A Study of Higher-order Sequential Computation

Shai Geva

August 1995

CMU-CS-95-163

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Thesis Committee:

Stephen Brookes, Chair
John Reynolds
Dana Scott
Prakash Panangaden, McGill University

This research was sponsored by the Office of Naval Research under grant no. N00014-93-I-0750

The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of ONR or the US Government.

Keywords: Theory, Sequentiality, Semantics, Full abstraction, Applicative (functional) programming, PCF

To Edith and Rachel,
Chana and Uri,
Asaf, Eyal,
and Diklah.

לאדית ורחל,
חנה ואורי,
אסף, איל,
ודקלה.

Acknowledgments

I thank with great pleasure the people whose help and support has helped me through the course of my graduate studies and completion of this dissertation.

The School of Computer Science at CMU provided a supporting and stimulating environment, and I thank all the people who made and continue to make it the special place that it is.

The members of my thesis committee have been most patient and encouraging. I deeply thank John Reynolds and Dana Scott for serving on the committee. I have greatly enjoyed discussions with Prakash Panangaden, who always contributed a fresh and interesting point of view.

Steve Brookes has been my advisor, and has closely guided my graduate career. I chose to work with Steve after taking his course on semantics of programming languages, and being impressed with his clear and elegant presentation. Since then, Steve has taught me a lot of what I now know in the area of semantics and in the art of scientific research. I have constantly enjoyed our work together and the synergy that we have developed. I wish to thank Steve for his faith and efforts and, above all, for his close friendship.

My first introduction to the problem of full abstraction for PCF was in the first course in semantics that I took with Steve, and the problem has accompanied me since. My work in the area is best seen as incremental progress, and I am glad to acknowledge my debt to previous efforts. Specifically, work by Kahn, Plotkin, Berry and Curien has been the major influence on my approach.

I am grateful for discussions with friends and colleagues that have helped improve the material in this thesis and its presentation. In particular, discussions with Boris Trakhtenbrot over the last year have helped me to sharpen the introductory material, and have also improved my understanding and appreciation of the work of the Siberian school. A last occasion to polish the presentation was afforded by the workshop on full abstraction for PCF and related languages, held in Århus in April 1995. I thank the organizers and participants of the workshop for a wonderful concentrated dose of modern and vintage work on full abstraction and sequentiality.

Finally, I wish to thank my parents and my family for their love and support. My wife, Diklah, has been my unofficial co-advisor during all these years, and has an important part in my reaching this milestone. With love and affection, I dedicate this work to her, to our children, to my parents, and to my grandmothers.

Contents

List of Figures

Chapter 1

Introduction

This dissertation studies the abstract characterization of sequential computation. The goal is to be able to develop a theory of sequential computation to answer questions such as "Is a given function f sequential?" Often presented as the full abstraction problem for the language PCF, the characterization of sequential computation, and especially higher-order sequential computation over function spaces, has been an open problem for a long time, and has attracted considerable research and interest.

This chapter gives a brief and informal introduction to the problem and to the proposed approach. Some aspects of previous work are presented here (not necessarily in the original terms). For a comprehensive account of the work on the problem, under its guise as the full abstraction problem for PCF, one should consult Ong's survey article [Ong95], or the second edition of Curien's book [Cur93]. Recent work providing constructions of fully abstract models is presented in [AJM94, HO94, OR95].

The last section of this chapter is a roadmap that describes the organization of the dissertation.

1.1 Sequentiality

Sequentiality does not quite make sense as a property of a set-theoretic function; rather, the question of sequentiality must be posed within some model of computation that seems to require the following attributes: (1) computation is an incremental process of obtaining information, and may be decomposed into atomic constituents, and (2) things may go wrong — a computation may fail (or, diverge), and failure of a sub-computation will lead to failure of a computation. This way, an inessential sub-computation may lead to unwarranted failure of its parent computation.

