "abstract computation" of an element of D. Its members are the concrete partial outputs that approximate the ideal result of the computation. The defining condition says that any two partial outputs eventually get superseded by a third, finer output (c.f. the Church–Rosser property of the λ-calculus). The ideal result is taken to be the join (also called supremum or least upper bound) of the concrete partial outputs. Because we want all computations to compute something, we postulate that D has joins of directed subsets. By a *directed complete poset*, or *dcpo* for short, we mean a poset with joins of directed subsets.

12.6 The Scott topology of a dcpo

We have already met the Scott topology a number of times (from a computational point of view in Chapters 4 and 6, and from a mathematical point of view in Chapters 8 and 11). A subset U of a dcpo is called *Scott open* if it is an upper set (i.e. $u \in U$ and $u \sqsubseteq x$ together imply $x \in U$) and every directed set with join in U already has a member in U. If we think of U as a test, the first condition says that if something passes the test then so does anything more defined, and the second that if the ideal result of a computation passes the test then some concrete partial output of the computation already passes the test. That is, the test can be observed at a finite stage of the computation.

For a proof of the following proposition, and other propositions provided without proof, see e.g. [55].

12.7 Proposition *1. The specialization order of the Scott topology of a dcpo is the given order.*

2. A function $f\colon D \to E$ of dcpos is Scott continuous (i.e. continuous with respect to the Scott topologies of D and E) if and only if it preserves the order (i.e. $x \sqsubseteq y$ implies $f(x) \sqsubseteq f(y)$) and directed suprema (i.e. $f(\bigsqcup S) = \bigsqcup_{s \in S} f(s)$ for every directed subset S of D).

Hint: Notice that, for preservation of directed suprema, the inequality $\sqsupseteq$ follows by order-preservation. To prove both items, argue by contradiction at certain points. For this, first show that the set $\{x \mid x \not\sqsubseteq y\}$ is Scott open for any y. (When we move to continuous dcpos, positive arguments are available.)

12.7 Continuous dcpos

For the purpose of giving mathematical meaning to computer programs of languages such as PCF, one can go a long way with directed complete posets (or even with posets that have suprema of ascending sequences). However, in order to consider, for example, computability notions, one considers continuous dcpos [120] or the particular case of algebraic dcpos [35, 103]. Moreover, as we shall see, in applications of mathematical semantics to correctness proofs of programs one often uses arguments involving algebraicity or continuity. Because we want to consider continuous, non-algebraic domains such as the interval domain, we introduce the more general situation.

We say that an element x of a dcpo is *way below* an element y, written $x \ll y$, if for every directed set S with $y \sqsubseteq \bigsqcup S$ there is some $s \in S$ with $x \sqsubseteq s$. Thinking of a directed set as an abstract computation as above, $x \ll y$ can be interpreted as saying that any computation of y or something more defined than y eventually outputs x or something more defined than x. In other words, x is an unavoidable step in any computation of y or something more defined than y. The continuity axiom for a dcpo says that the unavoidable steps are not only unavoidable but also enough, and, moreover, they form an abstract computation: A dcpo is called *continuous* if the set $\{b \mid b \ll x\}$ is directed and has x as its join. For applications to computation, one requires that there are enough unavoidable parts among a countable *basis* of the continuous dcpo — see e.g.[3] or [55]. These are meant to be the elements that a mechanical computer can output in finite time.

The following is an immediate, but rather useful, consequence of the definition:

12.8 PROPOSITION *In a continuous dcpo,*

1. *if* $b \sqsubseteq y$ *holds for every* $b \ll x$ *then* $x \sqsubseteq y$.

2. *if* $x \not\sqsubseteq y$ *then there exists* $b \ll x$ *such that already* $b \not\sqsubseteq y$.

The following lemma, whose proof is not so direct, plays a major role in the theory, where the second item is known as the *interpolation property*:

12.9 LEMMA *In a continuous dcpo,*

1. *if* $x \ll y \sqsubseteq \bigsqcup S$ *for* S *directed then* $x \ll s$ *for some* $s \in S$,

2. *if* $x \ll y$ *then there is some* b *with* $x \ll b \ll y$.

Using this, one easily shows that:

12.10 PROPOSITION

1. *In a continuous dcpo,*

   (a) *the set* $\twoheaduparrow b \stackrel{\mathrm{def}}{=} \{x \mid b \ll x\}$ *is Scott open,*

   (b) *an upper set* U *is Scott open if and only if for every* $u \in U$ *there is* $b \ll u$ *already in* U,

   (c) *the sets of the form* $\twoheaduparrow u$ *form a base of the Scott topology.*

2. *An order-preserving function* $f\colon D \to E$ *of continuous dcpos is Scott continuous if and only if whenever* $b \ll f(x)$, *there is some* $c \ll x$ *such that already* $b \ll f(c)$.

From a computational point of view, item (1b) says that if u passes an observation U then some unavoidable part of u already passes the observation.

12.8 Topological view of continuous dcpos

In the world of continuous dcpos, not only can we move between the order-theoretic and topological views, but we can start from either end. Topologists can regard the first part of the following as an order-theoretic characterization of certain spaces, and order-theoreticians can regard the second as a topological characterization of certain posets. Define a subset of a topological space to be *supercompact* if every open cover has a singleton subcover, and a topological space to be locally supercompact if each point has a base of supercompact neighbourhoods.

12.11 THEOREM

1. *The locally supercompact sober spaces are precisely the continuous dcpos under the Scott topology.*

2. *The continuous dcpos are precisely the locally supercompact sober spaces under the specialization order.*

We omit the proof, but we observe that if $b \ll x$ then the principal filter $\uparrow b = \{u \mid b \sqsubseteq u\}$ is a supercompact neighbourhood of x (which is open if and only if $b \ll b$).

In particular, continuous dcpos under the Scott topology are locally compact and hence exponentiable topological spaces (cf. Chapter 8).

12.9 Order-theoretic view of densely injective spaces

A *continuous Scott domain* is a continuous dcpo with meets of non-empty sets, or, equivalently, joins of upper bounded sets. In particular, a non-empty continuous Scott domain has a least element, which arises as the meet of the whole domain.

12.12 THEOREM *Restricting attention to T_0 spaces, we have:*

1. *The densely injective spaces are precisely the continuous Scott domains under the Scott topology.*

2. *The continuous Scott domains are precisely the densely injective spaces under the specialization order.*

We again omit the proof, but we indicate how one can calculate extensions. If D is a continuous Scott domain under the Scott topology and we have a dense subspace X of a space $\bar{X}$ and a continuous map $f\colon X \to D$, there is in fact a unique largest continuous extension $\bar{f}\colon \bar{X} \to D$ in the pointwise order, given by the formula

$$\bar{f}(\bar{x}) = \bigsqcup\{\bigsqcap f(\bar{U} \cap X) \mid \bar{x} \in \bar{U} \in \mathcal{O}\,\bar{X}\}.$$

Notice that the meet is of a non-empty set by density of X in $\bar{X}$. Analysts will recognize this as a limit-inferior construction related to lower semicontinuous functions, and, indeed, lower semicontinuous functions are a special case of this situation, when one considers the Scott topology of the natural order of the real line (with points at infinity to get the required completeness property), which is just the topology of lower semicontinuity — see [54] or [55].

12.10 Continuous Scott domains and function spaces

We have already seen that if D is a densely injective space and Y is an exponentiable space then D^Y is a densely injective space. By the previous two theorems, we know that densely injective spaces are locally compact and hence exponentiable. This proves:

12.13 Corollary *The densely injective spaces form a cartesian closed category with finite products and exponentials inherited from the category of topological spaces.*

The continuous Scott domains, considered as order-theoretical gadgets, also form a cartesian closed category. Products are given by set-theoretical products under the coordinatewise order, and exponentials are given by sets of Scott continuous maps under the pointwise order. Then the Scott-topology construction can be seen as a (full and faithful) inclusion functor of continuous Scott domains into topological spaces with image landing precisely into the category of densely injective spaces. This functor preserves finite products. It is crucial here that the source category consists of continuous dcpos. In fact, the Scott-topology construction regarded as a functor from dcpos to topological spaces doesn't preserve binary products in general. But this is the case if one of the factors is a continuous dcpo. From the machinery developed here, one can also see that the inclusion functor of continuous Scott domains preserves exponentials. It suffices to show that, for densely injective spaces D and E, the specialization order of E^D under the compact-open topology coincides with the pointwise specialization order. This can be routinely done by considering point-open sets (using the fact that singletons are compact).

