# RSMG Progress Report #4
# Thesis Proposal

*Draft of February 16, 2022*

Ayberk Tosun
a.tosun@pgr.bham.ac.uk

---

Working Title
*Exhaustible Sets and C-spaces*

| Date | | Thesis Group | |
|---|---|---|---|
| | | Supervisor | Martín H. Escardó |
| Report date | 19 January 2022 | Second supervisor | Vincent Rahli |
| Meeting date | 15 February 2022 | RSMG rep. | Steven J. Vickers |
| | | Member | Paul B. Levy |

---

# Contents


**1 Introduction** **1**
- 1.1 Brief overview of exhaustible and searchable sets . . . 1
- 1.2 Primary approach: constructive reasoning with $C$-spaces . . . 2
- 1.3 Alternative approach: constructive reasoning with locales . . . 3
- 1.4 Notes and acknowledgements . . . 4

**2 Background** **6**
- 2.1 Domain theory and Stone Duality . . . 6
- 2.2 Searchable and exhaustible sets . . . 12
- 2.3 Synthetic topology . . . 18
- 2.4 May-must testing in PCF . . . 20
- 2.5 Sheaves and Grothendieck topoi . . . 24
- 2.6 $C$-spaces . . . 27

**3 Proposed Work** **32**
- 3.1 Higher-type computation in the category of $C$-spaces . . . 32
- 3.2 Application to may-must testing in a total setting . . . 33
- 3.3 Category of locales as an alternative? . . . 36

**4 Evaluation of Work** **38**
- 4.1 Evaluation of the development in C-**Space** . . . 38
- 4.2 Evaluation of application to may-must testing . . . 38

**5 Plan and Timetable** **40**

# 1 Introduction

In Progress Report #3, the thesis group meeting for which took place on 22 June 2021, three possible thesis topics were outlined. These topics were:

1. continuation of the predicative development of the patch frame in univalent type theory;
2. implementation of synthetic topology in univalent type theory following the ideas of Escardó [Esc04a], based on the notion of *dominance* (due to Rosolini [Ros86]) as formalised by Knapp and Escardó [Kna18; EK17] in the Agda proof assistant;
3. constructive reformulations of Escardó's [Esc08] results on exhaustible and searchable sets in a domain-theoretic setting.

The thesis work that we propose in this report is essentially a refinement of topic (3) in this list. Instead of using domain theory, however, we plan to use Escardó and Xu's sheaf topos [Xu15; EX16; XE13], constructed to serve as a "nice" category of spaces [nLa] conducive to *constructive* reasoning about computation. Furthermore, we plan to develop these constructive results with a specific application in mind: modelling Gödel's System T augmented with nondeterministic may-must testing modalities, based on Escardó's previous work [Esc09] on the denotational semantics of PCF extended with such modalities.

## 1.1 Brief overview of exhaustible and searchable sets

Escardó's results on exhaustible and searchable sets in the setting of higher-type computation [Esc08; Esc07b] form the basis of the work that we propose. An exhaustible set is one that admits a computable *quantification functional* whereas a searchable set is one that has a computable *search functional*. A "set" here is understood to be a certain subset of inhabitants of a type in a programming language. A (universal) quantification functional on a set takes a decidable predicate (defined on the set) and checks if all terms in the set satisfy the predicate. A search functional, on the other hand, similarly takes a decidable predicate on a set (that is defined on the set) and returns an element of the set satisfying the predicate, *if* such an inhabitant exists; otherwise, it returns a counterexample that does not satisfy the predicate. Sets on which search functionals are defined are therefore required to be nonempty.

It is obvious that searchable sets are exhaustible: one can simply run the search functional and check if the result it returns satisfies the predicate. What about the converse direction? It also holds in the natural setting of higher-type computation, but this direction is nontrivial—even rather surprising. This is a result due to Escardó [Esc08] and is obtained through the importation of topological ideas into computation. More precisely, sets consisting of *total* functionals are topologised using the Kleene-Kreisel topology in [Esc08, Paragraph 2.3] which enables the identification of exhaustible sets of total elements with compact sets (i.e. compact in the Kleene-Kreisel topology). Furthermore, certain closure properties for searchable and exhaustible sets are also given in *loc. cit.* by Escardó. Let us list some of these "methods for building searchable sets". Escardó shows that both exhaustible and searchable sets are closed under:

(**a**) intersections with *decidable* sets [Esc08, Proposition 4.2],

(**b**) the formation of computable images [Esc08, Proposition 4.3], and

(**c**) finite and countably infinite products [Esc08, Proposition 4.4].

These methods also come from the topological interpretation of the notion of exhaustibility that is used in the proof that exhaustible sets are searchable.

Property (**c**) here is the most relevant result for our purposes as it gives rise to a practical implementation of a program that exhaustively searches the Cantor space [Esc07c], called the *Cantor search functional*. Through the interpretation of exhaustibility as compactness, this result can be understood as a computational version of the (countable) Tychonoff Theorem [Tyc30; Čec37], which is one of the fundamental theorems of point-set topology.

From the point of view of constructive mathematics, a conspicuous limitation of Escardó's results from [Esc08] is that the topological ideas employed in showing the correctness of these methods for building searchable and exhaustible sets use classical results from topology.

As a result, no constructive proof of correctness of these methods exist as of writing (personal communication with Escardó). This is the main issue that we intend to address in the proposed work. We would like to develop a *constructive* understanding of these results on searchable and exhaustible sets. To achieve this goal, we consider two approaches in the following two sections.

## 1.2 Primary approach: constructive reasoning with $C$-spaces

Classical mathematics is a viable setting for reasoning about algorithms and computation. One can always maintain a distinction between the algorithms in consideration and their specifications, and carry out classical reasoning for showing that the former satisfy the latter. Ideally, however, one would like to be able to work in the context of Bishop-style constructive mathematics (also called "neutral mathematics"). The motivation for this is two-fold:

1. It is *more general* (in a precise sense): neutral mathematics can be considered the internal language of the free topos [LS86; Tay03; Esc21] and every theorem that holds in the free topos holds in any other topos.

2. By working in the setting of constructive mathematics, we can treat proofs of the theorems in consideration as programs themselves, obviating the need for an artificial distinction between the two. Mathematical constructions formalised in the underlying type theory of proof assistants such Coq and Agda can be *directly* interpreted as programs and we hope to leverage this by working in a constructive setting.

Having clarified our credo for working in a constructive setting, let us now shift our attention towards the question of *how* we plan to achieve this. To attempt to reformulate Escardó's work [Esc08; Esc07b] in a constructive setting, we plan to make use of Escardó and Xu's $C$-spaces. $C$-spaces are defined as analogues of Johnstone's *subsequential spaces* that arise as the concrete sheaves of his topological topos [Joh79] (these concepts are to be presented in detail in Section 2.6). We believe that $C$-spaces are promising for this purpose as the category of $C$-spaces was constructively proven to have a fan functional. Using this fan functional, a continuous model of Gödel's System T was constructed in the category of $C$-spaces [XE13].

In Escardó's previous work on exhaustible sets, *domain theory* is used to define the notions of searchability and exhaustibility and to facilitate the importation of topology into denotational semantics. A definition of a subset of domain consisting of total elements is given and such total subsets are topologised using the Kleene-Kreisel topology. Topology is thereby used

to obtain a definition of the Cantor search functional *for domains* which is then understood as a PCF-definable algorithm. Even though a concrete algorithm for searching the Cantor space can be extracted from this development, the proof of correctness of this algorithm is not constructive (as mentioned in the previous section).

We plan to carry out an analogous development in the category C-**Space** of $C$-spaces. Our development will be simpler as we will be only interested in a total programming language: a suitable extension of System T. Therefore, we will not have to work with a category of domains that is known to be able to account for the possibility of undefinedness in the language in consideration. Our plan is to define notions of searchable and exhaustible subsets in the category C-**Space** of $C$-spaces, and then prove an analogue of the Tychonoff Theorem in C-**Space** (as well as other characterisations of searchable sets if time permits). Through an extension of the model of System T constructed by Escardó and Xu [XE13], we plan to interpret the Tychonoff Theorem as the Cantor search functional in System T, which should allow us to extend Escardó's semidecidability result for may-must testing in PCF to *decidability* results for may-must testing in System T.

More details on our plan to use the category of $C$-spaces for modelling System T augmented with nondeterministic testing constructs are given in Section 3.1 and the relevant background material is presented in detail in Section 2.2.

## 1.3 Alternative approach: constructive reasoning with locales

An alternative approach to obtaining constructive reformulations of Escardó's results could be using the category **Loc** of locales, through Samson Abramsky's computational interpretation of Stone Duality [Abr91] (which in fact goes back to Smyth [Smy83; Smy93]). It is well-known that the category of locales is an alternative to the category of topological spaces that is more suitable for purposes of constructive reasoning (see for instance [Joh83]). A notable example of this is the fact that the Tychonoff Theorem, whose proof in the point-set setting makes use of classical reasoning, has a constructive proof in the pointfree setting of locale theory [Joh81; Coq92; Vic05] (see also [Joh02, pg. 86]).

In his doctoral work [Abr88; Abr91], Samson Abramsky gave an interpretation of Stone Duality (to be presented in detail in Section 2.1) in terms of programming languages. Stone Duality is about points and their open sets and these two sides of the duality are called the *spatial* side and the *logical* side respectively. Abramsky interprets these two sides of the duality as follows:

**The spatial side:** the points of spaces are viewed as denotations of programs;

**The logical side:** open sets of the points are viewed as observable properties of these programs.

This interpretation is in fact due to Smyth [Smy78; Smy93] and the novelty presented in Abramsky's work is the use of Stone Duality as a unifying framework for clarifying the relationship between denotational semantics and program logics. This is achieved through a presentation of the logical side of the duality as a formal system with axioms and inference rules.

In the context of this correspondence between domain theory and locale theory, it could be possible to reason about the PCF-implementation of the Cantor search functional in the

category of locales, talking about the frames of opens of Scott topologies of domains. This is an alternative possible approach to addressing the use of classical reasoning in Escardó's work on exhaustible sets that we intend to consider if time permits.

The reason we have not decided to pursue this as the primary approach is that there are multiple foreseeable problem with working in the category of locales. Unlike the category C-**Space**, the category **Loc** is not cartesian closed and it is therefore not straightforward to work with exponentials of locales. In particular, it is not clear how to work with the notion of *totality* in the context of locale theory. The notion of a subset of a domain consisting of total elements is vital to the (somewhat surprising) existence of the Cantor search functional and it is not clear how to formulate the notion of totality in the category **Loc**. A result in this direction would, however, be certainly interesting and we plan to work on this alternative approach if time permits.

More details on this alternative approach and the foreseeable problems are provided in Section 3.2, and the relevant background material is given in Section 2.1.

## 1.4 Notes and acknowledgements

The idea of this thesis topic was suggested by Martín Escardó (supervisor), shortly after the thesis group meeting on 22 June 2021 in which Progress Report #3 was discussed. The initial motivation sprang from the question of may-must testing in System T and the fact that Escardó's previous work [Esc09] on this has some connections to synthetic topology, which was one of the possible thesis topics discussed in Progress Report #3. At this point, the focus was more on the application to may-must testing rather than constructive reformulations of the results on exhaustible sets.

Through some discussions between Steve Vickers and Martín Escardó regarding the possibility of obtaining constructive proofs about exhaustible sets in the category **Loc** of locales, the third possibility mentioned in Section 1.3 came up. After multiple discussions on this, we have come to the conclusion that there will be some difficulties in a potential thesis on this question due to aforementioned possible problems mentioned in Section 1.3. More details on this are provided in Section 3.3.

Even though we have decided not to pursue this as the primary thesis topic, these discussions with Steve Vickers led us to generalise the work concerning the category of $C$-spaces as to orient the primary goal on developing the ideas of Escardó's work on exhaustible and searchable sets.

Let us also briefly discuss the reasons for not pursuing the continuation of the line of work we have started on implementing locale theory and pointfree topology in univalent type theory as a thesis topic. One of the projects we have completed throughout the first year was the predicative implementation of the patch frame [Esc99] in univalent type theory. Therefore, it would have been natural to pursue the continuation of this project as a thesis research topic. After some discussions about this, however, we have decided to reorient the thesis topic towards a question that would be more directly applicable to computability theory and programming language semantics (i.e. denotational semantics).

The patch frame project itself could in fact be oriented towards applications to denotational semantics as the patch frame exhibits the category of compact regular locales and continuous maps as a coreflective subcategory of the category of stably compact locales and perfect maps, which allows one to view domains as compact Hausdorff spaces and apply classical topology

to domain theory [Esc99, pg. 1]. However, we have not found a concrete thesis topic in the direction of a computational application of this kind, therefore eventually deciding to change the direction of the thesis work. More specifically, it seemed to the author that the work on the patch frame was not heading towards the direction of a computational application which was one of the reasons that instigated the discussion of alternative thesis topics. Nevertheless, the patch frame project is now complete and has been formalised in the Agda proof assistant. A preprint summarising our work on it is soon to be finalised and we intend to submit this for publication.

# 2 Background

In this section, we provide self-contained presentations of the background material required for Section 3 where we outline the proposed work.

## 2.1 Domain theory and Stone Duality

Even though the primary work that we propose in this report is not planned to take place in a domain-theoretic setting, the work of Escardó [Esc09; Esc08] that we plan to build upon makes extensive use of domain theory. Furthermore, the alternative approach based on the category of locales (which we have mentioned in Section 1.3) is directly relevant to domain theory (see Section 3.3 for details). Therefore, we provide a brief explanation of the core ideas of domain theory and their relationship to Stone Duality (as formulated by Abramsky [Abr88; Abr91] in his doctoral work) in this section.

Let us start with the history of the subject[1]. Domain Theory arose from the problem of finding a model for the untyped $\lambda$-calculus. Until the construction of a nontrivial mathematical model by Dana Scott in 1969, all known models of the untyped $\lambda$-calculus had been *term models* constructed from the calculus itself. Scott remarked in an unpublished note from 1969 [Sco69] (unpublished until its publication in 1993 as [Sco93]) that the untyped $\lambda$-calculus should not be used as a tool in programming languages due to this problem that it has no known nontrivial mathematical models, and contended that *typed* systems should be preferred instead.

To make a case for typed systems, Scott constructed the system LCF and presented it together with a mathematical model, combining Richard Platek's doctoral work [Pla66] on hereditarily consistent functionals with Kleene and Kreisel's work on *continuous functionals* [Kre59] (sometimes called the *countable* functionals [Nor06]). Later on, the underlying computational language of LCF (abbreviation of "Logic for Computable Functions") was investigated as a programming language in itself by Plotkin [Plo77] and Milner [Mil77]. They called this language PCF. The mathematical model given for together with LCF (applying implicitly to PCF) by Scott in [Sco69] can be considered the precursor to domain theory.

Ironically, this mathematical model Scott developed to argue for the superiority of typed programming languages led him to find a solution to the open problem of finding a nontrivial mathematical model for the untyped $\lambda$-calculus. Scott's work on constructing a mathematical model for the untyped $\lambda$-calculus is the origin point to which the genesis of domain theory can be traced back.

One can say that at the heart of domain-theoretic programming language semantics lies the abstract notion of an *information ordering* $M \leq N$, to be interpreted as "$N$ provides at least as much information as $M$ does" or "$N$ is a more refined stage of information than $M$". Drawing an analogy between this information ordering and the familiar specialisation preorder from topology allows the importation of topological ideas into programming languages, thereby giving rise to a rich mathematical theory of algebraic structures called *domains*.