For example, consider these attributes in a sequential programming language. If we invoke a procedure with several input arguments (for simplicity, assume that they are atomic, *e.g.*, booleans), it will attempt to evaluate its arguments in a certain order. An invocation of the procedure computes some function of the input; it performs some computation over the input, until it obtains information sufficient to determine an output value. The atomic constituents of the computation over the input are the computations of the individual arguments. And divergence of the evaluation of any argument will cause the procedure to diverge[1].

1.1.1 Computation Strategies

Rather than a linear ordering of the arguments by evaluation time, a procedure as above describes a tree, since the value obtained for the first argument to be evaluated may affect the choice of the next argument to be evaluated; at each invocation, the input will determine which path of the tree is followed. This tree is referred to informally as the (sequential) computation strategy of the procedure over its input.

The operational concept of a sequential computation strategy, even without reference to a programming language, underlies (sometimes implicitly) previous definitions of sequentiality [Mil77, Vui73, KP78, Tra85, Saz76c]. Intuitively, a function f (with atomic result) is sequential if it may be computed sequentially, according to some sequential computation strategy. A sequential computation strategy is a tree describing a collection of possible computations over the input domain of the function. At each internal node of the tree is an atomic computation, with outgoing edges corresponding to the possible outcomes of this atomic computation.

Computation of the value of a sequential function at some input according to a sequential computation strategy is then a simple process of walking along a path in the tree from its root to a leaf, carrying out the atomic computations over the input — that is, incrementally obtaining more information about the input — until a leaf is reached. What makes a computation strategy adequate for computing a certain function is that a leaf is reached if, and only if, sufficient information is known about the input to determine an output value.

If each leaf of a computation strategy for a function is tagged with the function's output value at the leaf, then one obtains a combined representation of the function and a computation strategy for the function. An example for such a representation is Berry and Curien's notion of sequential algorithm [BC82, Cur93].

To remain within a framework of deterministic computation, the choice among outgoing edges from an internal node (*i.e.*, between possible outcomes of an atomic computation) is assumed to be uniquely determined by an input, so that each input determines a unique path from the root of the tree up to some node that is either a leaf, or is an internal node that represents an operation that fails for that input.

In fact, we can label each node by the information known about the input if it is reached during computation, starting with no known information at the root, and increasing along paths to the leaves. This labelling is uniquely determined under the requirement for determinism. Every atomic computation step now corresponds to a choice between different ways to increment the information, going from the node to one of its mutually exclusive successors.

The process of computing according to a strategy, as presented so far, can be seen to

[1] This description applies directly to a call-by-name language. A similar effect may be obtained in a call-by-value language if the evaluation of arguments may be delayed so that an invoked procedure is again free to decide which of its arguments to evaluate, and in what order. Alternatively, we may bypass argument-passing issues by considering here an expression of the language, rather than a procedure, and treating it as computing a function of its input, that is, its free variables.

maintain a state. This is just the current node at any point, or may also be represented by the information implied by the node about the value being computed.

While this informal introduction employs computation strategies as a central notion, they are less prominent in the ensuing technical development. This is because the requirement that a function have a sequential computation strategy turns out to be equivalent to the requirement that, given any state of the computation, there is an atomic computation that amounts to a root for a computation strategy for the function, starting at the current state. Following Kahn and Plotkin, our technical development will use the latter formulation. The relationship between the two formulations is discussed in section 3.4.

As a trivial example, the trivial computation strategy consists of a single node, a leaf, corresponding to null information content. This strategy is adequate for a function (with an atomic result) iff the function is constant, *i.e.*, its value doesn't depend at all on its input.

A sequential computation strategy as described here is sequential in that there is a *single*, *atomic*, sub-computation at each internal node, which must be completed before moving on. This establishes a single thread of control. Failure or divergence of a sub-computation (or for that matter, an unanticipated response — one that has no corresponding outgoing edge) will lead to a failure to complete the parent computation. For this reason, a sequential computation strategy will typically *not* represent all possible computations over the input, and therefore not all functions may be computed sequentially.