A number of examples of continuous Scott domains examples have already occurred: (1) The interval domain discussed in Chapter 6. (2) The lattice of open sets of an exponentiable space in Chapter 8 — see Chapter 12.11 for more details. (3) Various algebraic dcpos — see Section 12.12 for more details.

12.11 Continuous lattices, injective spaces and exponentiable spaces

By a *continuous lattice* it is meant a continuous complete lattice (equivalently, a continuous dcpo with finite joins, including that of the empty set). Scott showed that the injective spaces (defined by removing the density condition in the definition of densely injective space) are precisely the continuous lattices under the Scott topology. The same argument as that of Theorem 12.5 shows that if D is injective and Y is exponentiable then D^Y is injective. It follows from the very definition of subspace that the Sierpinski space is injective. Hence if X is exponentiable then $\mathbb{S}^X$ is injective, and thus a continuous lattice under the Scott topology. Via the bijection of $\mathcal{O}\,X$ with Sierpinski-valued continuous maps, we conclude that $\mathcal{O}\,X$ is a continuous lattice if X is an exponentiable topological space, as claimed in Chapter 8.5.

12.12 Algebraic dcpos

A poset is called *algebraic* if every element x is the directed join of the elements $b \sqsubseteq x$ with $b \ll b$. An element b with $b \ll b$ is called *compact* and sometimes *finite* (neither terminology is optimal). It is clear from the definition that every algebraic dcpo is continuous. Hence, by Lemma 12.9, in an algebraic dcpo we have that every computation of b has to output b at some stage, and this is a good reason for referring to such elements as finite in a computational context.

For example, the Kahn domain (Chapter 6.2) is an algebraic dcpo with the truly finite sequences playing the role of finite elements in the order-theoretic sense just defined. The data type of natural numbers of the programming languages considered in Chapters 3 and 4 is interpreted as the algebraic dcpo consisting of the natural numbers together with a new element $\perp$ and order defined by $x \sqsubseteq y$ if and only if $x = \perp$ or $x = y$. All elements are finite and the directed subsets have finite cardinality, and hence this is trivially an algebraic dcpo, which is denoted by $\mathbb{N}_\perp$.

So notice, in particular, that finite doesn't mean computable in finite time: $\perp$ takes an infinitely long time to be computed, and so do the finite sequences of the Kahn domain, which is an algebraic poset under the prefix order. What finite means is that all the information content will be explicitly exhibited after a finite amount of time: This is fine for $\perp$, which has no information content, and so is for the finite elements of the Kahn domain. Notice that the Kahn domain has a bottom element, namely the empty sequence. Results of computations that terminate in a finite number of steps correspond to maximal finite elements. In the Kahn domain, for example, there are none. (One can work with a variation of the Kahn domain in which finite sequences corresponding to finite terminating computations are added. For example, consider a symbol that can only occur as the last element of a finite sequence, and still work with the prefix order.)

12.13 Scott domains

A *Scott domain* is a continuous Scott domain which is also an algebraic dcpo. The examples of algebraic dcpos just given are in fact examples of Scott domains. Moreover, finite products and exponentials of Scott domains are again Scott domains. So, for example, the Baire domain $(\mathbb{N}_\perp)^{\mathbb{N}_\perp}$ is algebraic. It has some unexpected finite elements, namely the constant functions with non-bottom value. Computationally speaking, these are the functions that don't look at their argument. Their relatives, the functions that are constant on non-bottom arguments but map bottom to bottom, are not finite. This gives examples of finite elements which have non-finite elements below them in the information order. Notice that we defined in Chapter 3 the Baire space to be the subset of the Baire domain consisting of functions that map bottom to bottom and natural numbers to natural numbers.

12.14 Exercise Show that, under the relative Scott topology, they form a space homeomorphic to the Baire space (a product of countably many copies of the discrete space with countably many points).

12.14 Fixed points, function spaces and recursive definitions

The implicit definitions of functions that occur in Chapter 3, known as *recursive definitions*, are the computational analogue of the differential equations in physics. Just as typical physicists write down differential equations and take the existence (and sometimes uniqueness) of their solutions for granted on operational grounds, so do typical computer scientists with their recursive definitions (and that's the way we proceeded in Chapter 3).

The following lemma is often attributed variously to Knaster, Tarski, Davis and Kleene:

12.15 LEMMA *Let D be a poset with joins of countable ascending chains and a bottom element. Then every function $f\colon D \to D$ that preserves order and suprema of countable ascending chains has a least fixed point. That is, there is x with $f(x) = x$ and with $x \sqsubseteq y$ for every y with $f(y) = y$.*

PROOF (Sketch) $x = \bigsqcup_{n \in \mathbb{N}} f^n(\perp)$. □

12.16 COROLLARY *Every continuous endomap of a non-empty densely injective space has a least fixed point in the specialization order.*

In practice, this is used as follows. Given an implicit (or recursive) definition of a function $f\colon D \to E$ of the form

$$f(x) = \text{some expression involving } x \text{ and } f \text{ itself},$$

one defines, from that expression, a continuous map $F\colon E^D \to E^D$ such that the above equation is equivalent to

$$f = F(f).$$

Examples of this situation occur in Chapter 13 below. Then we know that the original equation has at least one continuous solution, in fact a smallest one in the pointwise (operational or specialization) order. On computational grounds, one takes the smallest solution. The precise mathematical reason why this is what we are forced to choose is part of a theorem known as *computational adequacy* [101, Theorem 3.1], which we now briefly discuss.

12.15 The Scott model of PCF and its fundamental properties

To fully answer the questions posed in Section 12.1, we should define the domain-theoretic semantics of PCF and go through various fundamental theorems. The reader is referred to e.g. Streicher's notes [128] or the original paper by Plotkin [101].

In summary, the ground type of natural numbers is interpreted as the domain $\mathbb{N}_\perp$ discussed in Section 12.12 and similarly the ground type of booleans is interpreted by adding a bottom element to the discretely ordered two-element set. Function types are interpreted as exponentials (see Section 12.10). Implicit definitions of functions are interpreted via least fixed points as discussed in Section 12.14. Constants for primitive functions of the language are interpreted as suitable continuous functions. The simply-typed λ-calculus machinery is interpreted via the exponential laws for function spaces.

This is known as the *Scott model* of PCF. This mathematical model has a number of fundamental computational properties: (1) It is *computationally adequate*: A program of ground type evaluates to a non-bottom value in finitely many steps, according to the operational semantics of the language, if and only if it denotes that value in the model. (2) By extending the language with the parallel-or operation mentioned in Chapter 4, the model becomes *fully abstract*: Two programs of the same type denote the same entity in the model if and only if they are operationally equivalent. (3) By further adding the existential quantifier, it becomes *Turing-universal*, for a notion of computability for elements (including function(al)s) of domains: Every computable element of the model is definable in the language. (4) By further adding what we called external inputs in Chapter 4, it becomes *fully complete*: All elements of the model become definable in the language.

These are the ingredients used in the proof of Theorem 4.1. Thus, working with PCF as a programming language and with $\mathrm{PCF}^{++}_{\Omega}$ as its data language, as in Chapter 4, is equivalent to working with PCF^{++} as a notational system for the computable entities of the Scott model. In order to mathematically argue about programs in Chapter 13, we adopt the latter. This has the advantage that we can completely ignore the evaluation mechanism of the language in order to establish program correctness.