In the context of a deterministic programming language, the intuition of the information ordering is easy to motivate: it captures what is called the *observational approximation* order, defined[2] as:

$$M \subset_\sigma N \quad :\equiv \quad \forall P \in \mathsf{Prg}_{\sigma\to\mathtt{Nat}}.\, \forall n \in \mathbb{N}.\, P(M) \Downarrow \lceil n \rceil \to P(N) \Downarrow \lceil n \rceil,$$

[1] Our discussion of history here is mostly based on [LN15].

[2] Our presentation here follows Streicher [Str06].

where $\mathsf{Prg}_\sigma$ denotes the set of closed PCF programs of type $\sigma$. The idea of this observational approximation ordering is:

every observation that can be made about $M$ can also be made about $N$.

This notion of observation is based on the idea of a "finite piece of information" about the result of a computation, which is here expressed by normalisation to a natural number. In the abstract context of a domain, this is what is captured by the information ordering.

The information ordering is of course reminiscent of the familiar specialisation preorder from topology:

**Definition 1** (Specialisation preorder)**.** Let $x, y \in X$ be two points of some topological space $X$. $x$ is below $y$ in the *specialisation preorder* iff one of the following equivalent conditions hold:

1. any open set of $X$ that contains $x$ also contains $y$;
2. $x \in \mathsf{Clos}(\{y\})$.

The analogy here is between finitely observable properties of programs and open neighbourhoods of a point of a topological space. This is one of the key topological insights into programming that underlies the intuition for domains.

As we have previously mentioned, the open problem that first motivated the formulation of the notion of a domain was the construction of a nontrivial mathematical model for the untyped $\lambda$-calculus. A key idea in the construction of such models was the fact that algebraic structures assigned to types must involve this information ordering and the notion of morphism between such structures must be one that respects this information ordering. In other words, if a PCF function $f : \sigma \to \tau$ is computing an inhabitant of $\tau$ when given an inhabitant of $\sigma$, it should be doing this by making only *finite* use of the information provided by its argument.

To arrive at the notion of domain, we need more than the information ordering. In topology, the specialisation preorder can be viewed as a special case of continuity as all continuous maps are monotonic with respect to the specialisation preorder. Similarly, we want to work with a suitable class of continuous functions in domain theory, to define which we require domains to have suprema of directed sets and define continuous maps as those maps preserving these suprema. This leads to the definition[3] of a domain:

**Definition 2** (Domain (pointed dcpo))**.** A *domain* (or *pointed dcpo*) is a directed-complete partial order with a bottom element $\bot$.

**Definition 3** (Directed-complete partial order (dcpo))**.** A *directed-complete partial order* is a poset that is *directed-complete* i.e. all directed subsets of which have joins.

Notice that a domain, as defined in Definition 2, always contains a bottom element. This is to account for the possibility of undefinedness in languages capable of partial computation such as PCF (or the untyped $\lambda$-calculus). The undefined element is below everything else with respect to the observational approximation order as it has no observable behaviour.

Conceptually, directed-complete sets are understood as sequences of finite computations that approximate an infinite computation with increasing accuracy (i.e. the upper bound

[3] Our presentation and terminology here partially follows Streicher [Str06].

that is required to exist in the directed set is a more refined stage of information). Directed-completeness in this context can be understood as the coexistence of finite, *concrete* computations in the domain together with infinite, *ideal* computations (as explained by Escardó in [Esc07a]). We will refrain from fully motivating the definition of a domain in more detail here and hope that the computational intuition we have described so far is sufficient; the reader is referred to [Str06; Plo83; Esc07a] for further reading.

Once the definition of domain is pinned down, the notion of a Scott-continuous function between domains is defined simply as a function that preserves suprema of directed sets:

**Definition 4** (Scott-continuous function)**.** A function $f : D \to E$ between domains is called *Scott-continuous* iff given any directed set $S \subseteq D$,

$$f\left(\bigvee_{s\in S} s\right) = \bigvee_{s\in S} f(s).$$

It is a natural question to ask if $f : D \to E$ should preserve $\bot$ as well. This is essentially the difference between call-by-value and call-by-name evaluation since, in a call-by-name language, a function that takes an undefined argument and returns a defined argument is possible to define whereas in a call-by-value language there is no hope that $f(\bot)$ can be a defined element. As PCF is a call-by-name language and the standard definition of a domain is motivated by constructing a model for PCF, $\bot$ is not required to be preserved in the definition of Scott-continuity.

The use of the term "continuous" in the term "Scott-continuous" is justified by the fact that Scott-continuous functions are exactly the topologically continuous functions when domains are equipped with their Scott topology. We provide the definition of the Scott topology in Definition 6.

**Definition 5** (Scott-open subset of a dcpo)**.** Let $U \subseteq D$ be a subset of some domain $D$. $U$ is said to be *Scott-open* iff it is:

- upwards-closed: $\forall x \in U.\, y \in D.\, x \leq y \to y \in U$;
- inaccessible by directed joins: $\forall S \subseteq D.\, S \text{ directed } \to (\bigvee_{s\in S} s) \in U \to \exists s \in S.\, s \in U$.

**Definition 6** (Scott topology of a dcpo)**.** Let $D$ be a domain. The *Scott topology* of $D$ (denoted $\sigma(D)$) is the topology formed by the Scott-open subsets of $D$.

Once the Scott topology has been defined, we can also make the relationship between the information order and the specialisation order more precise.

**Proposition 1.** *Let $D$ be a domain with information ordering $\sqsubseteq$. The specialisation preorder $\leq$ of the space $(D, \sigma(D))$ coincides with $\sqsubseteq$.*

Note, however, that this equivalence is not constructive unless additional requirements are imposed on the notion of domain.

In practice, one often works with certain subcategories of the category of domains. It is common practice in domain-theoretic programming language semantics to take as the definition of domain, the class of domains one is interested in. Two such classes of domains of interest are *algebraic* and *continuous* domains. We now define these as we will make references to algebraic domains in Section 3.

**Definition 7** (Order of approximation [AJ94, Definition 2.2.1])**.** Let $x, y \in D$ be two elements of a domain $D$. $x$ is said to *approximate* $y$ (denoted $x \ll y$) iff

$$\forall S \subseteq D.\ S \text{ directed} \rightarrow y \leq \bigvee_{s \in S} s \rightarrow \exists s \in S.\ x \leq s.$$

The set $\{y \in D \mid y \ll x\}$ of all approximants of $x$ is denoted $\twoheaddownarrow x$.

This order is also called the "way below" relation or "the order of definite refinement" [Smy86]. Here, we will follow Abramsky and Jung's terminology from [AJ94].

Consider $x, y \in D$ with $x \ll y$. The idea of the approximation order here is that, in relation to $y$, $x$ is a much simpler stage of information. For example, $y$ could be an infinite computation in the domain and $x$ could be a finite approximation or it. We have previously mentioned Escardó's conceptual description of a domain as a set of finite, concrete computations *together with* infinite, ideal computations (see Escardó [Esc07a]). Once the order of approximation has been defined, we can in fact make the idea of a finite computation precise (also called "isolated" and "compact"):

**Definition 8** (Finite element of a domain)**.** Let $D$ be a domain and let $x \in D$. $x$ is called *finite* iff $x$ *approximates* itself i.e. $x \ll x$.

Next, we define the notions of algebraic and continuous domains in Definition 11 and Definition 10.

**Definition 9** (Basis [AJ94, Definition 2.2.3])**.** A subset $B$ of a dcpo $D$ is a basis for $D$ iff for every element $x \in D$, the set $B_x :\equiv \twoheaddownarrow x \cap B$ contains a directed subset with supremum $x$. Elements of $B_x$ are called approximants of $x$ relative to $B$.

**Definition 10** (Continuous domain)**.** A continuous domain is one that has basis.

Notice that the basic cover $B_x$ of every $x \in D$ consists of elements that are finite *relative* to $x$. The notion of an algebraic domain is a strengthening of this in which the elements in the basic covering that are not just finite relative to $x$, but they are also finite relative to themselves (i.e. are finite).

**Definition 11** (Algebraic domain)**.** A dcpo is called *algebraic* iff it has a basis consisting of finite elements. It is called $\omega$-algebraic iff the set of finite elements is a countable basis.

Let us now turn to the interpretation of Stone Duality in terms of domain theory. As we will explain in more detail later, the fundamental contrast in Stone Duality is that between points and opens. From the point of view of logic, this distinction is akin to the distinction between *logical theories* and their *models*. The overarching theme in Abramsky's thesis [Abr91] is that the logical interpretation of this distinction also makes sense in the context of programming languages and computation: domains, that are usually viewed as models, can also be viewed as logical theories. This idea has been termed *Domain Theory in Logical Form* by Abramsky.

We now provide a self-contained introduction to Stone Duality before giving more details on Abramsky's work. To present Stone Duality, we first take a look at the historical context in which it arose. Our presentation here is based on [Abr91] as well as [Joh02].

Stone's famous representation theorem [Sto36] can be motivated (in classical mathematics) starting from a very simple observation: given any set $X$, its powerset $\mathcal{P}(X)$ forms a Boolean

algebra under set union, set intersection, and set difference. The question that motivates Stone's representation theorem is: what about the converse of this implication? In other words, what are the conditions under which a Boolean algebra is the *algebra of all subsets* of some set? A precise characterisation of the conditions under which a Boolean algebra behaves like an algebra of subsets was given by Lindenbaum and Tarski [Tar35] in 1935:

**Theorem 1** (Tarski and Lindenbaum [Tar35])**.** *A Boolean algebra is isomorphic to the powerset algebra of some set iff it is complete and atomic.*

An *atom* in a Boolean algebra $B$ is an element $a \in B$ such that for every $b \leq a$, either $b = 0$ or $b = a$ and a Boolean algebra is said to be atomic iff there exists an atom $a$ below any element $b \neq \bot$.

This characterisation of Boolean algebras that behave like powerset algebras gives a dual equivalence between the category of complete and atomic Boolean algebras and the category of sets. On one side of the duality, we have sets of points and on the other side we have algebras consisting of tokens without any internal structure; this is the hallmark of Stone-type dualities and Theorem 1 can therefore be considered a proto-Stone duality prior to Stone's importation of topological ideas into the question of representation.

The more interesting question of representation that was answered by Stone was the question of generalising this to all Boolean algebras. Of course, it could not be that every Boolean algebra can be expressed as an algebra of all subsets (i.e. powerset algebra) but the question of whether every Boolean algebra could be expressed as an algebra of *certain* subsets remained open.

An answer to this question was given by Stone [Sto36] in his representation theorem. Stone started from the observation[4] that the clopens of any topological space $X$ form a Boolean algebra, which can be viewed as a mapping $\mathsf{Clop} : \mathbf{Top} \to \mathbf{BA}$ of the category of topological spaces into the category of Boolean algebras. Stone constructed a method for going in the opposite direction. This method consisted in associating with a Boolean algebra $B$, the space $\mathsf{Spec}(B)$ of its *ultrafilters*, sometimes referred to as its *spectra*. This space of ultrafilters is topologised by taking as open subsets of the form:

$$U_a :\equiv \{x \in \mathsf{Spec}(B) \mid a \in x\} \quad \text{(for every } a \in B\text{)}.$$

Stone gave an answer to the representation problem by showing that *every Boolean algebra is isomorphic to the algebra of clopens of its space of spectra*:

$$B \quad \cong \quad \mathsf{Clop}(\mathsf{Spec}(B)),$$

and every space is homeomorphic to the space of spectra of its algebra of clopens:

$$X \quad \cong \quad \mathsf{Spec}(\mathsf{Clop}(X)).$$

In addition to these, he showed that Spec and Clop could be extended to morphisms:

$$\frac{X \xrightarrow{\ f\ } Y}{\mathsf{Clop}(X) \xleftarrow[f^{-1}]{} \mathsf{Clop}(Y)} \qquad \frac{A \xleftarrow{\ h^\star\ } B}{\mathsf{Spec}(A) \xrightarrow[h]{} \mathsf{Spec}(B)}$$

[4] Our presentation here follows [Abr88; Abr91].

where $h \equiv \mathrm{Spec}(h^\star) :\equiv x \mapsto \{b \in B \mid h^\star(b) \in x\}$, obtaining functors that yield a *dual equivalence* of categories:

$$\mathrm{Spec} \quad : \quad \mathbf{Stone} \quad \simeq \quad \mathbf{BA}^{\mathrm{op}} \quad : \quad \mathrm{Clop}.$$

This is the archetype of Stone-type dualities and historically, it was the first example of a Stone-type duality.

The fundamental idea in Stone's duality is the fact that the category **BA** of Boolean algebras, which is a familiar category of algebraic objects, is equivalent to a category of different nature: a category of spaces. The discovery of this duality gave rise to many other "Stone-type" dualities by which we understand the geometric interpretation of a category of algebraic objects. Johnstone [Joh83, pg. 42] explains this as follows:

> … but what concerns us here is the revolutionary idea that it is possible to construct *topologically interesting* spaces from purely algebraic data (such as a Boolean algebra).

As an example of another Stone-type duality, consider the question: what is the Stone dual of the more general category of distributive lattices that are not necessarily Boolean? The answer is the category of *spectral spaces*:

$$\mathbf{Spec} \quad \simeq \quad \mathbf{Dist}^{\mathrm{op}}.$$

The role played by ultrafilters in the previous equivalence is undertaken here by the *prime* filters of a distributive lattice. The details of this duality is not particularly interesting for the purposes of this proposal so we will not dwell too much on this.

By generalising this equivalence all the way to its most general form, we end up at the general Stone-type duality between *spatial locales* and frames of opens of *sober spaces*:

$$\mathbf{Sob} \quad \simeq \quad \mathbf{SFrm}^{\mathrm{op}},$$

in which the equivalence in consideration is between the frame of opens of a sober space and the completely prime filters of a spatial locale. This is as close as the equivalence gets to topological spaces in general before it must be relaxed into an adjunction. By weakening this equivalence to an *adjunction*, we obtain a dual adjunction between the category of topological spaces and the category of frames which is, in some sense, the starting point of pointfree topology.

Let us now turn to the question that we are interested in: the meaning of Stone Duality in terms of domain theory. Abramsky's doctoral thesis [Abr91] presents several contributions motivated by Stone Duality:

1. A typed computational metalanguage is constructed; the types of this metalanguage are viewed as universes of discourse for various computational situations and the terms are interpreted as "syntactic intensions" for models.

2. This metalanguage is given a logical interpretation in which types are interpreted as propositional theories and terms are interpreted via a program logic.

3. These two interpretations are shown to be Stone duals of each other. It follows from this that semantics and logic are "guaranteed to be in harmony with each other".

As explained in [AJ94], we can understand the notion of a function in three different ways:

(**a**) as given by an algorithm,

(**b**) as given by a graph,

(**c**) as given by a logical specification.

The passage from (**a**) to (**b**) is roughly the idea of denotational semantics whereas the passage from (**a**) to (**c**) is the idea of *program logics*. In computer science, it is natural to take the interpretation of a function as an algorithm in (**a**) as the native view. In this context, the conceptual contribution of "domain theory in logical form" is to explicate the relationship between the denotational description of program behaviour (as in (**b**)) and the description of program behaviour via specifications expressed in program logics (as in (**c**)). The relationship boils down to a Stone-type duality, suggesting that denotational models of programming languages can be understood by studying the locales corresponding to them.

In [Abr91, Chapter 4], a typed computational language is introduced and is viewed as a metalanguage for denotational semantics. The standard interpretation of the language is given in terms of domains: each type $\tau$ is assigned a domain $\mathcal{D}(\tau)$. In addition to this, each type $\tau$ is assigned a propositional theory $\mathcal{L}(\tau)$, that is viewed as the *localic* side of the Stone Duality in question. It is shown in [Abr91, Theorem 4.2.4] that these are Stone duals of each other.