The quintessential example of a non-sequential function is the parallel-or function por presented by Scott [Sco69]. This function extends logical disjunction of two booleans to undefined values (as may be presented by a diverging computation), and is specified as follows, for every x, whether it has one of the boolean values T or F or is undefined:

$$\begin{array}{rcccl} \mathsf{por}(\mathsf{T}, x) & = & \mathsf{por}(x, \mathsf{T}) & = & \mathsf{T} \\ \mathsf{por}(\mathsf{F}, \mathsf{F}) & = & \mathsf{F}. & & \end{array}$$

Is there a sequential computation strategy for por, under the assumption that the computation of each of its arguments is atomic, and that computation of an undefined value diverges? For any non-trivial sequential computation strategy that one may adopt over two (atomic) arguments, one of the arguments must be chosen to be computed first. If the left argument is chosen, then a computation of the right argument alone is excluded, and the strategy will not compute por when computation of the left argument fails, even if the right argument is T; and symmetrically if the right argument is computed first. Therefore, no sequential strategy can compute parallel-or. Parallel-or may be computed, however, given the right parallel computation facilities: both arguments may be computed in parallel, with preemption of one of the computations if the other yields T; and since this parallel computation may be simulated on a sequential processor — a Turing machine, if so desired — parallel-or is an effectively computable function.

1.1.2 Connection with Effective Computability

At this point it is perhaps appropriate to mention the conceptual analogy between a characterization of the sequentially computable functions and a characterization of the effectively computable functions [Tra85]: in both cases, we are seeking to define a class of functions by considering the ways in which they may be computed. This analogy serves (at the very least) to illustrate the fundamental nature of the problem under consideration.

The two issues are orthogonal to a large degree; parallel-or is an example of a function that is not sequential and yet is effectively computable, and, on the other hand, the characteristic function of an undecidable set of naturals is not effectively computable, but is intuitively sequential, having only one, atomic, argument.

However, when considering both issues simultaneously, it is also necessary to consider their possible interactions. Thus, when judging effectiveness of sequential functions, one should also examine the effectiveness of their computation strategies.

For instance, for a set A of naturals, let the function f_A over pairs of naturals be defined by $f_A(x, y) = 0$ iff x is defined and either $x \in A$ or y is defined. Then f_A is sequential (first evaluate x, and, if it is defined but not in A evaluate y), and f_A is computable for any r.e. A (by simulating a parallel computation with preemption of y and the predicate $x \in A$), but, for A r.e. but not recursive, the computation strategy for f_A is not effectively computable, because following the computation of x, it cannot be decided whether y should be evaluated or not. (This example is due to Trakhtenbrot [Tra85].)

In the present work we consider sequentiality in isolation, and we will not concern ourselves at all with effectiveness of sequential functions or their computation strategies.

1.2 A Search for Structure

By the reasoning above, the parallel-or function is not definable in a sequential programming language, where in fact divergence of a sub-computation leads to divergence of the entire computation, and compound values (such as a pair of arguments) are examined incrementally and sequentially. Thus, if we wish to give a semantic model with a good fit – technically, a fully abstract semantics — for a sequential language we need to eliminate the non-sequential functions from the model. For this reason, the problem of characterizing sequential computation has traditionally been posed as the problem of finding a fully abstract model for PCF, a paradigmatic sequential programming language.

Of course, we know intuitively when a programming language is sequential[2]. Why has it proved difficult to transfer this understanding of sequentiality at the linguistic level to an abstract characterization of sequential computation?

The abstractness of the desired characterization implies that all unnecessary detail be discarded. But for talking about sequentiality of computation one needs specialized structure. For instance, one may wish to talk of argument positions or evaluation orders, as we have done above. And in fact, the early definitions of sequentiality [Mil77, Vui73, Saz76c] used such notions. But in abstract settings there may be no discernible argument positions to talk about; what will take their place? What should the atomic computations be? In my

[2] The intuitions about sequentiality of languages can be formalized. For PCF see, *e.g.*, the activity lemma [Plo77], or the syntactic sequentiality theorem [Ber79] which asserts the sequentiality of the evaluation function. Note that only first-order sequentiality is needed when computing over the terms of a language, so that first-order sequentiality of the evaluation function may be proposed as an uncontroversial general criterion for the sequentiality of a programming language.

opinion, the difficulty of the problem stems from the difficulty of identifying the structure that is necessary to model sequentiality adequately.