Historically, this is the view of affairs originally proposed by Scott [113]. He introduced a logic for reasoning about programs, together with a domain model for it. Later on, Plotkin [101] regarded the terms of the logic as a programming language, with a subset of the equational rules of the logic as its evaluation mechanism, and proved the basic computational properties (1)-(3), and implicitly (4), of the mathematical model. The logic and the programming language are called LCF and PCF, which stand for *logic of computable functions* and *programming language for computable functions*.

Of course, this was just the beginning of work on programming-language semantics, and many more languages and mathematical models have been investigated. From our perspective, the Scott model has the advantage of being intrinsically topological in nature, as Scott proved and emphasized right from the beginning. However, we observe that the synthetic topology developed in Chapter 3 is model-independent. In fact, we deliberately based the theory on operational grounds, in order to make it clear that the topology is there independently of what mathematical model one favours. This is particularly important in the absence of the parallel operations, in which case fully abstract models look very different from the Scott model.

12.16 Notes

Some historical notes on domain theory have already been given. Detailed notes can be found in [55] (see also [68] regarding function spaces). Theorem 12.12 was formulated and proved by Scott, but it was only published as an exercise in [54] for a long time, until it eventually appeared in [55] (see also [46]). What appeared in print first was the characterization by Scott of the injective spaces as the continuous lattices under the Scott topology [111], with the applications already mentioned above.

Theorem 12.5 is due to Keimel and Gierz [76]. The proof given here is due to Johnstone and Joyal [69], as are the arguments given in Section 12.11. A more direct proof of the result established in Section 12.11, which avoids the characterization of the injective spaces as the continuous lattices under the Scott topology, is contained in the reference on which Chapter 8 is based.

Chapter 13

Sample applications

In this chapter we give some non-trivial examples of how topology and domain theory, with the aid of denotational semantics, can be applied to develop programs and prove their correctness. We revert to the programming language Haskell, but, as in Chapters 2 and 3, the fragment considered here can be regarded as the language PCF discussed in Chapter 4. In fact, in order to establish program correctness, we treat programs in our language as PCF programs interpreted in the Scott model, as explained in Chapter 12.15. We continue from the program fragments constructed in Chapters 2 and 3. In this chapter, all results depend on the external view of data.

13.1 A computational version of the countable Tychonoff theorem

We have seen that there is a λ-expression that proves the binary case of the Tychonoff theorem (Proposition 9.5), which can also be regarded as a program (Proposition 3.18). It is natural to wonder whether one can prove the Tychonoff theorem in the arbitrary case via the λ-calculus. We don't know the answer to this question. However, we are able to develop a program that implements the Tychonoff theorem in the countable case, which allows us to conclude that a product of an r.e. sequence of computationally compact spaces is computationally compact. The given proof of correctness of the program relies on the classical Tychonoff theorem.

We have shown that, under the external view of data types defined in Chapter 3 and explored in Chapters 4 and 12, the Cantor space is computationally compact in the sense that its universal quantification functional is computable. The Tychonoff theorem shows that an arbitrary product of compact spaces is compact. Because the Cantor space is the product of countably many copies of the two-point discrete space, we ought to be able to define the quantification functional of the Cantor space from the trivial quantification functional for the two-point discrete space by implementing the countable case of the Tychonoff theorem.

Ideally, what we would like is, given a sequence $\forall_i \colon \mathbb{S}^{D_i} \to \mathbb{S}$ of quantifiers of a sequence of compact subspaces Q_i of data types D_i, to construct the quantifier $\forall \colon \mathbb{S}^{\prod_i D_i} \to \mathbb{S}$ of the compact subspace $\prod_i Q_i$ of the countable product $\prod_i D_i$. The difficulty here is that the countable product $\prod_i D_i$ is not definable in our language unless the data types D_i are the same (what we would need are dependent types). For simplicity, we accept this restriction but we still allow the subspaces Q_i to be different, but we remark that the dependently typed program would be literally the same as the one proposed here, with a different type signature. This is the first obstacle that we face.

The second is that, in the absence of parallel features, we are not able to solve the problem from the above data. We additionally need to be given a choice of points $u_i \in Q_i$, i.e. a sequence $u \in \prod_i Q_i$.

We define an abbreviation for a type of countable powers or sequences, and recall the previously introduced abbreviation for quantifiers:

```
type Seq a = Nat -> a
type Quant a = (a -> S) -> S
```

Then the type of our countable-Tychonoff program is

```
tych :: (Seq a, Seq (Quant a)) -> (Quant (Seq a))
```

The first argument of the function is the sequence of choices of points for the given sequence of compact subspaces of the data type `a`, and the second is the sequence of quantifiers for the given compact subspaces. What results is a quantifier for the product of the given sequence of compact subspaces.

For example, to define the quantifier for the Cantor space within the Baire data type, we can first define the quantifier for the two-point discrete space $2 = \{0, 1\}$ regarded as a subspace of the data type of natural numbers,

```
forall_2 :: Quant Nat
forall_2 p = p(0) /\ p(1)
```

and then define the quantifier of the Cantor space within the Baire data type as, where c is the arbitrary choice:

```
c :: Baire
c = \i -> 0

forall_C :: Quant Baire
forall_C = tych(c, \i -> forall_2)
```

In order to obtain a program for the countable-Tychonoff functional, we first recall the program for the quantifier of the Cantor space provided in Chapter 3:

```
forall_C :: Quant Baire
forall_C(p) = p(ifs(forall_C(\s -> p(cons(0,s))), c))
           /\ p(ifs(forall_C(\s -> p(cons(1,s))), c))
```

Notice that arbitrary choices already occur in this case. Binary conjunction can be regarded as quantification over the two-point discrete space, and hence we can equivalently write:

```
forall_C(p) =
  forall_2(\x -> p(ifs(forall_C(\s -> p(cons(x,s))), c)))
```

By allowing the quantifier over the two-point discrete space to vary, we finally get the required program:

```
tych :: (Seq a, Seq (Quant a)) -> (Quant (Seq a))
tych(u,quants)(p) =
  forall(\x -> p(ifs(forall'(\s -> p(cons(x,s))), u)))
    where forall = hd(quants)
          u' = tl(u)
          quants' = tl(quants)
          forall' = tych(u',quants')
```

For this we need to generalize the types of the head, tail and cons maps:

```
hd :: Seq a -> a
tl :: Seq a -> Seq a
cons :: (a, Seq a) -> Seq a
```

We keep the same definitions of the functions as in Chapter 2.

In the remainder of this section we prove that the program `tych` satisfies the required property. As discussed at the beginning of this chapter, our proof uses the Scott model of a fragment of our language, which is essentially PCF, and relies on computational adequacy as reported in Chapter 12.15.

But notice that the specification of the program is purely operational: Given a sequence of quantifiers for subsets of a data type, produce the quantifier for the product. We use denotational semantics to prove that the program satisfies this operational specification. At the time of writing, we don't know of any operational proof (but recall that an operational specification and proof of correctness for a particular case of this program has been provided in Chapter 3.11).

If D is the domain that interprets the data type `a`, then the interpretation of the program `tych` in the model is the least continuous solution in

$$A\colon (\mathbb{N}_\perp \to D) \times (\mathbb{N}_\perp \to ((D \to \mathbb{S}) \to \mathbb{S})) \to (((\mathbb{N}_\perp \to D) \to \mathbb{S}) \to \mathbb{S}),$$

of the equation

$$A(u,\alpha)(p) = \alpha_0(\lambda x.p(\text{if } A(u',\alpha')(\lambda s.p(\text{cons}(x,s))) \text{ then } u)),$$

Equivalently, as explained in Chapter 12.14, it is the least fixed point of the explicitly defined functional

$$\Phi(A) = \lambda(u,\alpha).\, \lambda p.\, \alpha_0(\lambda x.\, p(\text{if } A(u',\alpha')(\lambda s.\, p(\text{cons}(x,s))) \text{ then } u)).$$

Here u' and α' denote the tails of the sequences u and α, the function cons is the evident interpretation of the `cons` program, and the if-then construction denotes the evident interpretation of the `ifs` program. Notice that the functional Φ is continuous, as required to apply fixed-point techniques of Chapter 12.14, because functions which are λ-definable from continuous functions are themselves continuous (all required exponentials exist in this context, as explained in the previous chapter).