What is illustrated by Abramsky's dually interpreted metalanguage is that denotational semantics can be understood by reasoning in the category of locales, through assertions made about locales corresponding to domains in which the types of the language in question are interpreted. In Section 3.3, we will mention a possible alternative approach to the problem of interest that uses the category of locales.

## 2.2 Searchable and exhaustible sets

The proposed work that we present in Section 3 is based on Escardó's work on exhaustible sets in the context of higher-type computation [Esc08]. In preparation for Section 3, we present some of Escardó's relevant results from *loc. cit.* in this subsection.

The main contribution of Escardó [Esc08] is the study of exhaustibility and searchability in a domain-theoretic setting, giving methods for building exhaustible and searchable sets. These concepts are to be defined precisely later on, but informally: *exhaustibility* of a set refers to the existence of a universal or existential quantification functional for it whereas *searchability* means the existence of a search functional for the set. It is immediate that every searchable set is exhaustible and, perhaps surprisingly, the converse is also true, which is an important result given in *loc. cit.* by Escardó.

Let us take a look at Escardó's methods for systematically building exhaustible and searchable sets. In [Esc08], it is shown for instance that exhaustible and searchable sets are closed under:

(**I**) intersections with decidable sets,

(**II**) the formation of computable images, and

(**III**) finite and countably infinite products (i.e. the Tychonoff theorem).

Here, (**III**) is of special interest to us since it is the method of building searchable sets that gives rise to the Cantor search functional that we have previously mentioned. In addition to these, Escardó shows that *all* searchable sets are computable images of the Cantor space [Esc08, Theorem 6.2].

The precise formulation of Escardó's results from *loc. cit.* make use of domain theory. The higher-type algorithms of interest that are implicitly defined through these results, however, can be easily expressed informally using the syntax of a functional programming language. To make things a bit more concrete, we will start with such an informal presentation. We first define the Cantor space:

$$\texttt{type Cantor} \coloneqq \texttt{Nat} \to \texttt{Bool},$$

which is the type of sequences of Booleans. An important result due to Ulrich Berger[5], mentioned by Escardó in [Esc08], is the existence of the "seemingly impossible" program that *searches* the Cantor space: the subset of the Cantor type consisting of total elements. The search functional belonging to the Cantor type amounts to a function:

$$\epsilon \ : \ (\texttt{Cantor} \to \texttt{Bool}) \to \texttt{Cantor}$$

that satisfies the specification:

$$p(\epsilon(p)) = \texttt{True} \quad \leftrightarrow \quad \text{there is some } \alpha : \texttt{Cantor} \to \texttt{Bool} \text{ s.t. } p(\alpha) = \texttt{True}.$$

This is to say that $\epsilon$ finds an inhabitant of the Cantor type that satisfies the predicate $p$ if such an inhabitant exists. Otherwise, it returns a counterexample that does not satisfy the predicate $p$. In either case, it is defined to return some total inhabitant of the Cantor space; it is important that the returned counterexample is required to be total as well.

The search functional $\epsilon$ can be concisely written down in functional programming notation as:

$$\begin{aligned} &\lambda\epsilon.\ \texttt{if } \exists\alpha.\ p(\texttt{True} :: \alpha) \texttt{ then} \\ &\qquad \texttt{True} :: \epsilon(\lambda\alpha.\ p(\texttt{True} :: \alpha)) \\ &\quad \texttt{else} \\ &\qquad \texttt{False} :: \epsilon(\lambda\alpha.\ p(\texttt{False} :: \alpha)) \end{aligned}$$

where existential quantification:

$$\exists : (\texttt{Cantor} \to \texttt{Bool}) \to \texttt{Bool}$$

is itself defined as:

$$\exists p = \texttt{True} \quad \leftrightarrow \quad p(\epsilon(p)) = \texttt{True}$$

using the fact that $\epsilon$ satisfies its specification:

$$\exists p = \texttt{True} \quad \leftrightarrow \quad \text{there is some } \alpha : \texttt{Cantor} \text{ s.t. } p(\alpha) = \texttt{True}.$$

In the domain-theoretic setting, the correctness of this program is given by the countable Tychonoff Theorem that we have mentioned in method (**III**). The problem with this situation

[5] Attributed to Berger [Ber90] by Escardó in [Esc08] but the history of the subject seems to be rather complicated. See [Esc08, pg. 3] for details.

is that the proof of correctness of the program makes use of nonconstructive reasoning. Addressing this shortcoming is one of the main results that we aim to achieve in the proposed work. We hope to develop the same program in an intrinsically verified fashion i.e. as the computational content of a constructive proof of an analogue of the Tychonoff Theorem (in a "convenient" category of spaces).

Let us now turn to the formulation of this algorithm in the context of domain theory. Recall that an *effectively given domain* [Smy77] is, conceptually, a domain endowed with a notion of computability. In [Esc08], the algorithm in question is defined in a cartesian closed category of *computable* maps of effectively given domains, containing the flat domains of Booleans and the natural numbers:

$$\begin{aligned} \mathcal{B} &:\equiv \{1, 0, \bot\}; \\ \mathcal{N} &:\equiv \mathbb{N} \cup \{\bot\}. \end{aligned}$$

It is ensured that the algorithms defined in this domain-theoretic setting correspond to PCF programs by working only with computable functions and using the fact that they are closed under $\lambda$-abstraction, application, and least fixed points.

To define the notion of searchability, the notion of a *defined* predicate on the subset of a domain is used:

**Definition 12** (Predicate defined on a subset $K$ [Esc08, Definition 3.1])**.** Given a subset $K \subseteq D$ of a domain $D$, a predicate $p : D \to \mathcal{B}$ is said to be *defined on* $K$ iff $p(x) \neq \bot$ for every $x \in K$.

*Searchability of a subset* of a domain then refers to the existence of a search functional as previously mentioned:

**Definition 13** (Searchable subset [Esc08, Definition 3.3])**.** A subset $K \subseteq D$ of a domain $D$ is said to be *searchable* iff there is a computable functional $\epsilon_K : (D \to \mathcal{B}) \to D$, called the *search functional* for $K$, such that, for every predicate $p : D \to \mathcal{B}$ defined on $K$,

1. $\epsilon_K(p) \in K$, and
2. $p(\epsilon_K(p)) = 1$ iff $p(x) = 1$ for some $x \in K$.

Next is the notion of an exhaustible set which coincides with that of a searchable set in the setting of higher-type computation.

**Definition 14** (Exhaustible set [Esc08, Definition 3.2])**.** A subset $K \subseteq D$ of some domain $D$ is said to be *exhaustible* iff there is a computable functional $\forall_K : (D \to \mathcal{B}) \to \mathcal{B}$, called the universal quantification functional of $K$, that satisfies

$$\forall_K(p) = \begin{cases} 1 & \text{if } p(x) = 1 \text{ for all } x \in K \\ 0 & \text{if } p(x) = 0 \text{ for some } x \in K \end{cases}$$

for any predicate $p : D \to \mathcal{B}$ *defined* on $K$.

It is clear that every searchable set is exhaustible.

**Lemma 1** ([Esc08, Lemma 3.4])**.** *Every searchable set is exhaustible.*

*Proof.* Suppose $K \subseteq D$ is a searchable set with search functional $\epsilon_K$. $K$ must then have a universal quantification functional defined as:

$$\forall_K(p) \quad :\equiv \quad \neg p(\epsilon_K(\neg p)).$$

□

What about the converse of this lemma? It is shown in [Esc08] that the converse holds for *hereditarily total elements* in the hierarchy of continuous functionals. We will revisit this question later on.

Now, let us give the precise definitions of the methods for building searchable sets. These methods do not require any topological machinery, but they are motivated by topological considerations. We first define the notion of a decidable subset.

**Definition 15** (Decidable subset)**.** Let $F \subseteq K \subseteq D$ be two subsets of a domain $D$. $F$ is said to be *decidable on* $K$ iff its characteristic map $\psi_F : D \to \mathcal{B}$ specified as

$$\psi_F(x) = 1 \quad \leftrightarrow \quad x \in F$$

for all $x \in K$, is computable and defined on $K$.

The formulation of (**I**) that we have previously mentioned is as follows:

**Proposition 2** ([Esc08, Proposition 4.2])**.** *Let $F \subseteq K \subseteq D$ be two subsets of the domain $D$ such that $F$ is decidable on $K$.*

1. *If $K$ is exhaustible then so is $K \cap F$.*
2. *If $K$ is searchable then so is $K \cap F$, provided that it is nonempty.*

This proposition can be viewed as a computational analogue of the fact that the intersection of a closed set with a compact set is compact.

Let us now look at the result that corresponds to method (**II**):

**Proposition 3** ([Esc08, Proposition 4.3])**.** *Exhaustible and searchable sets are closed under the formation of computable images.*

Finally, we look at the computational version of the countable Tychonoff Theorem. For this, we first define the product functional that takes the product of a family search functionals:

**Definition 16** (Product functional [Esc08, Definition 4.5])**.** The product functional

$$\Pi : ((D \to \mathcal{B}) \to D)^\omega \to ((D^\omega \to \mathcal{B}) \to D^\omega)$$

is recursively defined by

$$\Pi(\epsilon)(p)(n) :\equiv \epsilon_n \left(\lambda x.\ p_{n,x,\epsilon}\left(\Pi(\epsilon^{(n+1)})(p_{n,x,\epsilon})\right)\right)$$

where

$$p_{n,x,\epsilon}(\alpha) :\equiv p\left(\lambda i.\ \begin{cases} \Pi(\epsilon)(p)(i) & \text{if } i < n \\ x & \text{if } i = n \\ \alpha_{i-n-1} & \text{if } i > n \end{cases}\right)$$

and $\beta^k$ denotes the sequence $\beta$ with the first $k$ elements removed.

The closure of searchable sets under countable products boils down to the following theorem about the product functional:

**Proposition 4** ([Esc08, Theorem 4.6])**.** *Given a countable family $\{K_i\}_{i\in I}$ of subsets of the domain $D$, $\prod_{i\in I}\epsilon_i$ is the search functional for the set $\prod_{i\in I}K_i \subseteq D^\omega$.*

This theorem applies *only* to searchable sets unlike Proposition 2 and Proposition 3, which are results about both searchable and exhaustible sets. It is nevertheless the case that exhaustible sets are closed under countable products as well, but this is obtained via the theorem that says exhaustible and searchable sets coincide in the setting of higher-type computation. Escardó [Esc08, pg. 15] remarks:

> *... we don't know how to approach countable products of boolean-valued quantifiers without the detour via selection functionals at the time of writing.*

Let us now turn to this result of Escardó showing that exhaustible sets are searchable (the aforementioned converse of Lemma 1). For the converse direction, the Kleene-Kreisel topology is used in [Esc08]. To define that, we first need a way of delineating the subset of a domain consisting of total elements:

**Definition 17** (Total functional)**.** For every PCF type $\sigma$, the subset $T_\sigma$ consisting of total elements of the domain $D_\sigma$ is defined as follows:

1. The total elements of $\mathcal{B}$ and $\mathcal{N}$ are the defined elements: $T_{\mathtt{Bool}} :\equiv \{0,1\}$ and $T_{\mathtt{Nat}} :\equiv \mathbb{N}$.
2. For the product type $\sigma\times\tau$, $T_{\sigma\times\tau} :\equiv T_\sigma \times T_\tau$.
3. For the function type $\sigma\to\tau$, $T_{\sigma\to\tau} :\equiv \{f \in D_{\sigma\to\tau} \mid f(T_\sigma)\subseteq T_\tau\}$

Notice that (3) here says: a total function is one that maps total elements to total elements. This subset of total elements is defined in tandem with the *total equivalence* relation:

$$\begin{aligned} x \sim_\gamma y &\iff x,y\in T_\gamma \wedge x = y && \text{(for } \gamma \text{ ground)}\\ (x,y)\sim_{\sigma\times\tau}(x',y') &\iff x\sim_\sigma x' \wedge y\sim_\tau y' \\ f\sim_{\sigma\to\tau} g &\iff \forall x\sim_\sigma y.\ f(x)\sim_\tau g(x) \end{aligned}$$

such that $x\sim_\sigma y$ is to be read "$x$ and $y$ are equal total elements of type $\sigma$".

**Definition 18** (Entire set [Esc08, Definition 4.14])**.** A subset $K\subseteq D$ of the domain $D$ is called *entire* iff it consists of total elements and is closed under total equivalence.

Next, we define the Kleene-Kreisel functionals.

**Definition 19** (Kleene-Kreisel functionals [Esc08, Paragraph 2.3])**.** The *Kleene-Kreisel functionals* of type $\sigma$, denoted $C_\sigma$, are defined by induction, in conjunction with a surjection $\rho : T_\sigma \to C_\sigma$. The idea is that $C_\sigma$ expresses the quotienting of $T_\sigma$ by $\sim_\sigma$. The definition is as follows:

- $C_{\mathtt{Bool}} :\equiv T_{\mathtt{Bool}} \equiv \{0,1\}$.
- $C_{\mathtt{Nat}} :\equiv T_{\mathtt{Nat}} \equiv \mathbb{N}$.

- $\rho_\gamma(x) :\equiv x$, for $\gamma \in \{\texttt{Bool}, \texttt{Nat}\}$.
- $C_{\sigma\times\tau} :\equiv C_\sigma \times C_\tau$.
- $\rho_{\sigma\times\tau} = \rho_\sigma \times \rho_\tau$.

The definitions of $C_{\sigma\to\tau}$ and $\rho_{\sigma\to\tau}$ are a bit more complicated. Let $f : D_\sigma \to D_\tau$ and consider the diagram:

$$\begin{array}{ccccc} D_\sigma & \hookleftarrow & T_\sigma & \xrightarrow{\rho_\sigma} & C_\sigma \\ \downarrow{\scriptstyle f} & & \vdots{\scriptstyle \psi} & & \vdots{\scriptstyle \varphi} \\ D_\tau & \hookleftarrow & T_\tau & \xrightarrow{\rho_\tau} & C_\tau \end{array}$$

The existence of a $\psi$ making the left-hand square commute amounts to $f$ being total i.e. $f \in T_{\sigma\to\tau}$. In this case, there exists a unique $\varphi : C_\sigma \to C_\tau$ making the right-hand square commute, since $\rho_\sigma$ is a surjection. In this context, $C_{\sigma\to\tau}$ and $\rho_{\sigma\to\tau}$ are defined as follows:

- $C_{\sigma\to\tau} :\equiv \{\varphi : C_\sigma \to C_\tau \mid \exists f \in T_{\sigma\to\tau}.\ \rho_\tau \circ \psi = \varphi \circ \rho_\sigma\}$.
- $\rho_{\sigma\to\tau}(f) :\equiv$ the unique $\varphi$ such that the diagram commutes.

**Definition 20** (Shadow)**.** Let $\sigma$ be a PCF type. The *shadow* of a subset $K \subseteq T_\sigma$ is defined to be its $\rho$-image $\rho(K)$.

**Definition 21** (Kleene-Kreisel topology)**.** The *Kleene-Kreisel topology* on $C_\sigma$ is the quotient topology of the Scott topology on $T_\sigma$ by the surjection $\rho_\sigma : T_\sigma \to C_\sigma$.

The following definition of compactness is the main reason we need the machinery of the Kleene-Kreisel topology. It allows us to identify exhaustibility with topological compactness in a precise sense.

**Definition 22** (Kleene-Kreisel compact)**.** A set $K \subseteq T$ is called *Kleene-Kreisel compact* iff its shadow is compact with respect to the Kleene-Kreisel topology.