As an analogy, consider the property of monotonicity. This property formalizes the intuition that if a function is given more input information, then it must produce the same output information as before, and possibly more. This property cannot be stated for a set-theoretical function $f : A \to A'$ defined between two sets. Instead, one must look for structure adequate for formalizing the property. Of course, it is well known that this can be done by working with a partial order of information content over (total and partial) values, so that monotonicity is defined for functions between posets, and captures the desired intuition. So now we may ask: what structure must be added to sets that will make it possible to give a formal definition of our intuitive understanding of sequentiality?

1.2.1 First-order Sequentiality

Kahn and Plotkin's definitions of concrete domains and sequential functions between concrete domains [KP78] have identified structure necessary for formalizing first-order sequential computation. Their definition generalizes earlier definitions of sequentiality [Mil77, Vui73], which have been given for functions from products of flat posets to a flat poset, employing explicitly argument positions to represent the atomic sub-computations.

A first contribution of Kahn and Plotkin's definition of sequential functions is the introduction of a demand-driven view of computation, enabling a definition of sequentiality for more general co-domains. Roughly, if we start with a definition of sequential functions from product of flat posets to a flat poset, this makes it possible to define sequential functions from products of posets to products of posets, by saying that $f : B_1 \times \cdots \times B_n \to B'_1 \times \cdots \times B'_m$ is sequential iff each of f's components $\pi_i \circ f : B_1 \times \cdots \times B_n \to B'_i$, for $i \leq m$, is sequential. The rationale is that under demand-driven computation, the function f need not compute the entire output value fx when applied to some input x, but rather, it only needs to produce a component of fx when prompted for that component; therefore, it should have a sequential computation strategy for computing each component of the answer — but this is equivalent to sequentiality of each of the components $\pi_i \circ f$.

Note that, once we employ distinct computations over the input to answer different demands, the state of a computation must be refined to include not only the node of the computation strategy (or its label), but, moreover, it needs to indicate the demand that is currently driving the computation. Note also that we have said nothing to prevent several demands being presented in parallel and prompting parallel computation. This depends on the assumptions that can be made about the environment that poses the demands, and on the details of the operational behavior. At any rate, the computation corresponding to a single demand can be carried out sequentially.

A second contribution by Kahn and Plotkin was the identification of a class of posets — concrete domains — that can serve as domains of data. The class contains the flat domains and is closed under product and sum. Moreover, Kahn and Plotkin identified the notion of *cells* of concrete domains (or equivalently, concrete data structures), which serve as an abstract substitute for the components of a product. Thus, in their definition of sequential functions — which we may hence call KP-sequential functions — it is the cells that serve as demands in the demand-driven computation. The sequential functions between concrete domains are then those functions that may be computed by a demand-driven sequential computation, or, in other words, functions that may produce a reply to any demand by a sequential computation over their input.

1.2.2 Who Defines Higher-order Sequentiality?

It seems that the intuitions regarding first-order sequential computation are uncontroversial, and Kahn and Plotkin's formalization is generally accepted as being satisfactory. But as we go to higher order, it is necessary to pause and ask whether we are trying to formalize a well posed, even if informal, concept. Moreover, we can ask what, if any, will be the yardstick according to which a candidate definition will be measured for success.

There seems to be one such yardstick in use, namely, the PCF language (as a representative of a family of applicative languages) and the criterion for achieving a good fit is the construction of a fully abstract model for PCF. PCF presents a clear and formal notion of higher-order sequentiality, and this will serve as our guideline for the purpose of characterizing higher-order sequential computation.

We do not delve here into the details of full abstraction. For our purposes, it is sufficient to note Milner's theorem that full abstraction of a (continuous, order-extensional) model for PCF is equivalent to definability of the isolated (finite) elements of the model [Mil77]; thus, our goal will be to construct a model in a way which guarantees definability.