For the sake of clarity and simplicity of exposition, we replace occurrences of function spaces with domain $\mathbb{N}_\perp$ by countable cartesian powers and we write some function spaces using the equivalent exponent notation:

$$A\colon D^\omega \times (\mathbb{S}^D \to \mathbb{S})^\omega \to (\mathbb{S}^{D^\omega} \to \mathbb{S}).$$

The readers can easily check that this doesn't make any essential difference.

By Lemma 12.15, the least continuous solution is $A = \bigsqcup_n A_n$, where

$$\begin{aligned} A_0(u,\alpha)(p) &= \perp, \\ A_{n+1}(u,\alpha)(p) &= \alpha_0(\lambda x.p(\text{if } A_n(u',\alpha')(\lambda s.p(\text{cons}(x,s))) \text{ then } u)). \end{aligned}$$

In order to prove that this function A satisfies the required specification, we need to prove a slightly more general proposition. For each natural number k, define

$$A^{(k)}(u,\alpha)(p) = A(u^{(k)},\alpha^{(k)})(\lambda s.p(s^{(k)})),$$

where $t_i^{(k)} = t_{i+k}$ for any sequence t. A simple induction on k using the equation that A satisfies shows that

$$A^{(k)}(u,\alpha)(p) = \alpha_k(\lambda x.p(\text{if } A^{(k+1)}(u,\alpha)(\lambda s.p(\text{cons}(x,s))) \text{ then } u^{(k)})).$$

Hence, if we define

$$\begin{aligned} A_0^{(k)}(u,\alpha)(p) &= \bot, \\ A_{n+1}^{(k)}(u,\alpha)(p) &= \alpha_k(\lambda x.p(\text{if } A_n^{(k+1)}(u,\alpha)(\lambda s.p(\text{cons}(x,s))) \text{ then } u^{(k)})), \end{aligned}$$

we conclude that $A^{(k)} = \bigsqcup_i A_n^{(k)}$, because continuous maps commute with directed joins.

Now suppose that Q_i is a sequence of compact subspaces of D with corresponding sequence of quantifiers $\alpha_i \colon \mathbb{S}^D \to \mathbb{S}$, and let $u \in \prod_i Q_i$. We show that $A^{(k)}(u,\alpha) \colon \mathbb{S}^{D^\omega} \to \mathbb{S}$ is the quantifier of $\prod_i Q_{i+k}$. The case we are interested in is $k = 0$ because $A^{(0)} = A$ by construction.

We first show that, for every k and every $p \in (D^\omega \to \mathbb{S})$, if $p(s) = \top$ for all $s \in \prod_i Q_{i+k}$, then $A^{(k)}(u,\alpha)(p) = \top$.

We proceed by induction on a suitably defined uniform modulus of continuity $m(p,k)$ of p with respect to k (and with respect to the above fixed data). For each $s \in \prod_i Q_{i+k}$ we have $p(s) = \top$ by assumption and hence there is some finite $t_s \sqsubseteq s$ such that already $p(t_s) = \top$ by continuity of p. Since the sets $\uparrow t_s$ form an open cover of $\prod_i Q_{i+k}$, we conclude by the classical Tychonoff theorem that finitely many of them already cover $\prod_i Q_{i+k}$, say $\uparrow t_{s_1}, \dots, \uparrow t_{s_l}$. Now, if t is finite, there is a smallest n such that $t_{n'} = \bot$ for all $n' \geq n$. Call this the size of t, and let $m(p,k)$ denote the maximum of the sizes of $t_{s_1}, \dots, t_{s_l}$.

By construction, if $m(p,k) = 0$ then $p(\bot) = \top$, and an easy verification shows that if $m(p,k) = n+1$ then $m(\lambda s.p(\text{cons}(x,s)), k+1) = n$ for any x in Q_k. These are the only two properties of the uniform modulus of continuity that we use.

To conclude the first part of the proof, we show by induction on m that the equation $A_{m+1}^{(k)}(u,\alpha)(p) = \top$ holds for every k and every predicate p with $p(s) = \top$ for all $s \in \prod_i Q_{i+k}$ and with $m = m(p,k)$.

Base case: If $m(p,k) = 0$ then

$$A_1^{(k)}(u,\alpha)(p) = \alpha_k(\lambda x.p(\text{if } \bot \text{ then } u^{(k)})) = \alpha_k(\lambda x.p(\bot)) = \alpha_k(\lambda x.\top) = \top,$$

as required

Induction step: If $m(p,k) = m+1$ then $m(\lambda s.p(\mathrm{cons}(x,s)), k+1) = m$. By the induction hypothesis, for any $x \in Q_k$ we have that

$$A^{(k+1)}_{m+1}(u,\alpha)(p) = (\lambda s.p(\mathrm{cons}(x,s))) = \top$$

and hence

$$A^{(k)}_{m+2}(u,\alpha)(p) = \alpha_k(\lambda x.p(\mathrm{if}\,\top\,\mathrm{then}\,u^{(k)})) = \alpha_k(\lambda x.p(u^{(k)})) = \alpha_k(\lambda x.\top) = \top,$$

because α_k quantifies over Q_k by assumption, as required. This concludes the first part of the proof.

For the second and last part of the proof, we show that for every k and p, if $A^{(k)}(u,\alpha)(p) = \top$ then $p(s) = \top$ for all $s \in \prod_i Q_{i+k}$. The premise is equivalent to saying that there is an n with $A^{(k)}_n(u,\alpha)(p) = \top$. We proceed by induction on such n.

Base case: The claim holds vacuously by definition of $A^{(k)}_0$.

Induction step: Assume that

$$A^{(k)}_{n+1}(u,\alpha)(p) = \alpha_k(\lambda x.p(\mathrm{if}\,A^{(k+1)}_n(u,\alpha)(\lambda s.p(\mathrm{cons}(x,s)))\,\mathrm{then}\,u^{(k)})) = \top$$

Then

$$p(\mathrm{if}\,A^{(k+1)}_n(u,\alpha)(\lambda s.p(\mathrm{cons}(x,s)))\,\mathrm{then}\,u^{(k)}) = \top,$$

for every $x \in Q_k$ because α_k quantifies over Q_k by assumption. Fix an arbitrary $x \in Q_k$. If

$$A^{(k+1)}_n(u,\alpha)(\lambda s.p(\mathrm{cons}(x,s)) = \bot$$

then

$$\mathrm{if}\,A^{(k+1)}_n(u,\alpha)(\lambda s.p(\mathrm{cons}(x,s))\,\mathrm{then}\,u^{(k)} = \bot$$

and hence $p(\bot) = \top$ which gives $p(\mathrm{cons}(x,s)) = \top$ for every for every $s \in \prod_i Q_{i+k+1}$. Otherwise,

$$A^{(k+1)}_n(u,\alpha)(\lambda s.p(\mathrm{cons}(x,s)) = \top$$

and hence the induction hypothesis gives

$$\lambda s.p(\mathrm{cons}(x,s)) = \top$$

and we again conclude that $p(\mathrm{cons}(x,s)) = \top$ for every $s \in \prod_i Q_{i+k+1}$. Because $x \in Q_k$ was arbitrary, we conclude that $p(t) = \top$ for every $t \in \prod_i Q_{i+k}$, as required.

If the parallel disjunction operation on the Sierpinski space is available, we don't need to be given the arbitrary choice:

```
ptych :: Seq (Quant a) -> (Quant (Seq a))
ptych(qs)(p) = tryfrom(0)
 where tryfrom(n) = f(n,qs)(p) \/ tryfrom(n+1)
       f(0,qs)(p)=p(bot)
       f(n+1,qs)(p)=q(\x -> f(n,qs')(\s -> p(cons(x,s))))
         where q = hd(qs)
               qs' = tl(qs)
```

We refer to this as the *parallel Tychonoff program*. A similar correctness proof using uniform moduli of continuity is left to the reader. We don't now how to remove the choice without using the parallel operation or whether this is possible.