To import topological ideas into computation, Escardó defines a notion of topological exhaustibility as follows:

**Definition 23** (Topologically exhaustible)**.** A subset $K \subseteq D$ of some domain is called *topologically exhaustible* iff it has a *continuous* quantification functional $\forall_K : (D \to \mathcal{B}) \to \mathcal{B}$ (i.e. one satisfying the same condition as in Definition 14)

The following lemma is crucial for obtaining the converse of Lemma 1 that we have previously mentioned.

**Lemma 2** ([Esc08, Lemma 5.5 (1)])**.** *Any topologically exhaustible set of total elements is Kleene-Kreisel compact.*

This is then used to prove the desired result as follows in Theorem 6.3 from [Esc08]:

**Theorem 2** ([Esc08, Theorem 6.3] (1))**.** *Let $K \subseteq D$ be a nonempty, exhaustible entire set; $K$ is then searchable (uniformly in any quantification functional for $K$).*

Even though we have not presented Escardó's work on exhaustible sets in full detail, this overview should be sufficient to give an idea of the line of work we aim to carry out in the category of $C$-spaces. In Section 3.1, we mention some results that we plan to obtain for the thesis work. These results will be analogous to Escardó's results that we have outlined in this subsection.

## 2.3 Synthetic topology

Even though synthetic topology [Esc04b] is not directly relevant to the thesis work that we propose, it is still tangentially connected: Escardó's work on may-must testing [Esc09] that we present in Section 2.4 draws inspiration from synthetic topology. In fact, Escardó [Esc09, Remark 6.3] remarks that the connection of the executable logic from [Esc09] with synthetic topology might be useful in addressing some problems that arise in the investigation of may-must testing in PCF. As we will make a few references to synthetic topology in the context of the proposed work, we provide a self-contained summary of the core ideas of synthetic topology in this subsection.

In a "synthetic" theory of mathematics, a high-level notion is taken as primitive and mathematical concepts of interest are *derived* from it. The contrast between analytic and synthetic geometry is an illustrative example of the opposition between the analytic and the synthetic in mathematics. In Euclidean geometry, the primitive high-level notions are those of a straightedge and compass. These are taken as primordial and all geometrical constructions are expressed in terms of these two simple high-level devices. Crucially, the notions of straightedge and compass are not broken down into simpler building blocks; their existence is assumed by means of postulates, and other concepts are of interest are derived from these high-level notions. In analytic geometry, on the other hand, concepts of interest such as that of a "line", are built by attempting to explain what they are made of. So in short[6],

- in analytic mathematics, constructions are *analysed* by breaking them down into their constituents, whereas
- in synthetic mathematics, constructions are *synthesised* by means of other high-level notions whose existences are assumed.

This invites the question: what is the sense in which synthetic topology is "synthetic"? In analytic (i.e. point-set) topology, a space is taken to consist of a set of points equipped with some topology. The notion of an open subset is defined as a set of points of the space in consideration, which is in turn used to define the notion of a continuous function. In synthetic topology (as conceived by Escardó [Esc04b]), on the other hand, the concept of a continuous function is taken as primitive and open sets are derived from it as continuous functions into the Sierpiński space. As continuity is hard to make sense of without a notion of space, the identification between continuity and computability is taken advantage of to use computability as a *substitute* for continuity. As computability coincides with continuity to some extent, a computational language can be used as a tool in which concepts of classical topology can be eloquently expressed.

In the *Barbados Notes* [Esc04b], Escardó realises this idea of carrying out classical topology using a computational language. To make this a bit more concrete, let us give the definition of topological space from [Esc04b]. Suppose that we are working with a fixed PCF-like functional language that we will call $\mathfrak{L}$ (for which we will use Haskell notation following [Esc04b]) but we note that the computational language in consideration may vary. The Sierpiński datatype in this language is defined simply to be the unit type with a single inhabitant `tt`. One thinks of the Sierpiński type `S` as a type of truth values of *semidecisions* [Esc04b, pg. 40] with

- `tt` representing truth, and

[6] As explained in [Rey20].

- the undefined inhabitant $\bot$ representing falsity.

The idea is that the truth of a semidecision is observable whereas its falsity is not.

As previously mentioned, in the context of synthetic topology *continuity* is defined as *computability*.

**Definition 24** (Continuous map of datatypes)**.** A function $f : \mathtt{a} \to \mathtt{b}$ of datatypes is *continuous* iff it is definable in $\mathfrak{L}$.

The use of the term "definable" instead of "computable" here highlights an important distinction; the notion of computability is variable with respect to the language in consideration. It is this notion of continuity that other concepts of topology are reduced to. For instance, it allows us to define open sets as maps into the Sierpiński type.

**Definition 25** (Open subset of a datatype)**.** A subset $U$ of a datatype $\mathtt{a}$ is called *open* iff its characteristic function $\chi_U : \mathtt{a} \to \mathtt{S}$ is continuous.

**Definition 26** (Closed subset of a datatype)**.** A subset $F \subseteq \mathtt{a}$ is called *closed* iff its complement is open.

In this setting, a topological space can be defined simply as a subset of a data type of $\mathfrak{L}$.

**Definition 27** (Topological space (synthetic))**.** A *topological space* is a subset $X$ (i.e. of the inhabitants) of some datatype $\mathtt{a}$ in $\mathfrak{L}$.

**Definition 28** (Continuous map of spaces)**.** Given spaces $X \subseteq \mathtt{a}$ and $Y \subseteq \mathtt{b}$, a function $\varphi : X \to Y$ is continuous iff there is at least one continuous function $f : \mathtt{a} \to \mathtt{b}$ with $\varphi(x) = f(x)$ for every $x \in X$.

**Definition 29** (Open subset of a space)**.** Given a space $X \subseteq \mathtt{a}$, a subset $U \subseteq X$ of it is called an *open subset* of $X$ iff there is an open subset $U'$ of $\mathtt{a}$ such that $X \cap U' = U$.

**Definition 30** (Closed subset of a space)**.** Given a space $X \subseteq \mathtt{a}$, a subset $F \subseteq X$ of it is called a *closed subset* of $X$ iff there exists a closed subset $F'$ of $\mathtt{a}$ such that $X \cap F' = F$.

In this synthetic setting, we can take Smyth's topology-computability [Smy93; Smy83] dictionary literally and use it as a tool to write down enlightening proofs of theorems of classical topology. As an illustrative example of this, let us recapitulate the synthetic proof (from [Esc04b]) of the fact that compact subspaces of Hausdorff spaces are closed. The relevant definitions are:

**Definition 31** (Hausdorff space)**.** A space $X \subseteq \mathtt{a}$ is called *Hausdorff* iff its Sierpiński-valued apartness map $\mathtt{apart}_X : \mathtt{a} \to \mathtt{a} \to \mathtt{S}$ specified by

$$\mathtt{apart}_X(x, y) = \mathtt{tt} \leftrightarrow x \neq y, \text{ for every } x, y : X$$

is continuous.

**Definition 32** (Compact space)**.** A space $X \subseteq \mathtt{a}$ is called *compact* iff its Sierpiński-valued universal quantification functional $\forall_X : (\mathtt{a} \to \mathtt{S}) \to \mathtt{S}$ specified by

$$\forall_X(p) = \mathtt{tt} \quad \leftrightarrow \quad p(x) = \mathtt{tt} \text{ for every } x \in X$$

is continuous.

**Proposition 5** ([Esc04b, pg. 53]). *Compact subspaces of Hausdorff spaces are closed.*

*Proof.* Let $X \subseteq \mathtt{a}$ be a Hausdorff space and let $Q \subseteq X$ be a compact subspace of $X$. By definitions of Hausdorff and compact, $X$ has a continuous apartness map

$$\mathtt{apart}_X : \mathtt{a} \to \mathtt{a} \to \mathtt{S}$$

and $Q$ has a continuous universal quantification functional

$$\forall_Q : (\mathtt{a} \to \mathtt{S}) \to \mathtt{S}.$$

We need to show that the characteristic function of the complement of $Q$ is continuous. Intuitively, we can combine $\mathtt{apart}_X$ with $\forall_Q$ to write a function that exhaustively quantifies over every $y$ in $Q$ and checks that a given point of $x$ is apart from $y$ to to conclude that $x$ does not not fall in $Q$. More concretely, we define the following function:

$$\begin{aligned}&\chi_{\bar{Q}} : \mathtt{a} \to \mathtt{S}\\ &\chi_{\bar{Q}} :\equiv \backslash \mathtt{x} \to \forall_Q\ (\backslash \mathtt{y} \to \mathtt{apart}_X(\mathtt{x}, \mathtt{y}))\end{aligned}$$

which satisfies the specification

$$x \notin Q \quad \leftrightarrow \quad \chi_{\bar{Q}}(x) = \mathtt{tt}$$

since $\chi_{\bar{Q}}(x) = \mathtt{tt}$ iff $x \neq y$ for every $y \in Q$, the result follows from the specifications of $\mathtt{apart}_X$ and $\forall_Q$. □

This proof is an apt illustration of how proofs of classical topology can be expressed more directly in the context of synthetic topology. We simply "program" with the definitions of topological spaces.

We conclude our discussion of synthetic topology by mentioning Escardó's work on carrying out synthetic topology in topoi [Esc04a]. The idea of synthetic topology as in [Esc04b] goes back to synthetic domain theory [Hyl91] (which in fact goes back to synthetic differential geometry [Koc06] itself).

The idea in synthetic domain theory is to axiomatise topoi whose objects can be viewed as domains, rather than defining domains as sets with extra structure. Escardó also began work in [Esc04a] on the beginnings of a similar approach to synthetic topology, in which one can work with a topos whose objects behave like topological spaces.

In the topos-theoretic axiomatisation of synthetic topology, the Sierpiński object is axiomatised in a topos, as a subobject $\Sigma \subseteq \Omega$ of the subobject classifier of the topos. In this setting, the choice of $\Sigma$ determines the topological structure that all "sets" (i.e. objects of the topos) come equipped with. This line of work has been pursued further by Davorin Lešnik [Leš10] in his doctoral work.

## 2.4 May-must testing in PCF

In [Esc09], Escardó gives an application of the results discussed in Section 2.2 to the theory of nondeterminism in the context of programming languages, specifically to PCF extended with constructs for may and must testing. Our goal in the proposed work (to be presented in Section 3.2) is to obtain a similar application to nondeterministic testing in programming

languages through a constructive reformulation of Escardó's searchability results. Unlike Escardó's work on may-must testing, however, the proposed work is planned to involve obtaining similar applications to a *total* language. As Escardó's work [Esc09] on may-must testing in PCF forms the basis of the work that we propose, we present Escardó's results from [Esc09] in this section.

In [Esc09], Escardó considers an extension of PCF with angelic, demonic, and probabilistic choice. More specifically, the type system of PCF is extended with type constructors (called "powertypes") that admit certain kinds of nondeterministic testing. The full type system of the nondeterministic extension of PCF in consideration is:

$$\sigma, \tau \quad ::= \quad \mathtt{Bool} \quad | \quad \mathtt{Nat} \quad | \quad \sigma \times \tau \quad | \quad \sigma \to \tau \quad | \quad \mathtt{H}\sigma \quad | \quad \mathtt{S}\sigma \quad | \quad \mathtt{P}\sigma,$$

where H, S, and P are respectively the Hoare, Smyth, and Plotkin powertypes. In fact, a third powertype for probabilistic testing is also considered in [Esc09] and a semidecidability result for it is given as well, but we will not be concerned with probabilistic testing in this report as it is not relevant to the proposed work.

After the addition of Hoare, Smyth, and Plotkin powertypes, the language in consideration is extended with a binary choice operator $\_ \mathbin{⩒}^F \_ : F\sigma \times F\sigma \to F\sigma$ for each powertype $F \in \{\mathtt{H}, \mathtt{S}, \mathtt{P}\}$, such that

$$M \mathbin{⩒}^F N$$

expresses a choice between computation $M$ and computation $N$. Furthermore, each powertype $F \in \{\mathtt{H}, \mathtt{S}, \mathtt{P}\}$ comes equipped with monad operations $\eta$ and $\mu$:

$$\begin{aligned} \eta_F^\sigma &: \sigma \to \mathtt{F}\sigma \\ \mu_F^\sigma &: \mathtt{FF}\sigma \to \mathtt{F}\sigma \end{aligned}$$

as one might expect. Operation $\eta$ here is the lifting operation that allows us to view deterministic computations as nondeterministic ones whereas the multiplication $\mu$ is the operation that flattens nested deterministic computations.

Whenever a choice $M \mathbin{⩒} N$ occurs in nondeterministic PCF, the interpreter must make a choice between computations $M$ and $N$. Therefore, the big-step operational semantics $- \Downarrow -$ is equipped with a scheduler $- \Downarrow^s -$ that decides how to resolve choices. Intuitively, the scheduler expresses whether to choose the left-hand operand or the right-hand operand in each choice so it can be thought of as an infinite sequence of Booleans i.e. a point of the Cantor space. Exactly at this point, the connection to Escardó's results on the searchability of the Cantor space (that we presented in Section 2.2) starts to become visible. In the context of this evaluation relation equipped with a scheduler, we say that a term $M$:

- *must converge* iff for every scheduler $s$ there exists a value $v$ with $M \Downarrow^s v$;
- *may converge* iff there exists a scheduler $s$ and a value $v$ such that $M \Downarrow^s v$.

In must testing, an uncountably infinite set is quantified over, and the semidecidability of may-must testing follows from the fact that this universal quantification is decidable: we can search the space of all possible schedulers for a scheduler under which evaluation gives rise to a value.

Escardó [Esc09] further extends nondeterministic PCF into an *executable program logic* in which may and must testing are internalised by means of two modal operators. In other words,

the extended language can make assertions about programs written in it and by running the interpreter on such programs, the properties can be exhaustively semidecided. This is in fact related to Abramsky's *domain logic* from [Abr91] (see Section 2.1) but a full account of the relationship between the two is not given in [Esc09]. Escardó remarks in [Esc09, pg. 220] that "the relationship between the two approaches deserves more scrutiny". A clarification of the link between the two approaches might be a secondary contribution of the proposed thesis work.

To turn the language into an executable program logic, the language is first extended with a Sierpiński type $\mathtt{S}$ that gives the truth values in this executable program logic. The Sierpiński type has the following constructors:

- the top element $\top : \mathtt{S}$;
- dependent conjunction: $\mathtt{if}\ M\ \mathtt{then}\ N$, where $M : \mathtt{S}$ and $N : \sigma$; and
- parallel or: $M \vee N : \mathtt{S}$ where $M, N : \mathtt{S}$.

The nonterminating element $\bot$ inhabiting the Sierpiński type is interpreted as falsity. We think of $\mathtt{S}$-valued functions $\sigma \to \mathtt{S}$ as the opens of $\sigma$ and view may-must testing constructors as yielding certain open sets of a powertype $\mathtt{F}\sigma$.

It should be noted here that the use of the Sierpiński type is reminiscent of its use in synthetic topology as in the *Barbados Notes* [Esc04b] (see Section 2.3). In synthetic topology, the existence of the Sierpiński type in the computational language in consideration is taken advantage of to express the notion of open set, using the view of open sets $U \in \mathcal{O}(X)$ of a topological space $X$ as *continuous* maps $X \to \mathbb{S}$ from $X$ into to the Sierpiński space $\mathbb{S}$. This enables the formulation of notions of classical topology through a computational language. A formulation of the results in which we are interested in terms of synthetic topology would certainly be interesting, but this is not an investigation we plan to carry out. Time permitting, however, more work can be done on this aspect of the aforementioned executable logic.