But it is natural to ask whether there is sufficient ground to adopt the "Church-Turing thesis for sequentiality" [Ong95], that anoints PCF-sequentiality as providing the canonical characterization of higher-order sequential computation, or whether there are any other contenders for a notion of higher-order sequential computation.

One contender that could be suggested is Berry and Curien's framework of sequential algorithms between concrete domains, and the associated language CDS0 [BC82, BC85, Cur93]. This model does incorporate a notion of higher-order computation over an algorithm space; but it employs for this computation the Kahn-Plotkin sequentiality. Thus it seems that, rather than defining an alternative notion of higher-order sequential computation, this framework only shows that algorithms, like terms or other syntactic objects, are to be considered as data, as opposed to computation over functions which is genuinely a higher-order notion.

But then again, ruling out CDS0 as presenting a version of higher-order sequentiality is akin to saying that higher-order computation is that provided by function application, *i.e.*, essentially, PCF. Thus, the real question is, perhaps, when is higher-order genuinely higher-order.

At any rate, apart from these remarks, we will not question here the status of PCF as undisputed arbiter of sequentiality.

1.2.3 Towards Higher Order: The Approach Taken

While Kahn and Plotkin's definition of sequentiality has satisfactorily formalized first-order sequential computation, it has been far from clear what additional structure is required to model higher-order sequential computation. Concrete domains are not closed under the sequential function space. Even in cases where the sequential function space is a concrete domain, it is not appropriate to formalize sequential computation over these function spaces as if they were domains of data (this is shown in section 4.4.2).

A first proposal, then, is to equip domains with a parameterized structure that will help to formalize sequential computation; if a domain is used as a domain of data it will be equipped with a structure equivalent to the cell structure, and if it is used as a function space then a different construction may be employed. A similar approach has been taken by Bucciarelli and Ehrhard, who have equipped domains with a parameterized coherence structure, for precisely the same effect (the relationship to their work is discussed in section 3.6.3).

Having decided to continue previous efforts by attempting to formalize sequential computation within the framework of incremental demand-driven computation, we now propose the following thesis:

> Various computational disciplines and assumptions — such as sequentiality — may be captured by an appropriate choice of *demands* or *queries* that are the atomic computations in an incremental, demand-driven framework.

A single such query, formalized as an *index*, is an atomic computation, and may be issued to gain information about an element of the underlying domain. The collection of available queries is formalized as an *index structure* for the domain.

By restricting the class of queries that may be posed, and restricting our attention to *indexed computation* that is carried out using only valid indices, we are able to mirror in the semantic realm restrictions such as those imposed by sequentiality of a language and its set of primitives. Thus, the search for a satisfactory general characterization of sequential computation is reduced to a search for suitable classes or constructions of index structures. The framework of indexed computation is defined in chapter 3. Once we have set up this framework, we proceed in chapter 4 to show that it generalizes KP-sequentiality. In fact, an appropriate definition of *data indices* serves to show this, and extends the applicability of definitions of first-order sequentiality from concrete domains to the larger class of dI-domains.

To construct indices for function spaces, we observe that information about a function space is obtained by application to an argument (and subsequent examination of the output in the co-domain). At first we investigate in chapter 5 a limited notion of such applicative behavior, where the arguments used may only be constant; the resulting model of functions sequential in this sense is fully-abstract for a corresponding restriction of the language PCF.

Subsequently, we explore in chapter 7 the general situation, where the argument used to gain information about a function space is given as a sequential function of the input. We also propose a way to treat the product as if it were a function space. The main ideas here are simple, but their full integration is still a challenge.

1.3 A Roadmap

The dissertation unfolds as follows:

- The next chapter presents necessary technical preliminaries, definitions and notations. The material is mostly well-known, and encompasses PCF, some domain theory and definitions of (Scott-) continuous and stable functions on domains (stability being a property intermediate between continuity and sequentiality), a modicum of generalized topology, and a touch of category theory.

- Chapter 3 defines the framework of *indexed computation*, based on an informal operational model of computation.