The above development can be summarized as follows:

13.1 THEOREM (EFFECTIVE TYCHONOFF) *A product of an r.e. sequence of computationally compact spaces is computationally compact.*

13.2 Universal quantification for boolean-valued predicates

We claimed in Chapter 3.13 that integer-valued continuous functions on the Cantor space have decidable equality. To prove this claim, we first consider universal quantification for boolean-valued predicates. A program for this task was discovered by Berger [13, 15]:

```
forall_C :: (Baire -> Bool) -> Bool
```

We are concerned with predicates $p \in (\texttt{Baire} \to \texttt{Bool})$ such that for all $s \in C$, where $C \subseteq \texttt{Baire}$ is the Cantor space, $p(s) \neq \texttt{bot}$. We don't care what happens outside the Cantor space. Using the conventions of Chapter 3.13, the space of such predicates is denoted by $(C \to T)$, where $T = \{\texttt{True}, \texttt{False}\} \subseteq \texttt{Bool}$ is the subspace of booleans.

The specification is that, for $p \in (C \to T)$, $\texttt{forall\_C}(p) = \texttt{True}$ if $p(s) = \texttt{True}$ for all $s \in C$ and $\texttt{forall\_C}(p) = \texttt{False}$ otherwise. We first define

```
epsilon_C :: (Baire -> Bool) -> Baire
```

with the specification that $\texttt{epsilon\_C}(p)$ is in C for any $p \in (C \to T)$, and $p(\texttt{epsilon\_C}(p)) = \texttt{True}$ if there is some $s \in C$ with $p(s) = \texttt{True}$. It follows that

```
exists_C :: (Baire -> Bool) -> Bool
exists_C(p) = p(epsilon_C(p))
```

gives rise to a function such that `exists_C`(p) = `True` if $p(s)$ = `True` for some $s \in C$ and `exists_C`(p) = `False` if $p(s)$ = `False` for all s in the Cantor space. Hence the desired universal quantification functional can be defined by

```
forall_C(p) = not(exists_C(\s -> not(p(s))))
```

The technique to define the ϵ-operator is the same as the one we have used in Chapter 3 to define the quantification functional for Sierpinski-valued predicates: We imagine the predicate as a binarily branching tree, and we recursively try the left and right branches, starting from the root. In fact, the readers can check that the function that we implement satisfies the stronger requirement that, for any predicate $p \in (C \to T)$, `epsilon_C`(p) is the infimum of the set $\{s \in C \mid p(s) = \texttt{True}\}$ in the lexicographic order of the Cantor space, where of course the infimum of the empty set is the maximum point of C, namely the constantly 1 sequence:

```
epsilon_C :: (Baire -> Bool) -> Baire
epsilon_C(p) = if p(l) then l else r
   where l = cons(0,epsilon_C(\s -> p(cons(0,s))))
         r = cons(1,epsilon_C(\s -> p(cons(1,s))))
```

To prove that this works, we proceed in the same way as in Chapter 3, by induction on the uniform modulus of continuity of p, which exists for p in $(C \to T)$. We omit the details, referring the reader to [13] or [15]. Interested readers can amuse themselves running some tests such as

```
forall_C(\s -> exists_C(\t -> s(t(0)+t(1)) == t(s(1)+s(2))))
exists_C(\s -> forall_C(\t -> s(t(0)+t(1)) == t(s(1)+s(2))))
```

and observing the answers `True` and `False` in finite times despite the fact that the Cantor space is uncountable.

13.3 Decidability of equality for integer-valued functions on the Cantor space

Of course, it follows that the space $(C \to Z)$ of (`Baire` $\to$ `Int`) is discrete, where Z is the subspace of non-divergent integers:

```
equal_CtoT :: ((Baire -> Int), (Baire -> Int)) -> Bool
equal_CtoT(p,q) = forall_C(\s -> p(s) == q(s))
```

Cf. Chapter 3.13. This may be surprising at first sight, but should become a triviality after we realize that it is possible to algorithmically construct the finite tree that represents a function in the space $(C \to Z)$.

13.4 The tree of an integer-valued function on the Cantor space

We have alluded to the representing trees of predicates a number of times. The binary tree of an integer-valued function on the Cantor space can be algorithmically constructed as follows. Firstly, one defines the data type of binary trees:

```
data Tree = Leaf Int | Branch Tree Tree
            deriving (Show,Eq)
```

(The interpretation of this data type in the Scott model is the canonical solution of a domain equation, a subject that we haven't touched in these notes — see e.g. [111], [123], [3] or [55]). The *deriving* directive instructs the language processor to create a method for writing down trees (when we want to output them) and another to define equality for those trees which are finite and don't have bottom branches or bottom values at the leaves. A program for converting a predicate into its representing tree is

```
isconstant_C :: (Baire -> Int) -> Bool
isconstant_C(p) =
        forall_C(\s -> forall_C(\t -> p(s) == p(t)))

tree_C :: (Baire -> Int) -> Tree
tree_C(p) = if isconstant_C(p)
            then Leaf (p(c))
            else Branch (tree_C(\s -> p(cons(0,s))))
                        (tree_C(\s -> p(cons(1,s))))
```

Recall that c is an arbitrary, definable element of the Cantor space (Section 13.1).

The produced tree is finite for $p \in (C \to Z)$. Of course, one can easily recover such a predicate from its tree:

```
pred_C :: Tree -> (Baire -> Int)
pred_C(Leaf n) = \s -> n
pred_C(Branch l r) = \s -> if hd(s) == 0
                           then pred_C(l)(tl(s))
                           else pred_C(r)(tl(s))
```

However, we don't recover the same function. We recover an equivalent one in the space $(C \to Z)$, in the sense of Chapter 3.13. If we do this again, however, we do get the same function. That is, we have a retraction [112] that picks canonical representatives of equivalence classes.

13.5 The supremum of the values of a function

We finish with two programs by Simpson [118], the first of which is is related to Proposition 9.13. It computes the supremum of the values of a function $f \in (C \to C)$ in the lexicographic order of C:

```
sup_C :: (Baire -> Baire) -> Baire
sup_C(f) = let d = hd(f(c)) in
             if forall_C(\s -> hd(f(s)) == d)
             then cons(d,sup_C(\s -> tl(f(s))))
             else maxlex(sup_C(\s -> f(cons(0,s))),
                         sup_C(\s -> f(cons(1,s))))

maxlex :: (Baire,Baire) -> Baire
maxlex(s,t) =  if hd(s) < hd(t) then t
           else if hd(s) > hd(t) then s
           else cons(hd(s),maxlex(tl(s),tl(t)))
```

Recall again that c is an arbitrary, definable element of the Cantor space. This relies on the equations

$$\sup(\mathrm{cons}_d \circ f) = \mathrm{cons}(d, \sup f)$$
$$\sup f = \max(\sup(f \circ \mathrm{cons}_0), \sup(f \circ \mathrm{cons}_1)),$$

where $\mathrm{cons}_d(s) = \mathrm{cons}(d, s)$. These equations, in the case of continuous functions $f : [0, 1] \to \mathbb{R}$, with $\mathrm{cons}_d(x) = (d + x)/2$, were previously considered by Edalat and Escardó [33] in order to define a related, but different algorithm for computing suprema of functions defined on a compact interval of real numbers with values on the real numbers via the interval-domain approach discussed in Chapter 6.4.

13.6 Definite integration

Replacing the maximum operator by the average operator $x \oplus y = (x+y)/2$, the Riemann integration functional satisfies analogous equations,

$$\int_0^1 \mathrm{cons}_d \circ f = \mathrm{cons}(d, \int_0^1 f)$$
$$\int_0^1 f = \int_0^1 f \circ \mathrm{cons}_0 \oplus \int_0^1 f \circ \mathrm{cons}_1,$$

which were also used by Edalat and Escardó in order to compute integrals using the interval-domain approach. Using signed-digit binary expansions as in Chapter 6.3, Simpson [118] developed functional programs for computing suprema and definite integrals, based on these equations and the above use of Berger's quantification functional.