To turn the language into an executable logic, it is further extended with may and must testing operators:

- a may testing operator for the Hoare powertype:

$$\Diamond^{\sigma}_{\mathtt{H}} \quad : \quad \mathcal{O}(\sigma) \to \mathcal{O}(\mathtt{H}\sigma);$$

- a must testing operator for the Smyth powertype:

$$\Box^{\sigma}_{\mathtt{S}} \quad : \quad \mathcal{O}(\sigma) \to \mathcal{O}(\mathtt{S}\sigma);$$

  and

- both may and must testing operators for the Plotkin powertype:

$$\begin{aligned} \Diamond^{\sigma}_{\mathtt{P}} \quad &: \quad \mathcal{O}(\sigma) \to \mathcal{O}(\mathtt{P}\sigma) \\ \Box^{\sigma}_{\mathtt{P}} \quad &: \quad \mathcal{O}(\sigma) \to \mathcal{O}(\mathtt{P}\sigma) \end{aligned}$$

Given a powertype $F \in \{\mathtt{H}, \mathtt{P}\}$, the idea of the may testing operator

$$\Diamond_F^\sigma \quad : \quad O(\sigma) \to O(F\sigma)$$

is that $\Diamond_F^\sigma(u)(M) = \top$ iff $u(x) = \top$ for some outcome $x$ of a run of $M$, whereas the must testing operator (for $F \in \{\mathtt{S}, \mathtt{P}\}$)

$$\Box_F^\sigma \quad : \quad O(\sigma) \to O(F\sigma)$$

is defined so that $\Box_F^\sigma(u)(M) = \top$ iff $u(x) = \top$ for *all* outcomes $x$ of a run of $M$.

To make things more concrete, let us look at some examples of programs that can be written using these may-must testing operators. Let $M$ be a program of type $\mathtt{P}(\mathtt{Nat})$. We might want to consider the question of whether outcomes of running $M$ yield prime numbers. We can simply write down an "open set" of $\mathtt{Nat}$:

$$\mathtt{prime} \quad : \quad \mathtt{Nat} \to \mathtt{S}$$

that semidecides if a given program of type $\mathtt{Nat}$ is prime and then convert it to an open of $\mathtt{P}(\mathtt{Nat})$ by using $\Box$ to construct a program $\Box\mathtt{prime} : \mathtt{P}(\mathtt{Nat})$ that tests if all outcomes of $M$ are prime.

Seeing that the Plotkin powertype admits both may and must testing, it is a natural question to ask why the Hoare and Smyth powertypes that admit only certain kinds of testing are considered. The answer is that these powertypes are motivated by considerations of domains that correspond to them. In domain theory, there are three kinds of powerdomains: Hoare, Smyth, and Plotkin powerdomains, and there are important differences between these. For example, Scott domains are closed under Smyth and Hoare powerdomains but not under the Plotkin powerdomain [GLS94]. Also, these powerdomains have different topological interpretations: Hoare, Smyth, and Plotkin powerdomains are viewed respectively as the closed subsets, compact upper sets, and lenses [Esc09, pg. 234].

A powerdomain is the analogue of the notion of a powerset for domains. As mentioned in Section 2.1, the notion of ordering captured by a domain is the *information ordering* $M \leq N$ to be read as "$N$ passes all finitely verifiable tests that $M$ passes". When we go from deterministic programming languages to nondeterministic ones, we are faced with the question of how to define an information ordering on nondeterministic programs. In the nondeterministic setting, we are dealing with a *set of possible outcomes* rather than a single outcome so the question to be answered in this context is: what counts as a finite piece of information about the result of a nondeterministic program (namely, a set of possible results)?

Let $A$ and $B$ be two sets which we think of as sets of possible outcomes of two computations. As we interested in the question of extending an existing deterministic language with nondeterministic constructs, we can assume that the outcomes themselves are already ordered by the information ordering of the underlying deterministic language. There are three natural ways to order a pair of ordered subsets in this context:

- $A \preceq_0 B :\equiv \forall b \in B.\ \exists a \in A.\ a \sqsubseteq b$,
- $A \preceq_1 B :\equiv \forall a \in A.\ \exists b \in B.\ a \sqsubseteq b$, and
- $A \preceq_2 B :\equiv A \preceq_0 B$ and $B \preceq_1 A$, called the *Egli-Milner ordering*.

Starting with each of these preorders, we can get a notion of *powerdomain*: an analogue of the powerset of a set for a domain. The preorders $\preceq_0, \preceq_1, \preceq_2$ give rise, respectively, to

- the Plotkin powerdomain,
- the Hoare powerdomain, and
- the Smyth powerdomain.

To make this a bit more precise, let us briefly explain Smyth's results from [Smy78]. In *loc. cit.*, Smyth gives an easy way of constructing the powerdomains of an $\omega$-algebraic domain (as explained by Winskel [Win85]). Let $(D, \sqsubseteq)$ be a dcpo (as in Definition 2) and assume that it is $\omega$-algebraic. Smyth [Smy78] considers the set $M(D)$ of finite, non-null sets of isolated elements of $D$ and the orders $\preceq_0$ and $\preceq_2$ on them. By taking the ideal completions of these preorders, he obtains the aforementioned powerdomains.

Escardó uses these powerdomains in [Esc09] to give a denotational semantics for nondeterministic PCF, whose details we will not present here. The domain-theoretic semantics used by Escardó in [Esc09] enables the importation of his results on exhaustible sets developed in a domain-theoretic setting in [Esc08] and gives the desired semidecidability results. In Section 3, we propose the investigation of a similar result in the context of *total computation*. The category of $C$-spaces is intended to play the role played by the category of domains in this context.

## 2.5 Sheaves and Grothendieck topoi

Before we proceed to the presentation of $C$-spaces, we first provide a review of elementary notions from sheaf theory for the sake of self-containment. Consider two topological spaces $X$ and $Y$. The continuity of a function $f : X \to Y$ can be *locally determined*, and this fact is made precise through the notion of a *sheaf* (as explained by Mac Lane and Moerdijk [MM94]). Consider the map $C$ that assigns the set $C(U)$ of all continuous functions $f : U \to Y$ to every open $U \in \mathcal{O}(X)$:

$$C(U) \quad :\equiv \quad \{f : U \to Y \mid f \text{ is continuous}\}.$$

There are two salient properties of $C$:

1. Given a continuous function $f \in C(U)$ from an open $U$ of $X$ to some other space $Y$, the restriction $f|_V$ of $f$ to a further open subset $V \subseteq U$ is also continuous i.e. $f|_V \in C(V)$. This is to say that the restriction of a map $f \in C(U)$ to a further open $V \subseteq U$ can be understood as an operation $\_|_V : C(U) \to C(V)$.

2. Given continuous functions $f : C(U)$ and $g : C(V)$ such that $f(x) = g(x)$ for every $x \in U \cap V$, there exists a unique *continuous* function $h : C(U \cup V)$ satisfying $h|_U = f$ and $h|_V = g$.

Property (2) here says that the continuity of a map $f : U \cup V \to Y$ can be understood by examining its behaviour on $U$ and $V$ separately. Accordingly, this is sometimes called the *collatability* condition as it says that continuous maps can be understood as the collation of smaller continuous maps.

If we view the restriction operation $\_|\_$ as the functorial action of $C$, we can reformulate both (1) and (2) in categorical terms. (1) can be understood as a functoriality condition under

the view of the lattice $\mathcal{O}(X)$ of opens of the space $X$ as a category. In this context, $C$ is precisely a contravariant functor $C : \mathcal{O}(X)^{\mathrm{op}} \to \mathbf{Sets}$ as it gives a restriction operation

$$\_|_V : C(U) \to C(V)$$

for any two opens $V \subseteq U$ of $X$. By also expressing (2) in categorical terms, we arrive at the standard definition of the notion of a *sheaf of sets* [MM94], capturing the general idea of a class of functions the membership in which can be locally determined.

**Definition 33** (Sheaf of sets [MM94])**.** A *sheaf of sets* $F$ on a topological space $X$ is a functor

$$F : \mathcal{O}(X)^{\mathrm{op}} \to \mathbf{Sets}$$

such that for each open covering $U = \bigcup_{i \in I} U_i$ of an open $U$ of $X$, the map

$$e : F(U) \to \prod_{i \in I} F(U_i)$$
$$e :\equiv t \mapsto \{t|_{U_i}\}_{i \in I}$$

is the equaliser of the maps

$$p : \prod_{i \in I} F(U_i) \to \prod_{i,j \in I} F(U_i \cap U_j) \qquad q : \prod_{i \in I} F(U_i) \to \prod_{i,j \in I} F(U_i \cap U_j)$$

defined by

$$p :\equiv \{t_i\}_{i \in I} \mapsto \{t_i|_{U_i \cap U_j}\}_{i,j \in I} \qquad q :\equiv \{t_i\}_{i \in I} \mapsto \{t_j|_{U_i \cap U_j}\}_{i,j \in I},$$

as expressed in the diagram:

$$F(U) \dashrightarrow^{e} \textstyle\prod_{i \in I} F(U_i) \underset{q}{\overset{p}{\rightrightarrows}} \prod_{i,j \in I} F(U_i \cap U_j).$$

Functors $F : \mathbb{C}^{\mathrm{op}} \to \mathbf{Sets}$ that do not necessarily satisfy this collatability condition are called *presheaves of sets*.

The general notion of a sheaf is a further abstraction of this idea of a sheaf of sets. More precisely, we abstract by categorifying from the simplified setting of the posetal category of the lattice of opens of a topological space to an arbitrary category. To do this, we generalise our notion of a topology on a space to a generalised "topology" called a *Grothendieck topology*. We call a category equipped with a Grothendieck topology a *site*. Before we give the precise definition of a Grothendieck topology, we define the technical notion of a *sieve*, the categorification of the notion of a *right ideal* of a monoid, which is needed to define Grothendieck topologies.

**Definition 34** (Sieve)**.** Let $\mathbb{C}$ be a category and $C$ an object of $\mathbb{C}$. A functor $S : \mathbb{C}^{\mathrm{op}} \to \mathbf{Sets}$ is called a *sieve* iff it is a subfunctor of the Yoneda embedding $\mathbf{y}(C)$.

The functor $\mathbf{y}$ here is the usual contravariant Hom-functor: $\mathbf{y} :\equiv C \mapsto \mathrm{Hom}_{\mathbb{C}}(-, C)$. An alternative definition of the notion of a sieve would be:

**Definition 35** (Sieve (†))**.** Let $\mathbb{C}$ be a category and $C$ and object of $\mathbb{C}$. A *sieve* on $C$ is a set $S$ of $C$-codomained arrows such that, given any $f : B \to C$ in $S$, and *any* morphism $h : A \to B$, $f \circ h : A \to S$ is in $S$.

The two definitions are equivalent in the sense that given subfunctor of $\mathbf{y}(C)$, we can define

$$S \quad :\equiv \quad \{f : B \to C \mid f \in Q(B), B \in \mathbb{C}\}$$

which would give a sieve in the sense of Definition 35, and given a set $S$ of $C$-codomained arrows as in Definition 35, we can get subfunctor $Q \subseteq \mathbf{y}(C)$ by defining:

$$Q(A) \quad :\equiv \quad \{f \mid f : A \to C \text{ and} f \in S\}.$$

The *maximal sieve* sieve on an object $C$ is the Yoneda embedding $\mathbf{y}(C)$ itself:

**Definition 36** (Maximal sieve)**.** Let $C$ be the object of some category $\mathbb{C}$. The maximal sieve $t_C$ on $C$ is the collection of all $C$-codomained morphisms in $C$ i.e. the Yoneda functor $\mathbf{y}(C)$ itself.

Consider a sieve $S$ on an object $C$ of some category $\mathbb{C}$ and a morphism $h : D \to C$. We define (as in [MM94])

$$h^*(S) \quad :\equiv \quad \{g : E \to D \mid h \circ g \in S\},$$

the sieve on $D$ induced by $h$.

**Definition 37** (Grothendieck topology)**.** A *Grothendieck topology* on a category $\mathbb{C}$ is a function $J$ assigning a collection $J(C)$ of sieves on $C$, satisfying the conditions:

1. the maximal sieve is in $J(C)$,
2. given any $S \in J(C)$ and any morphism $h : D \to C$, $h^*(S) \in J(D)$, and
3. given any $S \in J(C)$ and any sieve $R$ on $C$ such that $h^*(R) \in J(D)$ for all $h : D \to C$ in $S$, then $R \in J(C)$.

**Definition 38** (Site)**.** A *site* is a small category $\mathbb{C}$ equipped with a Grothendieck topology $J$.

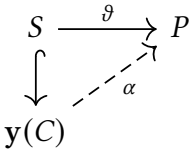
Given a site $(\mathbb{C}, J)$, one refers to a sieve $S \in J(C)$ as a *covering sieve* of $C$.

Now that we have pinned down what we have meant by "generalised topology" on a category, we can generalise sheaves of sets to *sheaves*.

**Definition 39** (Presheaf)**.** A *presheaf* on some category $\mathbb{C}$ is a functor: $F : \mathbb{C}^{\text{op}} \to \mathbf{Sets}$.

**Definition 40** (Matching family [MM94])**.** Let $P$ be a presheaf on some site $(\mathbb{C}, J)$. Given an object $C \in \mathbb{C}$ and a covering sieve $S \in J(C)$, a *matching family* for $S$ is a natural transformation $\vartheta : S \to P$.

**Definition 41** (Sheaf on a site)**.** Given a site $(\mathbb{C}, J)$, a presheaf $P$ on $C$ is called a *sheaf* iff, for all covering sieves $S$ of objects of $C$, any matching family $\vartheta : S \to P$ has a unique extension to $\mathbf{y}(C)$:




This unique extension is called an *amalgamation* for the matching family.

**Definition 42** (Grothendieck topos)**.** A category is called a *Grothendieck topos* iff it is equivalent to the category of sheaves on some site $(\mathbb{C}, J)$.

As pointed out by Johnstone, the notion of a sheaf can also be defined using a simpler notion of Grothendieck topology, called a *Grothendieck pretopology* or a *coverage*[7].

**Definition 43** (Coverage)**.** A *coverage* on a category $\mathbb{C}$ is a function assigning to each object $C \in \mathbb{C}$ a collection $\mathcal{J}(C)$ of families

$$\{f_i : C_i \to C \mid i \in I\}$$

called *covering families*, satisfying the *coverage property*:

> given any $C$ with some covering family $\{f_i : C_i \to C \mid i \in I\}$ and some morphism $g : B \to C$, there exists a covering family:
>
> $$\{h_j : B_j \to B \mid j \in J\}$$
>
> such that each morphism $g \circ h_j$ factors through some $f_i$.

Escardó and Xu define their topos on the uniform-continuity site using coverages. In the next section where we present Escardó and Xu's topos, we will look at an example of a coverage.

Let us conclude our review of elementary ideas of topos theory with a conceptual summary. Recall that an (elementary) topos can be viewed as a category that "behaves like" a category of sets. One might want to work with not just the category **Sets** of sets, for example, but the category $\mathbf{Sets}^{\mathbb{N}}$ of sets varying over time, parametrised by the category $\mathbb{N}$. A Grothendieck topos is like an elementary topos but is a bit more than that: it is not just a category of parametrised sets but a category of *continuously* parametrised sets, with respect to the notion of continuity given by the site in consideration. The work that we propose is planned to take place in the Grothendieck topos constructed by Escardó and Xu [XE13] and we present that in the next section.