  Index structures designate the indices over a domain. Additional components of the index structure relate the indices to the elements of the domain, and show how an index may be decomposed into its prime building blocks, which may be seen as the possible answers to the query posed by the index. Index structures are required to have satisfactory separation properties, and respect the information order on the domain.

  Indexable functions are then defined over *indexed domains* — domains equipped with index structures. These are essentially functions that may be computed using only the available indices, in a demand-driven way. We show that a stability property of the index structures guarantees stability of the indexable functions, and we show some closure properties of the indexable function space.

  Indexable functions may be equivalently defined topologically, as the functions continuous with respect to a generalized topology of *indexable open* sets. Yet another characterization is by preservation of criticality and meets of certain *critical sets*.

- Chapter 4 defines *data indices*. These are stable indices that formalize first-order computation. Indexability with respect to data indices, or *data sequentiality*, is shown to be a conservative extension of previous definitions of sequentiality by Kahn and Plotkin, Milner and Vuillemin.

  By examining the definition of data indices over a product of domains, we identify an important *product assumption*: that an increment in the information over a product corresponds to an increment in precisely one of the components. Adoption of this assumption leads to an easy definition of a simple product of indexed domains.

  However, when we start to consider the treatment of sequentiality at higher order, we find out that data indices are not adequate for formalizing computation over function spaces, and, significantly, the product assumption is *not* adequate any more.

  We remark that sequentiality of functions must be judged by looking at their *fully uncurried* forms.

- As a gentle introduction to a general treatment of higher-order sequentiality, chapter 5 treats a restricted notion of higher-order computation, where computation over function spaces proceeds by observing the result of an application to a fixed argument. Such a fixed argument, called a *prompter*, and a *residual index* used to gain information about the result of the application define together a *constant-applicative index*.

  Using constant-applicative indices over function spaces, together with data indices on domains of data and the simple product of indexed domains, we define a fully abstract model for a correspondingly restricted sub-language of PCF, where a variable of arrow type may only be applied to closed, constant terms.

Full abstraction and the definability of isolated elements of the constant-applicative model show that, at least in this instance, index structures provide the structure necessary for an abstract characterization of sequentiality.

We remark that the understanding afforded by the proof points out a discipline of extending the interpretation of ground types of PCF beyond flat domains, by evaluating in a demand-driven way, similar to Berry and Curien's CDS0 framework.

- Chapter 6 is devoted to introducing bi-ordered structures. The bulk of this chapter summarizes Berry's development of bidomains and stable functions between them. This is largely technical but necessary, because Berry has shown that the fully abstract model of PCF is a model of bidomains. Essentially, bidomains have two orders so that both the pointwise and the stable orders on a function space may be used to restrict the computations over the function space.

  Following this, we extend the framework of indexed computation to bidomains, and show that closure results for indexable function spaces lift cleanly.

  We observe that bi-ordered structures may be seen to be generated when the approximation order induced by an index structure (this would be the pointwise order over a function space) fails to coincide with the original intrinsic information order (the stable order over a function space). When this happens, we say that the index structure is not *conformal*. In a sense, this is saying nothing new: that the observational capabilities of application are limited to the extensional aspects.

- Chapter 7 tackles the unrestricted higher-order problem. By generalizing the constant prompters of constant-applicative indices to prompter functions, that depend on the entire input to the computation, we are able to define *applicative indices* over the function space. Since the indices used to define sequentiality now depend themselves on sequential functions, the definition must be given hierarchically.

  Since the simple product is not adequate at this point, we need to look for some other definition of product. At this point we argue the need to view all domains, including the product, as sequential function spaces, equipped with applicative indices. This is done by equipping every domain with an applicative structure that specifies its source and target space, when its elements are viewed as fully uncurried functions. The objects obtained are referred to as *spaces*, and they are also defined hierarchically, starting with domains of data. The functions of interest in this construction are the *fully indexable* functions, essentially functions whose fully uncurried form is indexable with respect to the applicative index structure.