13.7 Notes

The Tychonoff program presented above was discovered during the Barbados meeting for which the first version of this set of notes was prepared, just in time to be presented in the last lecture. We don't know whether the effective Tychonoff Theorem 13.1 has been formulated or proved before.

Bibliography

[1] Abramsky, S., *Domain theory in logical form*, Ann. Pure Appl. Logic **51** (1991), pp. 1–77. 4, 1

[2] Abramsky, S., R. Jagadeesan and P. Malacaria, *Full abstraction for PCF*, Inform. and Comput. **163** (2000), pp. 409–470. 4.3

[3] Abramsky, S. and A. Jung, *Domain theory*, in: S. Abramsky, D. Gabbay and T. Maibaum, editors, *Handbook of Logic in Computer Science*, Oxford science publications **3**, Clarendon Press, Oxford, 1994 pp. 1–168. 6, 12.7, 13.4

[4] Amadio, R. and P.-L. Curien, "Domains and Lambda-Calculi," CUP, 1998. 7

[5] Awodey, S., L. Birkedal and D. Scott, *Local realizability toposes and a modal logic for computability*, Math. Struct. Comput. Sci. **12** (2002), pp. 319–334.

[6] Bauer, A., *A relationship between equilogical spaces and type two effectivity*, MLQ Math. Log. Q. **48** (2002), pp. 1–15.

[7] Bauer, A., *Equilogical spaces as imaginary spaces* (2003), university of Ljubljana. Presented at the *Workshop on Domains, Topology and Constructive Logic*, LMU, Department of Mathematics, University of Munich, 1-2 November 2003. 10.2

[8] Bauer, A. and L. Birkedal, *Continuous functionals of dependent types and equilogical spaces*, Lec. Not. Comput. Sci. **1862** (2000), pp. 202–216.

[9] Bauer, A., L. Birkedal and D. Scott, *Equilogical spaces*, Theoret. Comput. Sci. . 10.2

[10] Bauer, A., M. Escardó and A. Simpson, *Comparing functional paradigms for exact real-number computation*, Lect. Not. Comp. Sci. **2380**, 2002, pp. 488–500. 6.4

[11] Bauer, A. and D. Scott, *A new category for semantics* (2001), notes from D.S. Scott's talk at MFCS 2001. CMU, available from Scott's web page. 3.14

[12] Beeson, M., "Foundations of Constructive Mathematics," Springer, New York, 1985. 3.11

[13] Berger, U., "Totale Objekte und Mengen in der Bereichstheorie," Ph.D. thesis, Mathematisches Institut der Universität München (1990). 3.14, 13.2

[14] Berger, U., *Total sets and objects in domain theory*, Ann. Pure Appl. Logic **60** (1993), pp. 91–117.

[15] Berger, U. and P. Oliva, *Modified bar recursion and classical dependent choice* (to appear). 13.2

[16] Bird, R. and P. Wadler, "Introduction to Functional Programming," Prentice-Hall, New York, 1988. 2.1

[17] Bourbaki, N., "General Topology," Addison-Wesley, London, 1966. 5

[18] Brattka, V., *Computability over topological structures*, in: S. Cooper and S. Goncharov, editors, *Computability and Models*, Kluwer Academic, 2003 pp. 93–136.

[19] Brattka, V. and P. Hertling, *Topological properties of real number representations*, Theoret. Comput. Sci. **284** (2002), pp. 241–257.

[20] Brouwer, L., *Besitzt jede reelle Zahl eine Dezimalbruchentwicklung?*, Math. Ann. **83** (1920), pp. 201–210.

[21] Brown, R., "Topology," Ellis Horwood Ltd., Chichester, 1988, second edition.

[22] Coquand, T., S. Sadocco, G. Sambin and J. Smith, *Formal topologies on the set of first-order formulae*, J. Symbolic Logic **65** (2000), pp. 1183–1192. 3.14

[23] Crole, R., "Categories for Types," Cambridge University Press, Cambridge, 1993. 8.6

[24] Davey, B. A. and H. A. Priestley, "Introduction to lattices and order," Cambridge University Press, New York, 2002, second edition, xii+298 pp. 2

[25] Day, B. J. and G. M. Kelly, *On topological quotient maps preserved by pullbacks or products*, Proc. Cambridge Philos. Soc. **67** (1970), pp. 553–558.

[26] DeJaeger, F., *An approach to effective functionals on the real numbers via filter spaces*, Topology Proceedings **26** (2001–2002), pp. 485–504.

[27] DeJaeger, F., M. Escardó and G. Santini, *On the computational content of the Lawson topology*, presented at MFPS XVI, available at Escardó's web page. 9.14

[28] Di Gianantonio, P., *Real number computability and domain theory*, Inform. and Comput. **127** (1996), pp. 11–25.

[29] Dubuc, E., *Concrete quasitopoi*, Lect. Notes Math. **753**, 1979 pp. 239–254. 10.2

[30] Dubuc, E. and J. Penon, *Objets compacts dans les topos*, J. Austral. Math. Soc. Ser. A **40** (1986), pp. 203–217. 3.14

[31] Dugundji, J., "Topology," Allin and Bacon, Inc., Boston, 1966. 5

[32] Edalat, A., *Domains for computation in mathematics, physics and exact real arithmetic*, Bulletin of Symbolic Logic **3** (1997), pp. 401–452. 6.5

[33] Edalat, A. and M. Escardó, *Integration in Real PCF*, in: *Proceedings of the Eleventh Annual IEEE Symposium on Logic In Computer Science*, New Brunswick, New Jersey, USA, 1996, pp. 382–393. 13.5

[34] Edalat, A. and M. Escardó, *Integration in Real PCF*, Inform. and Comput. **160** (2000), pp. 128–166.

[35] Egli, H. and R. Constable, *Computability concepts for programming languages*, Theoret. Comput. Sci. **2** (1976), pp. 133–145. 12.7

[36] Eilenberg, S., *Cartesian spaces and local compactess* (1985), columbia University, unpublished manuscript. 8.6

[37] Erker, T., M. Escardó and K. Keimel, *The way-below relation of function spaces over semantic domains*, Topology Appl. **89** (1998), pp. 61–74.

[38] Ershov, Y., *Computable functionals of finite types*, Algebra Logic **11** (1972), pp. 203–242.

[39] Ershov, Y., *Continuous lattices and A-spaces*, Soviet Mathematics Doklady **13** (1973), pp. 1551–1555.

[40] Escardó, M., *PCF extended with real numbers*, Theoret. Comput. Sci. **162** (1996), pp. 79–115.

[41] Escardó, M., *Real PCF extended with $\exists$ is universal*, in: A. Edalat, S. Jourdan and G. McCusker, editors, *Advances in Theory and Formal Methods of Computing: Proceedings of the Third Imperial College Workshop, April 1996* (1996), pp. 13–24.

[42] Escardó, M., *Injective spaces via the filter monad*, Topology Proceedings **22** (1997), pp. 97–110.

[43] Escardó, M., *PCF extended with real numbers: A domain-theoretic approach to higher-order exact real number computation*, Technical Report ECS-LFCS-97-374, University of Edinburgh (1997), phD thesis at Imperial College. 6.5

[44] Escardó, M., *Effective and sequential definition by cases on the reals via infinite signed-digit numerals*, Electron. Notes Theor. Comput. Sci. **13** (1998). 6.3

[45] Escardó, M., *A metric model of PCF* (1998), LFCS, University of Edinburgh. Presented at the Workshop on Realizability Semantics and Applications, Federated Logic Conference, Trento, June 29–July 12, 1999. Available at the author's web page.

[46] Escardó, M., *Properly injective spaces and function spaces*, Topology Appl. **89** (1998), pp. 75–120. 12.16

[47] Escardó, M., *Function-space compactifications of function spaces*, Topology Appl. **120** (2002), pp. 441–463.

[48] Escardó, M., *Mathematical foundations of functional programming with real numbers* (2003), university of Birmingham, lecture notes for a course in the Midlands Graduate School, available at the author's web page.