## 2.6 $C$-spaces

In this section, we present the $C$-spaces of Escardó and Xu [XE13] that were used to obtain certain constructive results about higher-type computation. Here is a nonexhaustive list of some of these results obtained using $C$-spaces:

1. The category of $C$-spaces has (constructively) a fan functional $(2^{\mathbb{N}} \to \mathbb{N}) \to \mathbb{N}$ that continuously calculates moduli of uniform continuity of maps $2^{\mathbb{N}} \to \mathbb{N}$;
2. The Kleene-Kreisel continuous functionals can be calculated [EX16] (classically) in the category of $C$-spaces;
3. $C$-spaces give (constructively) a continuous model of Gödel's System T and thereby a realisability semantics of $\mathsf{HA}^{\omega}$ [XE13];
4. $C$-spaces give (constructively) a continuous model of Martin-Löf Type Theory [Xu15].

[7] Our definition here follows the presentation of Baez and Hoffnung [BH09].

So what is a $C$-space exactly and how does the motivation for it arise? To put it concisely, one can say that $C$-spaces are generalised spaces that are "nice" in the sense that the category that they form is well-behaved (i.e. is cartesian closed). In addition to this, this category of $C$-spaces is conducive to constructive reasoning.

As the problem we are interested in is expressing ideas of higher-type computation in some category of spaces with the goal of obtaining constructive reformulations of certain results, we would like to be able to do the following in an ideal world: interpret

- types as topological spaces,
- terms as points of these spaces, and
- functions as continuous maps.

Unfortunately, if we take our notion of space to be the usual notion of *topological space*, this wish cannot be fulfilled literally (as explained in [Esc14]) as the category of topological spaces is not sufficiently well-behaved to enable this: it is not a topos and is not even cartesian closed as it does not have exponentials in general. However, there are better behaved categories of spaces in which it is possible to understand types as spaces. There have been many proposals for such "nice" categories whose objects are variations of the notion of topological space. The most relevant one among such categories of generalised spaces for us is Johnstone's *topological topos* [Joh79].

The construction of Johnstone's topological topos is as follows[8]: we take the full subcategory of the category **Top** of topological spaces consisting of only two spaces:

- $\mathbf{1}$, the trivial one-point space; and
- $\mathbb{N}_\infty$, the one-point compactification of the natural numbers under the discrete topology.

We will refer to this subcategory as $\Sigma$. Recall that the one-point compactification of a space is defined as follows:

**Definition 44** (One-point compactification [Kel75])**.** Let $X$ be a topological space. The *one-point compactification* of $X$ (denoted $X_\infty$) is the space whose underlying set is $X \cup \{\infty\}$ and whose topology consists of:

- all opens $U \in \mathcal{O}(X)$, and
- all subsets of the form $(X \setminus F) \cup \{\infty\}$ where $F$ is a compact, closed subset of $X$.

As remarked by Xu [Xu15], the one-point compactification of $\mathbb{N}$ under the discrete topology can be understood as the subset of the Cantor space consisting of decreasing sequences where the sequence

$$1, 1, 1, \ldots$$

consisting of infinitely many 1s represents $\infty$. These two definitions are equivalent classically but not constructively; in the constructive context in which they are not equivalent, the latter is better behaved.

Let us go back to the construction of the topological topos. In the context of the two-element subcategory $\Sigma$ of **Top**, Johnstone defines the topological topos as the Grothendieck topos given by the canonical Grothendieck topology on the two-element category $\Sigma$:

[8] As explained by Xu [Xu15].

**Definition 45** (Johnstone's topological topos [Joh79])**.** The sheaf topos $\mathbf{Shv}(\Sigma, \mathcal{G})$ on the site $(\Sigma, \mathcal{G})$, where $\mathcal{G}$ is the canonical Grothendieck topology.

As we have explained in Section 2.5, a sheaf on the site $(\Sigma, \mathcal{G})$ is a presheaf:

$$F : \Sigma^{\mathrm{op}} \to \mathbf{Sets}$$

(That satisfies certain conditions.) This is to say that every map $f : \mathbf{1} \to \mathbb{N}_\infty$ in $\Sigma$ induces a map

$$F(\mathbb{N}_\infty) \to F(\mathbf{1}).$$

We think of such a sheaf $F$ as a *generalised space* as follows: $F(1)$ is the set of points of the space and $F(\mathbb{N}_\infty)$ is the set of witnesses that a sequence $F(\mathbb{N}_\infty) \to F(\mathbf{1})$ of points converges. $F(\mathbb{N}_\infty)$ here plays the role of a test space: a space the maps from which (sometimes called "probes") into another space of interest allow us to understand the topology of that space.

Escardó and Xu's notion of $C$-space is a variation of Johnstone's notion of space in the topological topos. Instead of using $\mathbb{N}_\infty$ and the canonical Grothendieck topology on $\Sigma$ to obtain a notion of convergent sequence, they work with the monoid of *uniformly continuous* endofunctions on $2^{\mathbb{N}}$.

**Definition 46** (Uniformly continuous endomap on the Cantor space)**.** Let $t : 2^{\mathbb{N}} \to 2^{\mathbb{N}}$ be an endomap on the Cantor space. It is called *uniformly continuous* iff

$$\forall m \in \mathbb{N}.\ \exists n \in \mathbb{N}.\ \forall \alpha, \beta \in 2^{\mathbb{N}}.\ \alpha =_n \beta \to t(\alpha) =_m t(\beta)$$

where $\alpha =_n \beta$ denotes equality up to index $k$ i.e. equality of all $\alpha_k$ and $\beta_k$ for $k \leq n$.

This monoid of uniformly continuous endomaps on the Cantor space is viewed as a one-element category, which replaces $\Sigma$. Let $2^n$ denote the set of binary sequences of length $n$. The canonical Grothendieck topology on $\Sigma$ is replaced by the Grothendieck topology given by the coverage

$$\mathcal{J} \quad :\equiv \quad \{\mathsf{cons}_s\}_{s \in 2^n}$$

where each uniformly continuous map $\mathsf{cons}_s : 2^{\mathbb{N}} \to 2^{\mathbb{N}}$ is the operation of prepending the finite sequence $s$ onto the given sequence $\alpha$:

$$\mathsf{cons}_s(\alpha) \quad :\equiv \quad s\alpha.$$

$\mathcal{J}$ is called the *uniform-continuity coverage* and the site consisting of uniformly continuous endomaps on the Cantor space equipped with this coverage is called the *uniform-continuity site*. The Grothendieck topos formed by sheaves on the uniform-continuity site is Escardó and Xu's variation of Johnstone's topological topos.

To turn our attention towards the construction of $C$-spaces, let us start by defining the notion of a *concrete sheaf* (as given by Baez and Hoffnung [BH09]). As we have just explained, the fundamental idea in the topological topos (as well as Escardó and Xu's topos) is to view certain sheaves as generalised spaces. The defining property of such generalised spaces is that they are given by a Grothendieck topology (or coverage) on a given concrete site. On such sheaves on concrete sites, it makes sense to identify the class of sheaves that are *concrete*. Thinking of these sheaves as generalised spaces, the concrete sheaves among these correspond to spaces that have *underlying sets of points* out of which they are built.

To define the notion of a concrete site, we will need the following definition of a *subcanonical site*.

**Definition 47** (Subcanonical [BH09])**.** A site is *subcanonical* iff every representable presheaf on it is a sheaf.

Given a category $\mathbb{C}$ with terminal object 1, it makes sense to think of the functor

$$\mathsf{Hom}_{\mathbb{C}}(1, -) : \mathbb{C} \to \mathbf{Sets}$$

as giving the "set of points" of an object $C \in \mathbb{C}$. Accordingly, we define the functor

$$\mathsf{Pt}(-) \quad :\equiv \quad \mathsf{Hom}(1, -)$$

for a category $\mathbb{C}$ with terminal object 1. We make use of this when defining the notion of a concrete site [BH09, Definition 18].

**Definition 48** (Concrete site [BH09])**.** A site $(C, J)$ is called *concrete* iff it is subcanonical and $C$ has a terminal object satisfying:

1. The functor $\mathsf{Pt} : \mathbb{C} \to \mathbf{Sets}$ is faithful;
2. For each covering family $\{f_i : C_i \to C \mid i \in I\}$, the family of functions

$$\{\mathsf{Pt}(f_i) : \mathsf{Pt}(C_i) \to \mathsf{Pt}(C) \mid i \in I\}$$

   is jointly surjective.

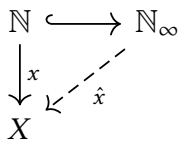
Let $X : \mathbb{C}^{\mathrm{op}} \to \mathbf{Sets}$ be a sheaf on a concrete site and let $C \in \mathbb{C}$. We can turn any $\varphi \in X(C)$ to a function $\hat{\varphi} : \mathsf{Pt}(C) \to X(1)$ by defining

$$\hat{\varphi}(d) \quad :\equiv \quad X(d)(\varphi).$$

**Definition 49** (Concrete sheaf [BH09, Definition 19])**.** Given a concrete site $(\mathbb{C}, \mathcal{J})$, a sheaf $X : \mathbb{C}^{\mathrm{op}} \to \mathbf{Sets}$ is called *concrete* iff for every object $C \in \mathbb{C}$, the function $\varphi \mapsto \hat{\varphi}$ is injective.

The underlying site of both the topological topos and Escardó and Xu's topos are concrete. We view the sheaves in both of these topoi as generalised spaces, and delineate the concrete ones among them as those spaces which we can think of as being built out of points. In the case of the topological topos, the spaces that arise as the concrete sheaves are called the *subsequential spaces*.

**Definition 50** (Convergence)**.** Let $X$ be a set. Given a collection $C \subseteq \mathbb{N}_\infty \to X$, we say that a sequence $x : \mathbb{N} \to X$ *converges to* $y$ iff there exists some $\hat{x} \in C$ such that the diagram




commutes and $\hat{x}(\infty) = y$.

**Definition 51** (Subsequential space [Joh79, pg. 241])**.** A *subsequential space* is a set $X$ together with a collection of functions $\mathbb{N}_\infty \to X$, called convergent sequences in $X$, satisfying the following conditions:

1. For every $x \in X$, the constant sequence $\{x\}_{n\in\mathbb{N}}$ converges to $x$;
2. If $\{x_n\}_{n\in\mathbb{N}}$ converges to $x$ then so does every subsequence of it;
3. If $\{x_n\}_{n\in\mathbb{N}}$ is a sequence and $x$ a point such that every subsequence of $\{x_n\}_{n\in\mathbb{N}}$ contains a subsequence converging to $x$ then $\{x_n\}_{n\in\mathbb{N}}$ converges to $x$.

The notion of a subsequential space is a way of representing the kind of spaces that arise as the concrete sheaves of the topological topos in a more concrete way. The idea of $C$-spaces arises in exactly the same way, but in the context where the topological topos is replaced by the Escardó-Xu topos on the uniform-continuity site.

**Definition 52** ($C$-space)**.** A $C$-space is a set $X$ equipped with a set $P$ of maps $2^{\mathbb{N}} \to X$, called the *probes* of $X$, satisfying the following probe axioms:

1. All constant maps are in $P$;
2. If $p \in P$ and $t \in 2^{\mathbb{N}}$ then $p \circ t \in P$;
3. For any $n \in \mathbb{N}$ and any family $\{p_s \in P\}_{s\in 2^n}$, the unique map $p : 2^{\mathbb{N}} \to X$ defined by $p(s\alpha) = p_s(\alpha)$ is in $P$.

A continuous map from $C$-space $(X, P)$ to $C$-space $(Y, Q)$ is a function $f : X \to Y$ such that $f \circ p \in Q$ whenever $p \in P$.

At this point, we should also mention the connection of $C$-spaces to Spanier's [Spa63] notion of quasitopological space:

**Definition 53** (Quasitopological space)**.** A *quasitopological space* is a set $X$ together with a quasi-topology: a function $Q$ that assigns a set $Q(K, X)$ of functions $K \to X$ such that,

1. all constant maps $K \to X$ are in $P(K, X)$;
2. given any continuous map $t : K \to K'$ and $q \in Q(K, X)$, then $q \circ t \in Q(K', X)$; and
3. given a family $\{t_i : K_i \to K\}_{i\in I}$ of finite, jointly surjective family and a map $q : K \to X$ with $q \circ t_i \in Q(K_i, X)$ for every $i \in I$, then $q \in Q(K, X)$.

**Definition 54** (Quasicontinuous map)**.** Let $X$ and $Y$ be quasitopological spaces. A function $f : X \to Y$ is called quasicontinuous iff $f \circ q \in Q(K, Y)$ for every $q \in Q(K, X)$.

Escardó and Xu's $C$-spaces are an analogue of the notion of quasitopological space that are more suitable for a constructive development. Instead of all compact Hausdorff spaces, only one compact Hausdorff space, namely the Cantor space, is considered in the definition of a $C$-space. This gives rise to a definition of generalised space more amenable to constructive reasoning.

The subcategory of Escardó and Xu's topos consisting of $C$-spaces is not a topos as the subobject classifier falls outside it. It is, however, cartesian closed and is sufficent for many purposes, including the construction of a model for System T, which involves a constructive account of the fan functional. Furthermore, it was shown by Escardó and Xu in [EX16] that the Kleene-Kreisel continuous functionals can be computed within the category of $C$-spaces, which is of special interest for working with total computation as we have mentioned in Section 2.2. It is this category in which we plan to carry out the proposed work of giving a constructive account of Escardó's results from [Esc08]. We explain the plan in more detail in Section 3.1.

# 3 Proposed Work

We present the proposed work in this section which consists of two main components:

1. Developing Escardó's results on searchable and exhaustible sets constructively in the category C-**Space** of $C$-spaces;

2. Obtaining, from these results, an application to may-must testing in the context of total computation (as opposed to the work in Section 2.4 that takes place in the context of partial computation).

This section is broken up into Section 3.1 and Section 3.2 in which we explain (1) and (2) in more detail respectively.

## 3.1 Higher-type computation in the category of $C$-spaces

The category of $C$-spaces was used by Escardó and Xu to obtain constructive reformulations of well-known proofs. In this section, we sketch how the notions of exhaustibility and searchability from [Esc08] can be formulated in Escardó and Xu's category of $C$-spaces.

As mentioned before, the result of Escardó [Esc08] in which we are interested is the method of "systematically building exhaustible and searchable sets" (given in Section 4 of [Esc08]): exhaustible and searchable sets are: (1) closed under intersections with decidable sets, (2) under the formation of computable images, and (3) of finite and countably infinite products.

A natural question to ask at this point is: would it not be possible to develop a Tychonoff Theorem for *types* in the setting of type theory? What exactly is the motivation for working in a category of spaces? Tychonoff Theorem for types would require axioms for postulating the existence of a well-behaved Sierpiński object in the setting of type theory, satisfying, for example, the dominance axiom [Ros86; Esc04a]. Carrying out a proof of the Tychonoff Theorem in the category of $C$-spaces can therefore be viewed as a way of avoiding the use of such axioms.

Let us now formulate the notion of a *searchable* $C$-space which is an adaptation of the one from [Esc08] (which we gave in Definition 13):

**Definition 55** (Searchable $C$-space)**.** A $C$-space $X$ is called *searchable* iff for every continuous map $p : X \to \mathbf{2}$ of $C$-spaces, there is a designated $x_0 \in X$ such that

$$\text{if } p(x_0) = 1 \text{ then } p(x) = 1 \text{ for every } x \in X.$$

Note that the set $\mathbf{2}$ is used here to express predicates. Unlike in Definition 13 where the domain $\mathcal{B}$ is used, this set has no undefined inhabitant and therefore we do not have to formulate a notion of defined predicate; all predicates are defined.