  A proof that this in fact is a model for PCF and a cartesian-closed category proceeds along unconventional lines, due to the structuring of the definitions. (As of this writing, compositionality of fully indexable functions is left as a conjecture that we have not been able to prove or disprove; we discuss the difficulties involved and the significance.)

  The chapter concludes by indicating further refinements necessary to obtain a fully abstract model for PCF.

- Chapter 8 offers concluding remarks. We briefly survey some of the main points presented, outline some of the connections to related work, and propose directions for continued work.

Chapter 2

Preliminaries

This chapter presents some notations, conventions, and basic definitions and technical development. The material is mostly well known, except perhaps for some new material in the section on topological characterizations (section 2.2), especially the sub-sections concerning the stable generalized topology (2.2.4 and 2.2.5). We start with some notational conventions.

We denote application by juxtaposition, as in fx for the application of a map $f : A \to A'$ to an element $x \in A$. We let application associate to the left, so that $fxy = (fx)y$.

We use the usual λ-notation; $\lambda x \in A \,.\, f(x)$ is a notation for the function f over A. Typically, the occurrence of $f(x)$ will be an expression in our meta-language. We may also use $x \mapsto f(x)$ for denoting f. For functions over products we may use the notation $\lambda(x_1, \ldots, x_n) \in A_1 \times \ldots \times A_n \,.\, f(x_1, \ldots, x_n)$ with the obvious meaning, *i.e.*, x_i is the i'th component of the input to the function.

For a map $f : A \to A'$ we write fX for the image $\{fx \mid x \in X\}$ of $X \subseteq A$ by f, also referred to as the elementwise or pointwise application of f to X. We write $f^{-1}X'$ for the inverse image $\{x \in A \mid fx \in X'\}$ by f of $X' \subseteq A'$.

We also extend binary operations and relations pointwise to sets, *e.g.*, for $R \subseteq A_1 \times A_2$ and $X \subseteq A_1$, we write XRy as a shorthand for the assertion $\forall x \in X \,.\, xRy$, and for a binary operation $\odot : A_1 \times A_2 \to A$ we write $X \odot y$ as a shorthand for the set $\{x \odot y \mid x \in X\}$.

We fix arguments to binary relations and infix binary operations as follows: For a binary relation $R \subseteq A_1 \times A_2$ and $x \in A_1$ we write (xR) for the set $\{y \in A_2 \mid xRy\}$; for an infix binary operation $\odot : A_1 \times A_2 \to A$ we write $(x \odot\ )$ for the function $\lambda y \in A_2 \,.\, x \odot y$. The analogous notation serves to fix the right-hand argument.

We write id_A for the identity function $\lambda x \in A \,.\, x$ over a set A. For sets A and A' and $y \in A'$, we write $\mathsf{K}_{A,A'}y$ for the constant y function $\lambda x \,.\, y$ from A to A', *i.e.*, $\mathsf{K}yx = y$ for every $x \in A$. We will freely drop the qualifying sub- and superscripts for these and other notations when no ambiguity results.

We let the arrow constructor associate to the right, so that, for instance, $A_1 \to A_2 \to A_3$ is to be read as $A_1 \to (A_2 \to A_3)$.

For a function $f : A \to A$ over some set A, we define inductively the n-times repeated composition f^n of f:

$$\begin{aligned} f^0 &= \mathsf{id}_A \\ f^{n+1} &= f^n \circ f. \end{aligned}$$

$f^n x$ is then the n-times repeated application of f to $x \in A$.

2.1 Domains

This section presents the necessary basics of the theory of domains, continuous functions, and stable functions.

2.1.1 Partial Orders

A ***partial order*** on a set A is a binary relation $\leq$ on A that is reflexive, transitive and anti-symmetric (for every $x, y, z \in A$, (1) $x \leq x$, (2) $x \leq y \;\;\&\;\; y \leq z \implies x \leq z$, and (3) $x \leq y \;\;\&\;\; y \leq x \implies x = y$). A ***partially ordered set*** or a ***poset*** $B = (A, \leq_B)$ is a set A together with a partial order $\leq_B$ on A. For a poset $B = (A, \leq_B)$, as well as for other sorts of sets with additional structure, we may write B for its underlying set A, and if the context is not ambiguous we may drop the qualifying subscript from the order or other constituents of the structure.