[49] Escardó, M. and R. Flagg, *Semantic domains, injective spaces and monads*, Electron. Notes Theor. Comput. Sci. **20** (1999).

[50] Escardó, M. and R. Heckmann, *Topologies on spaces of continuous functions*, Topology Proceedings **26** (2001–2002), pp. 545–564. 8, 9.6

[51] Escardó, M., M. Hofmann and T. Streicher, *On the non-sequential nature of the interval-domain model of exact real-number computation*, Math. Struct. Comput. Sci. (to appear).

[52] Escardó, M., J. Lawson and A. Simpson, *Comparing categories of (core) compactly generated spaces*, Topology Appl. (To appear). 10.2

[53] Escardó, M. and T. Streicher, *Induction and recursion on the partial real line with applications to Real PCF*, Theoret. Comput. Sci. **210** (1999), pp. 121–157.

[54] Gierz, G., K. Hofmann, K. Keimel, J. Lawson, M. Mislove and D. Scott, "A Compendium of Continuous Lattices," Springer, 1980. 3, 11.2, 12.9, 12.16

[55] Gierz, G., K. Hofmann, K. Keimel, J. Lawson, M. Mislove and D. Scott, "Continuous Lattices and Domains," Cambridge University Press, 2003. 3, 11.2, 12.6, 12.7, 12.9, 12.16, 13.4

[56] Gordon, A. D., *Bisimilarity as a theory of functional programming*, Theoret. Comput. Sci. **228** (1999), pp. 5–47, mathematical foundations of programming semantics (New Orleans, LA, 1995). 2.8

[57] Gunter, C. A., "Semantics of Programming Languages—Structures and Techniques," The MIT Press, London, 1992. 2.8

[58] Heckmann, R., *Power domains and second-order predicates*, Theoret. Comput. Sci. **111** (1993), pp. 59–88. 11.5

[59] Heckmann, R., *A non-topological view of dcpo's as convergence spaces*, Theoret. Comp. Sci. **305** (2003), pp. 159–186.

[60] Heckmann, R. and M. Huth, *Quantitative semantics, topology, and possibility measures*, Topology Appl. **89** (1998), pp. 151–178.

[61] Hennessy, M. and R. Milner, *Algebraic laws for nondeterminism and concurrency*, J. Assoc. Comput. Mach. **32** (1985), pp. 137–161.

[62] Hofmann, K. and J. Lawson, *The spectral theory of distributive continuous lattices*, Trans. Amer. Math. Soc. **246** (1978), pp. 285–310. 3.14

[63] Hofmann, K. and M. Mislove, *Local compactness and continuous lattices*, in: *Continuous Lattices*, Lect. Notes Math. **871**, 1981, pp. 209–248. 11.3

[64] Hyland, J., *Filter spaces and continuous functionals*, Ann. Math. Logic **16** (1979), pp. 101–143. 10.2, 10.2

[65] Hyland, J. M. E., *First steps in synthetic domain theory*, in: *Category theory (Como, 1990)*, Lecture Notes in Math. **1488**, Springer, Berlin, 1991 pp. 131–156. 0.5

[66] Hyland, J. M. E. and C.-H. L. Ong, *On full abstraction for PCF: I, II and III*, Inform. and Comput. **163** (2000), pp. 285–408. 4.3

[67] Hyland, M., *Function spaces in the category of locales.*, in: *Continuous lattices*, Lect. Notes Math. **871**, 1981, pp. 264–281. 9.9

[68] Isbell, J., *General function spaces, products and continuous lattices*, Math. Proc. Camb. Phil. Soc. **100** (1986), pp. 193–205. 8.6, 12.16

[69] Johnstone, P., "Stone Spaces," Cambridge University Press, Cambridge, 1982. 4, 11.2, 12.16

[70] Johnstone, P., *Open locales and exponentiation*, in: *Mathematical applications of category theory (Denver, Col., 1983)*, Amer. Math. Soc., Providence, RI, 1984 pp. 84–116. 3.14, 9.1

[71] Johnstone, P., *Vietoris locales and localic semilattices*, in: *Continuous lattices and their applications (Bremen, 1982)*, Dekker, New York, 1985 pp. 155–180.

[72] Johnstone, P., "Sketches of an Elephant: a Topos Theory Compendium," Oxford University Press, 2002. 3.14, 8.6

[73] Jones, M. et al., *hugs online*, . 2.1

[74] Jung, A., *Stably compact spaces and the probabilistic powerspace construction*, in: J. Desharnais and P. Panangaden, editors, *Domain-theoretic Methods in Probabilistic Processes*, Electron. Notes Theoret. Comp. Sci. **87** (2004), 15pp.

[75] Kahn, G., *The semantics of a simple language for parallel programming*, in: *Information processing 74 (Proc. IFIP Congress, Stockholm, 1974)*, North-Holland, 1974 pp. 471–475.

[76] Keimel, K. and G. Gierz, *Halbstetige Funktionen und stetige Verbände*, in: R.-E. Hoffmann, editor, *Continuous Lattices and Related Topics*, Mathematik-Arbeitspapiere **27**, Universität Bremen, 1982, pp. 59–67. 12.16

[77] Keimel, K. and J. Paseka, *A direct proof of the Hofmann-Mislove theorem*, Proc. Amer. Math. Soc. **120** (1994), pp. 301–303. 11.3

[78] Kelley, J., "General Topology," D. van Nostrand, New York, 1955. 5

[79] Kleene, S., "Introduction to Metamathematics," North-Holland, 1952.

[80] Kleene, S., *Countable functionals*, in: *Constructivity in mathematics: Proceedings of the colloquium held at Amsterdam, 1957 (edited by A. Heyting)*, Studies in Logic and the Foundations of Mathematics (1959), pp. 81–100.

[81] Ko, K.-I., "Complexitity Theory of Real Functions," Birkhauser, Boston, 1991.

[82] Kock, A., "Synthetic differential geometry," London Mathematical Society Lecture Note Series **51**, Cambridge University Press, Cambridge, 1981. 0.5

[83] Kreisel, G., *Interpretation of analysis by means of constructive functionals of finite types*, in: *Constructivity in mathematics: Proceedings of the colloquium held at Amsterdam, 1957 (edited by A. Heyting)*, Studies in Logic and the Foundations of Mathematics (1959), pp. 101–128.

[84] Lambek, J. and P. Scott, "Introduction to Higher Order Categorical Logic," CUP, 1986. 8.6

[85] Lawson, J., *Spaces of maximal points*, Math. Struct. Comput. Sci. **7** (1997), pp. 543–555.

[86] Longley, J., *When is a functional program not a functional program?*, in: *Proc. 4th International Conference on Functional Programming, Paris* (1999), pp. 1–7. 3.11, 4.3

[87] Longley, J., *The sequentially realizable functionals*, Ann. Pure Appl. Logic **117** (2002), pp. 1–93. 4.3

[88] Longo, G., *Some topologies for computations*, in: *Proceedings of Géométrie au XX siècle, 1930 - 2000, Paris* .

[89] Mac Lane, S., "Categories for the Working Mathematician," Springer, 1971. 6

[90] McLarty, C., "Elementary Categories, Elementary Toposes," Clarendon Press, Oxford, 1992. 8.6

[91] Menni, M. and A. Simpson, *Topological and limit-space subcategories of countably-based equilogical spaces*, Math. Struct. Comput. Sci. **12** (2002), pp. 739–770.

[92] Milner, R., *Fully abstract models of typed λ-calculi*, Theoret. Comput. Sci. **4** (1977), pp. 1–22.

[93] Mislove, M., *Topology, domain theory and theoretical computer science*, Topology Appl. **89** (1998), pp. 3–59.

[94] Myhill, J. and J. C. Shepherdson, *Effective operations on partial recursive functions*, Z. Math. Logik Grundlagen Math. **1** (1955), pp. 310–317.

[95] Nerode, A., *Some Stone spaces and recursion theory*, Duke Math. J. **26** (1959), pp. 397–406.