The Tychonoff Theorem for $C$-spaces can then be easily expressed as follows:

**Theorem 3** ((Countable) Tychonoff Theorem for $C$-spaces)**.** *Let $\{X_n\}_{n\in\mathbb{N}}$ be a family of $C$-spaces. If every $C$-space $X_n$ is* searchable *then $\prod_{n\in\mathbb{N}} X_n$ is searchable.*

To achieve this result, we will have to define an analogue of the product functional (see Definition 16) and prove a theorem similar to Theorem 4. The primary contribution of the proposed work will be a proof of this, and Theorem 3 as a corollary, which will be a constructive

version of Escardó's results from [Esc08] albeit in the simplified setting of total computation. We also plan to consider other methods for building searchable sets in the category of $C$-spaces such as Proposition 2.

The Cantor $C$-space is easy to describe. Recall that Escardó and Xu's topos is built atop the one-element category of uniformly continuous endomaps of the Cantor space. The only element of the monoid is the Cantor space $\mathbf{2}^{\mathbb{N}}$ itself. The Cantor $C$-space is simply the Yoneda embedding $\mathbf{y}(\mathbf{2}^{\mathbb{N}})$ of the Cantor space. In this setting, we should also be able to obtain a result analogous to the fact that every searchable set is a computable image of the Cantor space, but the precise formulation of this is not clear as of writing.

## 3.2 Application to may-must testing in a total setting

After we develop analogues of Escardó's methods for constructing searchable sets in C-**Space**, we plan to apply these to may-must testing in the setting of total computation, specifically, to the language System T of higher-order primitive recursion. The denotational semantics of System T is usually not interesting as it does not require structures as involved as domains; any cartesian closed category suffices. Our motivation for using the continuous model of System T given by the category of $C$-spaces is, however, the fact that we will be studying a version of System T extended with nondeterministic testing constructs which are to be given a topological interpretation. We aim this topological interpretation to coincide with the topological developments that we plan to carry out in the category of $C$-spaces.

Let us start with a definition of the nondeterministic extension of System T that we plan to consider. Gödel's System T is a language of higher-order primitive recursion with a base type `Nat` of natural numbers. Here, we consider System T augmented with product and sum types as well as constructs for nondeterministic testing, similar to Escardó's extension of PCF into an executable logic. The full syntax of the language we plan to consider is given in Figure 3.2.

We will not define the full type system and the operational semantics of the language here. It is important to note, however, that the must testing operator for the Smyth powertype gives a function of type:

$$\Box^{\tau} \quad : \quad (\tau \to \mathtt{Bool}) \to \mathtt{S}(\tau) \to \mathtt{Bool};$$

it takes a predicate of type $p : \tau \to \mathtt{Bool}$ and returns a `Bool` indicating if all possible outcomes of a nondeterministic program of type $\tau$ satisfy it. Similarly, the may testing construct for the Hoare powertype has the type:

$$\Diamond^{\tau} \quad : \quad (\tau \to \mathtt{Bool}) \to \mathtt{H}(\tau) \to \mathtt{Bool}.$$

As we are working in the context of total computation, the Sierpiński type (that was used in Escardó's work on may-must testing in PCF) is replaced with the type of Booleans.

The topological motivation for this investigation of extended System T is that the computational interpretation of the countable Tychonoff Theorem can be interpreted as a product functional `tychonoff` of type

$$\mathtt{tychonoff} \quad : \quad (\mathbb{N} \to \mathtt{S}(\tau)) \to \mathtt{S}(\mathbb{N} \to \tau)$$

in this language. The Cantor search functional can then be written down as a special case of `tychonoff`.

Figure 1: Syntax of nondeterministic System T.

| | | | | |
|---|---|---|---|---|
| *Powertypes* | $F$ | ::= | $\mathtt{H}$ | Hoare powertype |
| | | \| | $\mathtt{S}$ | Smyth powertype |
| *Types* | $\tau$ | ::= | $\mathtt{Nat}$ | natural numbers |
| | | \| | $\mathbf{0}$ | void |
| | | \| | $\mathbf{1}$ | unit |
| | | \| | $\tau_0 \times \tau_1$ | product |
| | | \| | $\tau_0 + \tau_1$ | sum |
| | | \| | $\tau_0 \longrightarrow \tau_1$ | function |
| | | \| | $F(\tau)$ | powertype |
| *Terms* | $t$ | ::= | $x$ | variable |
| | | \| | $\mathtt{zero}$ | zero |
| | | \| | $\mathtt{succ}(t)$ | successor |
| | | \| | $\star$ | inhabitant of $\mathbf{1}$ |
| | | \| | $\langle t_1, t_2 \rangle$ | pairing |
| | | \| | $\pi_1(t)$ | left projection |
| | | \| | $\pi_2(t)$ | right projection |
| | | \| | $\mathsf{inl}(t)$ | left injection |
| | | \| | $\mathsf{inr}(t)$ | right injection |
| | | \| | $\mathtt{match}\ t\ \mathtt{with}\ \{\}$ | null case |
| | | \| | $\mathtt{match}\ t\ \mathtt{with}\ \{\mathsf{inl}(x_1) \mapsto t_1 \mid \mathsf{inr}(x_2) \mapsto t_2\}$ | case analysis |
| | | \| | $\mathtt{rec}\ e\ \{\mathtt{zero} \mapsto t_0 \mid \mathtt{succ}(x)\ \mathtt{with}\ y \mapsto t_1\}$ | recursor |
| | | \| | $\lambda x : \tau.\ t$ | abstraction |
| | | \| | $t_0(t_1)$ | application |
| | | \| | $t_0 \mathbin{\textcircled{\scriptsize V}^F} t_1$ | choice |
| | | \| | $\eta^F$ | monad unit |
| | | \| | $\mu^F$ | monad multiplication |
| | | \| | $\Box^\sigma\ t_0\ t_1$ | must testing |
| | | \| | $\Diamond^\sigma\ t_0\ t_1$ | may testing |

The semidecidability results mentioned in Section 2.4 come from the computational interpretation of the fact that the Cantor space is topologically compact (see Lemma 2). In this context, we plan to obtain this search functional using the planned constructive proof of Theorem 3. More importantly, using the denotational model of System T in C-**Space**, we plan to extract this function as a realiser of the countable Tychonoff Theorem. In other words, this function must be a witness of the fact that the $\mathsf{S}(\mathbb{N} \to \tau)$ powertype in consideration is searchable, which can be understood as a certain product functional that we believe we will be able to construct in the category of $C$-spaces.

As exemplified by `tychonoff`, our primary motivation for studying the extension of System T with powertypes is the fact that our goal for the thesis is to give powertypes a *dual* interpretation in the context of total computation:

**(I)** operationally, they will be understood as language constructs for organising nondeterministic programming;

**(II)** denotationally, they will be interpreted as the computational manifestations of *compact* and *overt* sets.

In addition to this topological motivation, we also want to consider System T for the following reasons:

1. We want to study exhaustible and searchable sets without going through the complication of domains (and classical topology). Total computation can be studied in the category of $C$-spaces which is more suitable for constructive reasoning as we have previously mentioned.

2. Total computation is interesting in itself. For example, languages based on type theory (such as the Agda programming language) are total.

We conclude this subsection by summarising the work to be done on System T:

- *Operational semantics.* The operational semantics of the language is to be constructed and formalised.

- *Denotational semantics.* A denotational semantics corresponding to the operational semantics for the language must be given in C-**Space**. This has already been done for the core of System T by Xu and Escardó but we will have to extend it to account for nondeterministic testing constructs. Soundness of this semantics is to be proved.

- *Decidability.* Decidability of may and must testing is to be proved.

- *Proof of totality of operational semantics.* The fact that System T remains total when extended with nondeterministic testing constructs is to be proved. This will require some form of Tait's logical relations method.

- *Logic.* System T naturally corresponds to the logic $\mathsf{HA}^{\omega}$. This program logic is to be extended in conjunction with System T's augmentation with nondeterministic constructs.

## 3.3 Category of locales as an alternative?

An alternative approach to reformulating ideas of Escardó's work on exhaustible and searchable sets in PCF would be to use the category **Loc** of locales. This would in fact be a very natural approach to the aforementioned problem as the category of locales is suitable for constructive reasoning. However, there are some problems with this approach, the most salient of which being the fact that we do not know how to work with the notion of totality in it.

Denote by $\mathbb{N}_\perp$ and $\mathbf{2}_\perp$ the frame of opens of Scott topologies of the flat domains of the natural numbers and the Booleans. The beginning of this approach could be to start with the exponential locale $\mathbf{2}^{\mathbf{2}^{\mathbb{N}}}$ and consider its embedding into the corresponding domains to attempt to define the *total* elements of the domains:

$$\begin{array}{ccc} \mathbf{2}^{\mathbf{2}^{\mathbb{N}}} & \xrightarrow{\epsilon} & \mathbf{2}^{\mathbb{N}} \\ \downarrow & & \downarrow \\ \mathbf{2}_\perp^{\mathbf{2}_\perp^{\mathbb{N}_\perp}} & \xrightarrow{\texttt{search}} & \mathbf{2}_\perp^{\mathbb{N}_\perp} \end{array} \tag{‡}$$

Recall that in [Esc08], the idea of a total element of a domain plays a fundamental role in the definition of the Kleene-Kreisel continuous functionals. More specifically, Kleene-Kreisel continuous functionals are defined by means of the embeddings $T_\sigma \hookrightarrow D_\sigma$. The idea here would be that that this diagram would play a similar role of expressing the total elements. However, there are some problems with this.

In order to carry out this development, we have to start by defining the locales $\mathbf{2}^{\mathbb{N}}$ and $\mathbf{2}^{\mathbf{2}^{\mathbb{N}}}$. In fact, we have have already constructed the Cantor space $\mathbf{2}^{\mathbb{N}}$ [Tos20, Chapter 5] using a formal-topological description of it, but we have not yet proved the universal property i.e. that it is the exponential. This is the first thing that we would have to complete to get started with this development.

After this, we need to construct the domains $\mathbf{2}_\perp$, $\mathbb{N}_\perp$, and $\mathbf{2}_\perp^{\mathbb{N}_\perp}$ as locales, which involves a construction of the exponential. The category of locales does not have exponentials in general. The precise description of exponentiable locales was given by Hyland [Hyl81].

**Theorem 4** (Hyland [Hyl81])**.** *Given a locale $X$, the exponential $Y^X$ exists for every locale $Y$ iff $X$ is locally compact.*

In addition to these challenges, the more serious problem with working in the category of locales is that it is not clear how to get such an embedding

$$\mathbf{2}^{\mathbf{2}^{\mathbb{N}}} \hookrightarrow \mathbf{2}_\perp{}^{\mathbf{2}_\perp{}^{\mathbb{N}_\perp}}$$

mentioned in diagram (‡).

In summary, these problems are the motivation for taking the primary approach of using the category of $C$-spaces and restricting attention to total computation. If time permits, we plan to also work on this question of constructive reasoning about exhaustible and searchable sets in the category **Loc** but this will not be our primary focus (at least at the beginning).

Finally, we should also mention at this point that this approach could also make use of the implementation of formal topology in type theory due to Thierry Coquand and the author [CT21]. Formal topologies play a role similar to the notion of a domain: they present topologies. Sara Negri [Neg02] gives a description of the precise relationship between the

two and explains how to define domains as certain kinds of formal topologies. It might also be possible to use this understanding of domains to try to facilitate this development in the category of locales (or even carry it out using formal topologies) but the details of this are not clear at the time of writing. This is another idea that we would like to study if time permits.

# 4 Evaluation of Work

## 4.1 Evaluation of the development in C-Space

To state it in simple terms, the evaluation criterion for the proposed work amounts to the question: have we achieved *constructive* reformulations of the ideas from Escardó's work on exhaustible sets or otherwise, have we proved that they are not possible? However, the extent to which we may achieve this goal may vary. For example, we might achieve the main result we are interested in (the Tychonoff Theorem) even though we fail to develop constructive accounts of the other results.

The bare minimum of the results we would like to achieve is the Tychonoff Theorem for $C$-spaces. One can say that the proposed thesis work will not be considered done without this. It is also important that we study other methods for building searchable sets as well, such as the correspondence between exhaustible and searchable sets. It is not clear a priori that these two notions will coincide in the category of $C$-spaces. Either a positive or negative result in this direction could be interesting. Another result we may consider in the category of $C$-spaces is the Kleene-Kreisel density theorem.

In short, the evaluation criteria for the theoretical side of the proposed work boils down to the assessment of the extent to which we succeed to develop constructions and theorems of higher-type computation relevant to searchable and exhaustible sets in the category of $C$-spaces.

## 4.2 Evaluation of application to may-must testing

The goal to be achieved on this side of the proposed work is inherently connected to the development in $C$-spaces, as we will be viewing programs in extended System T as computational manifestations of topologically inspired facts in C-**Space**. Nevertheless, even if we fail to achieve the desired goals in the category of $C$-spaces, it should still be possible for us to obtain meaningful results about nondeterministic System T. We have already provided a summary of the work to be done on System T in Section 3.2; let us now review this from the point of view of evaluation criteria.

- *Operational semantics of the language.* The work to be completed here should be relatively straightforward. However, as the goal for the operational semantics is for it to coincide with the topological notions in the denotational semantics, the definition of the operational semantics might change as the denotational semantics is constructed. The goal to be achieved is, however, rather well-defined.

- *Denotational semantics of the language.* The goal of the denotational semantics is to enable a transition from System T to the category of $C$-spaces, using which we will be able to internalise facts about $C$-spaces in the nondeterministic extension of System T.

- *Decidability of testing.* This is perhaps the most salient result we hope to achieve. This result can itself be considered an evaluation criterion for success for the for the proposed work.

- *Logic.* The program logic to be constructed here should be topologically meaningful. We plan to devote a short section of the thesis to the relationship between this and Abramsky's work on domain logic.

For the application side of the proposed work, we must complete all of these goals.

# 5 Plan and Timetable

The tentative plan for the proposed work also follows the distinction between the theoretical side and the application side of the work that we have proposed:

- The results that need to be obtained in the category of $C$-spaces (corresponding to Section 3.1);
- Applications of these results about $C$-spaces to may-must testing in the total setting of System T (corresponding to Section 3.2).

At this point, it is hard to provide a concrete timetable for the work that we plan to carry out. We provide, however, a nonexhaustive list of tasks ordered with respect to their priorities. This corresponds to the order in which we aim to carry out the tasks that we have outlined throughout this proposal.

(**I**) Formulation of topological notions in C-**Space**. We already have a concrete definition of searchability. What about exhaustibility? Can it be easily defined in C-**Space**?

(**II**) Experiments with searchable and exhaustible $C$-spaces. We can start by proving simple properties about them as a warm-up exercise. An example of this could be closure under intersection with decidable sets.

(**III**) Closure under finite products. This should not be hard and should be a good way starting to work with products of $C$-spaces in preparation for the countable Tychonoff Theorem.

(**IV**) **Milestone**: Countable Tychonoff Theorem for $C$-spaces Achieving this will be the foundation for the rest of the thesis work so it is quite important. Hopefully, this could be completed before the end of Summer 2022.

(**V**) Operational semantics for extended System T.

(**VI**) Proof of totality of the operational semantics.

(**VII**) Denotational semantics for extended System T.

(**VIII**) Kleene-Kreisel density theorem.

(**IX**) **Milestone**: Definition of the Cantor search functional using the product functional given by (**IV**). This is quite an important step as it will present the contribution of a nontrivial program extracted from "spatial reasoning" in the category C-**Space**. It should also be suitable for publication if it turns out as we expect.

(**X**) Equivalence of exhaustibility and searchability for $C$-spaces.

(**XI**) Construction of a program logic for extended System T and the investigation of its relationship to Abramsky's work.

(**XII**) **Further topic**: Pointfree reasoning about computation in the category **Loc**.

(**XIII**) **Further topic**: Study of nondeterministic System T further extended with the type of real numbers.

(**XIV**) **Further topic**: Study of System T further extended with the probabilistic powertype.

# References


[Abr88] Samson Abramsky. "Domain Theory In Logical Form". PhD thesis. Imperial College of Science and Technology, Sept. 2, 1988.

[Abr91] Samson Abramsky. "Domain Theory in Logical Form". In: *Annals of Pure and Applied Logic* 51.1 (Mar. 14, 1991), pp. 1–77. ISSN: 0168-0072. DOI: 10.1016/0168-0072(91)90065-T.

[AJ94] Samson Abramsky and Achim Jung. *Domain Theory*. 1994.

[Ber90] Ulrich Berger. "Totale Objekte Und Mengen in Der Bereichstheorie". Mathematisches Institut der Universität München, 1990.

[BH09] John C. Baez and Alexander E. Hoffnung. *Convenient Categories of Smooth Spaces*. Oct. 8, 2009. arXiv: 0807.1704 [math]. URL: http://arxiv.org/abs/0807.1704 (visited on 01/21/2022).

[Čec37] Eduard Čech. "Topologicke Prostory". In: *Časopis pro pěstování matematiky a fysiky* 66.4 (1937), pp. D225–D264.

[Coq92] Thierry Coquand. "An Intuitionistic Proof of Tychonoff's Theorem". In: *The Journal of Symbolic Logic* 57.1 (Mar. 1992), pp. 28–32. ISSN: 0022-4812, 1943-5886. DOI: 10.2307/2275174.

[CT21] Thierry Coquand and Ayberk Tosun. "Formal Topology and Univalent Foundations". In: *Proof and Computation II*. WORLD SCIENTIFIC, Feb. 3, 2021, pp. 255–266. ISBN: 9789811236471. DOI: 10.1142/9789811236488_0006.

[EK17] Martín H. Escardó and Cory M. Knapp. "Partial Elements and Recursion via Dominances in Univalent Type Theory". In: *26th EACSL Annual Conference on Computer Science Logic (CSL 2017)*. Ed. by Valentin Goranko and Mads Dam. Vol. 82. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2017, 21:1–21:16. ISBN: 978-3-95977-045-3. DOI: 10.4230/LIPIcs.CSL.2017.21.

[Esc04a] Martín H. Escardó. "[Synthetic] Topology via Higher-Order Intuitionistic Logic". Unpublished note. Unpublished note. Mar. 18, 2004.

[Esc04b] Martín H. Escardó. "Synthetic Topology: Of Data Types and Classical Spaces". In: *Electronic Notes in Theoretical Computer Science*. Proceedings of the Workshop on Domain Theoretic Methods for Probabilistic Processes 87 (Nov. 12, 2004), pp. 21–156. ISSN: 1571-0661. DOI: 10.1016/j.entcs.2004.09.017.

[Esc07a] Martín H. Escardó. "Domain Theory and Denotational Semantics of Functional Programming". Lecture. Midlands Graduate School 2007 (Nottingham). Apr. 20, 2007. URL: http://www.cs.nott.ac.uk/~psznhn/MGS2007/LectureNotes/mgs2007-dom.pdf.

[Esc07b] Martín H. Escardó. "Infinite Sets That Admit Fast Exhaustive Search". In: *22nd Annual IEEE Symposium on Logic in Computer Science (LICS 2007)*. 22nd Annual IEEE Symposium on Logic in Computer Science (LICS 2007). Wroclaw, Poland: IEEE, 2007, pp. 443–452. ISBN: 978-0-7695-2908-0. DOI: 10.1109/LICS.2007.25.

[Esc07c] Martín Hötzel Escardó. *Seemingly Impossible Functional Programs*. Sept. 28, 2007. URL: http://math.andrej.com/2007/09/28/seemingly-impossible-functional-programs/ (visited on 01/22/2020).

[Esc08] Martín H. Escardó. "Exhaustible Sets in Higher-Type Computation". In: *Logical Methods in Computer Science* Volume 4, Issue 3 (Aug. 2008). DOI: 10.2168/LMCS-4(3:3)2008.

[Esc09] Martín H. Escardó. "Semi-Decidability of May, Must and Probabilistic Testing in a Higher-type Setting". In: *Electronic Notes in Theoretical Computer Science*. Proceedings of the 25th Conference on Mathematical Foundations of Programming Semantics (MFPS 2009) 249 (Aug. 8, 2009), pp. 219–242. ISSN: 1571-0661. DOI: 10.1016/j.entcs.2009.07.092.

[Esc14] Martín H. Escardó. "Continuity in Type Theory". Semantics of Proofs and Programs, IHP (Paris). June 12, 2014. URL: https://www.cs.bham.ac.uk/~mhe/.talks/ihp2014/escardo-ihp2014.pdf.

[Esc21] Martín H. Escardó. *Computing an Integer Using a Grothendieck Topos*. Mathematics and Computation. May 18, 2021. URL: http://math.andrej.com/2021/05/18/computing-an-integer-using-a-sheaf-topos/ (visited on 01/28/2022).

[Esc99] Martín H. Escardó. "On the Compact-regular Coreflection of a Stably Compact Locale". In: *Electronic Notes in Theoretical Computer Science* 20 (1999), pp. 213–228. ISSN: 15710661. DOI: 10.1016/S1571-0661(04)80076-8.

[EX16] Martín H. Escardó and Chuangjie Xu. "A Constructive Manifestation of the Kleene–Kreisel Continuous Functionals". In: *Annals of Pure and Applied Logic*. Fourth Workshop on Formal Topology (4WFTop) 167.9 (Sept. 1, 2016), pp. 770–793. ISSN: 0168-0072. DOI: 10.1016/j.apal.2016.04.011.

[GLS94] "Power Domains". In: *Mathematical Theory of Domains*. Ed. by E. R. Griffor, I. Lindström, and V. Stoltenberg-Hansen. Cambridge Tracts in Theoretical Computer Science. Cambridge: Cambridge University Press, 1994, pp. 283–309. ISBN: 978-0-521-06479-8. DOI: 10.1017/CBO9781139166386.013.

[Hyl81] J Martin E Hyland. "Function Spaces in the Category of Locales". In: *Continuous Lattices*. Springer, 1981, pp. 264–281.

[Hyl91] J. M. E. Hyland. "First Steps in Synthetic Domain Theory". In: *Category Theory*. Ed. by Aurelio Carboni, Maria Cristina Pedicchio, and Guiseppe Rosolini. Lecture Notes in Mathematics. Berlin, Heidelberg: Springer, 1991, pp. 131–156. ISBN: 978-3-540-46435-8. DOI: 10.1007/BFb0084217.

[Joh02] Peter T. Johnstone. *Stone Spaces*. Cambridge: Cambridge Univ. Press, 2002. ISBN: 978-0-521-33779-3.

[Joh79] P. T. Johnstone. "On a Topological Topos". In: *Proceedings of the London Mathematical Society* s3-38.2 (Mar. 1, 1979), pp. 237–271. ISSN: 0024-6115. DOI: 10.1112/plms/s3-38.2.237.

[Joh81] P. Johnstone. "Tychonoff's theorem without the axiom of choice". In: *Fundamenta Mathematicae* 113 (1981), pp. 21–35. ISSN: 0016-2736, 1730-6329. DOI: 10.4064/fm-113-1-21-35.

[Joh83] Peter T. Johnstone. "The Point of Pointless Topology". In: *Bulletin of the American Mathematical Society* 8.1 (1983), pp. 41–53. ISSN: 0273-0979, 1088-9485. DOI: 10.1090/S0273-0979-1983-15080-2.

[Kel75] John L. Kelley. *General Topology*. New York, 1975.

[Kna18] Cory M. Knapp. "Partial Functions and Recursion in Univalent Type Theory". PhD thesis. University of Birmingham, Dec. 2018. 246 pp. URL: https://etheses.bham.ac.uk/id/eprint/8448/ (visited on 02/01/2021).

[Koc06] Anders Kock. *Synthetic Differential Geometry*. Vol. 333. Cambridge University Press, 2006.

[Kre59] Georg Kreisel. "Interpretation of Analysis by Means of Constructive Functionals of Finite Types". In: (1959).

[Leš10] Davorin Lešnik. "Synthetic Topology and Constructive Metric Spaces". Ljubljana: University of Ljubljana, 2010.

[LN15] John Longley and Dag Normann. "Historical Survey". In: *Higher-Order Computability*. Ed. by John Longley and Dag Normann. Theory and Applications of Computability. Berlin, Heidelberg: Springer, 2015, pp. 35–50. ISBN: 978-3-662-47992-6. DOI: 10.1007/978-3-662-47992-6_2.

[LS86] J. Lambek and P. J. Scott. *Introduction to Higher Order Categorical Logic*. USA: Cambridge University Press, 1986. 293 pp. ISBN: 978-0-521-24665-1.

[Mil77] Robin Milner. "Fully Abstract Models of Typed $\lambda$-Calculi". In: *Theoretical Computer Science* 4.1 (1977), pp. 1–22.

[MM94] Saunders Mac Lane and Ieke Moerdijk. *Sheaves in Geometry and Logic: A First Introduction to Topos Theory*. Universitext. New York: Springer-Verlag, 1994. ISBN: 978-0-387-97710-2. DOI: 10.1007/978-1-4612-0927-0.

[Neg02] Sara Negri. "Continuous Domains as Formal Spaces". In: *Mathematical Structures in Computer Science* 12.1 (Feb. 1, 2002), pp. 19–52. ISSN: 0960-1295. DOI: 10.1017/S0960129501003450.

[nLa] nLab contributors. *Nice Category of Spaces*. nLab. URL: http://nlab-pages.s3.us-east-2.amazonaws.com/nlab/show/nice+category+of+spaces (visited on 01/11/2022).

[Nor06] Dag Normann. *Recursion on the Countable Functionals*. Vol. 811. Springer, 2006.

[Pla66] Richard Platek. "Foundations of Recursion Theory". PhD thesis. Stanford University, 1966. 430 pp.

[Plo77] Gordon D. Plotkin. "LCF Considered as a Programming Language". In: *Theoretical computer science* 5.3 (1977), pp. 223–255.

[Plo83] GD Plotkin. "Pisa Notes". In: *The Pisa Notes* 123 (1983), p. 127.

[Rey20] Georges Rey. "The Analytic/Synthetic Distinction". In: *The Stanford Encyclopedia of Philosophy*. Ed. by Edward N. Zalta. Winter 2020. Metaphysics Research Lab, Stanford University, 2020. URL: https://plato.stanford.edu/archives/win2020/entries/analytic-synthetic/ (visited on 01/16/2022).

[Ros86] Giuseppe Rosolini. “Continuity and Effectiveness in Topoi”. University of Oxford, Mar. 28, 1986.

[Sco69] Dana S. Scott. “A Type-Theoretical Alternative to ISWIM, CUCH, OWHY”. Unpublished note. Unpublished note. 1969.

[Sco93] Dana S. Scott. “A Type-Theoretical Alternative to ISWIM, CUCH, OWHY”. In: *Theoretical Computer Science* 121.1 (Dec. 6, 1993), pp. 411–440. ISSN: 0304-3975. DOI: 10.1016/0304-3975(93)90095-B.

[Smy77] M. B. Smyth. “Effectively given Domains”. In: *Theoretical Computer Science* 5.3 (Dec. 1, 1977), pp. 257–274. ISSN: 0304-3975. DOI: 10.1016/0304-3975(77)90045-7.

[Smy78] M. B. Smyth. “Power Domains”. In: *Journal of Computer and System Sciences* 16.1 (Feb. 1, 1978), pp. 23–36. ISSN: 0022-0000. DOI: 10.1016/0022-0000(78)90048-X.

[Smy83] M. B. Smyth. “Power Domains and Predicate Transformers: A Topological View”. In: *Automata, Languages and Programming*. Ed. by Josep Diaz. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 1983, pp. 662–675. ISBN: 978-3-540-40038-7. DOI: 10.1007/BFb0036946.

[Smy86] M. B. Smyth. “Finite Approximation of Spaces”. In: *Category Theory and Computer Programming: Tutorial and Workshop, Guildford, U.K. September 16–20, 1985 Proceedings*. Ed. by David Pitt et al. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 1986, pp. 225–241. ISBN: 978-3-540-47213-1. DOI: 10.1007/3-540-17162-2_125.

[Smy93] Michael B. Smyth. “Topology”. In: *Handbook of Logic in Computer Science (Vol. 1)*. USA: Oxford University Press, Inc., 1993, pp. 641–761. ISBN: 0-19-853735-2.

[Spa63] Edwin Spanier. “Quasi-Topologies”. In: *Duke Mathematical Journal* 30.1 (1963), pp. 1–14.

[Sto36] M. H. Stone. “The Theory of Representation for Boolean Algebras”. In: *Transactions of the American Mathematical Society* 40.1 (1936), pp. 37–111. ISSN: 0002-9947. DOI: 10.2307/1989664. JSTOR: 1989664.

[Str06] Thomas Streicher. *Domain-Theoretic Foundations of Functional Programming*. 2006.

[Tar35] Alfred Tarski. “Zur Grundlegung der Boole’schen Algebra. I”. In: *Fundamenta Mathematicae* 1.24 (1935), pp. 177–198. ISSN: 0016-2736. URL: https://www.infona.pl//resource/bwmeta1.element.bwnjournal-article-fmv24i1p19bwm (visited on 01/03/2022).

[Tay03] Paul Taylor. *Practical Foundations of Mathematics*. Cambridge: Cambridge University Press, 2003. ISBN: 978-0-521-07043-0.

[Tos20] Ayberk Tosun. *Formal Topology in Univalent Foundations*. Chalmers University of Technology, 2020. URL: https://odr.chalmers.se/handle/20.500.12380/301098 (visited on 01/18/2021).

[Tyc30] A. Tychonoff. “Über die topologische Erweiterung von Räumen”. In: *Mathematische Annalen* 102.1 (Dec. 1930), pp. 544–561. ISSN: 0025-5831, 1432-1807. DOI: 10.1007/BF01782364.

[Vic05] Steven Vickers. “Some Constructive Roads to Tychonoff”. In: *From Sets and Types to Topology and Analysis: Towards Practicable Foundations for Constructive Mathematics*. Ed. by Laura Crosilla and Peter Schuster. Red. by Dov M. Gabbay, Angus J. MacIntyre, and Dana S. Scott. Oxford Logic Guides 48. Oxford Science Publications, Dec. 2005, pp. 223–237. ISBN: 978-0-19-856651-9.

[Win85] Glynn Winskel. “On Powerdomains and Modality”. In: *Theoretical Computer Science* 36 (Jan. 1, 1985), pp. 127–137. ISSN: 0304-3975. DOI: 10.1016/0304-3975(85)90037-4.

[XE13] Chuangjie Xu and Martín Escardó. “A Constructive Model of Uniform Continuity”. In: *Typed Lambda Calculi and Applications*. Ed. by Masahito Hasegawa. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2013, pp. 236–249. ISBN: 978-3-642-38946-7. DOI: 10.1007/978-3-642-38946-7_18.

[Xu15] Chuangjie Xu. “A Continuous Computational Interpretation of Type Theories”. PhD thesis. University of Birmingham, July 2015. 127 pp. URL: https://etheses.bham.ac.uk/id/eprint/5967/ (visited on 01/31/2021).