[96] Normann, D., "Recursion on the countable functionals," Lec. Not. Math. **811**, Springer, Berlin, 1980, viii+191 pp.

[97] Normann, D., *Computability over the partial continuous functionals*, J. Symbolic Logic **65** (2000), pp. 1133–1142.

[98] Normann, D., *Hierarchies of total functionals over the reals*, Theoret. Comput. Sci. (to appear). 6.4

[99] Paulson, L., "ML for the working programmer," Cambridge University Press, Cambridge, 1991. 2.1

[100] Pitts, A., *Operationally-based theories of program equivalence*, in: *Semantics and logics of computation (Cambridge, 1995)*, Publ. Newton Inst. **14**, Cambridge Univ. Press, Cambridge, 1997 pp. 241–298. 2.8

[101] Plotkin, G., *LCF considered as a programming language*, Theoret. Comput. Sci. **5** (1977), pp. 223–255. 2, 2.8, 4.2, 12.14, 12.15

[102] Plotkin, G., $\mathbb{T}^\omega$ *as a universal domain*, J. Comput. System Sci. **17** (1978), pp. 209–236. 12.1

[103] Plotkin, G., *Pisa notes on domains* (1983), department of Computer Science, University of Edinburgh. Available at the author's web page. 2, 2.8, 12.7

[104] Plotkin, G., *Ignorance and uncertainty* (2003), lectures given at McGill University's Bellairs Research Institute, Barbados. 3.14

[105] Poigné, A., *Basic category theory*, in: *Handbook of logic in computer science, Vol. 1*, Oxford Univ. Press, New York, 1992 pp. 413–640. 6

[106] Pour-El, M. and J. Richards, "Computability in analysis and physics," Perspectives in Mathematical Logic, Springer, Berlin, 1989, xii+206 pp.

[107] Robinson, E., *Logical aspects of denotational semantics*, Lec. Not. Comput. Sci. **283**, 1987 pp. 238–253.

[108] Rogers, H., "Theory of Recursive Functions and Effective Computability," McGraw-Hill, New York, 1967. 1, 3.12, 4.1, 4.3

[109] Rosolini, G., *Equilogical spaces and filter spaces*, Rend. Circ. Mat. Palermo (2) Suppl. **64** (2000), pp. 157–175, categorical studies in Italy (Perugia, 1997). 10.2, 10.2

[110] Rosolini, G. and T. Streicher, *Comparing models of higher type computation*, Electron. not. Comput. Sci. **23** (1999), p. 7. 10.2

[111] Scott, D., *Continuous lattices*, Lec. Not. Math. **274**, 1972, pp. 97–136. 1, 12.16, 13.4

[112] Scott, D., *Data types as lattices*, SIAM J. Comput. **5** (1976), pp. 522–587. 1, 13.4

[113] Scott, D., *A type-theoretical alternative to CUCH, ISWIM and OWHY*, Theoret. Comput. Sci. **121** (1993), pp. 411–440, reprint of a 1969 manuscript. 1, 4.2, 12.15

[114] Scott, D., *A new category? Domains, spaces and equivalence relations* (1996), computer Science Department, Carnagie Mellon University. Available at the author's web page. 3.13

[115] Scott, D., *Effective versions of equilogical spaces*, Electron. Notes Theor. Comput. Sci. **35** (2000).

[116] Scott, D., *A new category for semantics*, Lec. Not. Comput. Sci. **2136** (2001), pp. 1–2.

[117] Simmons, G., "Introduction to Topology and Modern Analysis," McGraw-Hill, New York, 1963. 5

[118] Simpson, A., *Lazy functional algorithms for exact real functionals*, Lec. Not. Comput. Sci. **1450** (1999), pp. 323–342. 13.5, 13.6

[119] Simpson, A., *Towards a convenient category of topological domains*, in: *Proceedings of 13th ALGI Workshop, RIMS*, Kyoto University, 2003.

[120] Smyth, M., *Effectively given domains*, Theoret. Comput. Sci. **5** (1977), pp. 256–274. 6.4, 12.7

[121] Smyth, M., *Power domains and predicate transformers: a topological view*, Lec. Not. Comput. Sci. **154**, 1983, pp. 662–675. 3, 1

[122] Smyth, M., *Topology*, in: S. Abramsky, D. Gabbay and T. Maibaum, editors, *Handbook of Logic in Computer Science*, Oxford science publications **1**, Clarendon Press, Oxford, 1992 pp. 641–761. 3, 1, 6.1

[123] Smyth, M. and G. Plotkin, *The category-theoretic solution of recursive domain equations*, SIAM J. Comput. **11** (1982), pp. 761–783. 13.4

[124] Stoltenberg-Hansen, V., I. Lindström and E. Griffor, "Mathematical theory of domains," CUP, 1994, xii+349 pp.

[125] Stoltenberg-Hansen, V. and J. Tucker, *Concrete models of computation for topological algebras*, Theoret. Comput. Sci. **219** (1999), pp. 347–378.

[126] Stone, M. H., *The theory of representations for Boolean algebras*, Trans. Amer. Math. Soc. **40** (1936), pp. 37–111.

[127] Stone, M. H., *Applications of the theory of Boolean rings to general topology*, Trans. Amer. Math. Soc. **41** (1937), pp. 375–481.

[128] Streicher, T., *Mathematical foundations of functional programming* (2003), department of mathematics, University of Darmstadt. Available at the author's web page. 8, 12.15

[129] Sutherland, W. A., "Introduction to metric and topological spaces," Clarendon Press, Oxford, 1975, xiii+181 pp. 5, 9.6

[130] Taylor, P., "Practical foundations of mathematics," CUP, 1999, xii+572 pp.

[131] Taylor, P., *Geometric and higher order logic in terms of abstract Stone duality*, Theory Appl. Categ. **7** (2000), pp. No. 15, 284–338. 3.14

[132] Taylor, P., *Local compactness and the Baire category theorem in ASD: a feasibility study*, Electron. Notes Theor. Comput. Sci. **69** (2002). 11.5

[133] Taylor, P., *Sober spaces and continuations*, Theory Appl. Categ. **10** (2002), pp. No. 12, 248–300. 3.14, 11.5

[134] Taylor, P., *Subspaces in abstract Stone duality*, Theory Appl. Categ. **10** (2002), pp. No. 13, 301–368.

[135] Tennent, R. D., *Denotational semantics*, in: *Handbook of logic in computer science, Vol. 3*, Oxford Sci. Publ., Oxford Univ. Press, New York, 1994 pp. 169–322. 2.8

[136] Townsend, C. and S. Vickers, *A universal characterization of the double power locale*, Theoret. Comput. Sci. (to appear). 9.1, 11.5

[137] Tsuiki, H., *Computational dimension of topological spaces*, Lec. Not. Comput. Sci. **2064** (2001), pp. 323–335.

[138] Turing, A., *A correction.*, Proc. Lond. Math. Soc., II. Ser. **43** (1936), pp. 544–546. 6.3

[139] Turing, A., *On computable numbers, with an application to the Entscheidungsproblem.*, Proc. Lond. Math. Soc., II. Ser. **42** (1936), pp. 230–265. 6.3

[140] Vickers, S., "Topology via Logic," CUP, 1989. 5, 1, 9

[141] Vickers, S., *The double powerlocale and exponentiation: a case study in geometric logic* (2001), available at the author's web page. 9.1

[142] Weihrauch, K., "Computable analysis," Springer, 2000, x+285 pp. 9, 2.8, 6.3

[143] Weihrauch, K. and C. Kreitz, *Representations of the real numbers and the open subsets of the set of real numbers*, Ann. Pure and Appl. Logic **35** (1987), pp. 247–260. 6.3

[144] Wiedmer, E., *Computing with infinite objects*, Theoret. Comput. Sci. **10** (1980), pp. 133–155. 6.3

[145] Winskel, G., "The formal semantics of programming languages," Foundations of Computing Series, MIT Press, Cambridge, MA, 1993, xx+361 pp., an introduction. 2.8

Index