If $x \leq y$ we say that y ***dominates*** x, or that x ***approximates*** y. We may picture the order as increasing from bottom to top, so that we may talk of x being below y and y being above x. The ***strict order*** $<$ is defined by: $x < y$ iff $x \leq y$ and $x \neq y$. The ***covering*** relation $\prec$ on B is defined by: $x \prec y$ iff $x < y$ and there is no element strictly between x and y, *i.e.*, $\{z \in B \mid x < z \;\;\&\;\; z < y\}$ is empty.

The ***upper set*** of $X \subseteq B$ is $\uparrow X = \{y \in B \mid \exists x \in X \,.\, x \leq y\}$, and the upper set of $x \in B$ is $\uparrow x = \uparrow \{x\}$. $X \subseteq B$ is ***up-closed*** iff $X = \uparrow X$. An ***upper bound*** of $X \subseteq B$ is some $y \in B$ that dominates every $x \in X$. If X has an upper bound it is said to be ***upper bounded*** or just ***bounded*** or ***consistent***, written as $\Uparrow X$. We may also use the consistency predicate as a binary relation, writing $x \Uparrow y$ iff $\Uparrow \{x, y\}$. A ***least upper bound*** or ***lub*** of $X \subseteq B$, denoted by $\bigvee X$, is an upper bound of X that is dominated by every other upper bound of X. By anti-symmetry, the least upper bound of X is uniquely determined, if it exists. We define dually the notions of a ***lower set*** $\downarrow X = \{y \in B \mid \exists x \in X \,.\, y \leq x\}$, ***down-closed*** sets, ***lower bounds***, and ***greatest lower bound*** or ***glb*** $\bigwedge X$,

As a convention, we reserve the term ***join*** exclusively for the lub operation over sets that are *non-empty* and *finite*, denoted $\vee$. The join may be used as a binary operation, defined by $x \vee y = \vee \{x, y\}$ when it exists. Similarly, we reserve the term ***meet*** for the glb operation over *non-empty finite* sets or pairs, denoted $\wedge$. A poset is said to ***have meets*** iff every pair (or, equivalently, every non-empty finite set) of elements has a meet. A poset is said to ***have consistent joins*** iff every consistent pair (or, equivalently, every consistent non-empty finite set) of elements has a join. We may similarly talk of posets with bounded meets, *etc.*

We use several families of symbols for different orders: $\leq$, $\leq^\bullet$, $\leq^\star$, $\subseteq$ (usual set inclusion), $\sqsubseteq$, *etc.* The notations introduced here for $\leq$ will also apply to these families, with appropriate typographic changes and qualifications.

A ***chain*** is a subset $X \subseteq B$ that is a linear order under the restriction of the partial order; *i.e.*, either $x \leq y$ or $y \leq x$ for every $x, y \in X$. An ω-chain is a countable chain. A chain $x_1, x_2, \ldots$ is ***increasing*** if $x_i \leq x_{i+1}$ for every i, and ***decreasing*** if the dual holds.

$X \subseteq B$ is ***directed*** iff it is non-empty and every pair (or, equivalently, non-empty finite set) of elements of X has an upper bound in X. X is ***down-directed*** iff it is non-empty and every pair of elements of X has a *lower* bound in X. A ***filter*** of B is a subset $X \subseteq B$ that is up-closed and down-directed. The ***principal filters*** of B are the sets $\uparrow x$ for $x \in B$.

A poset is a ***directed-complete partial order***, or a ***dcpo***, iff every directed set has a lub. A poset is ***bounded-complete*** iff every non-empty consistent set has a lub. A ***bc-dcpo*** is a bounded-complete dcpo. A poset is ***coherent*** if every non-empty pairwise-consistent subset has a lub; a subset X is pairwise-consistent if every pair of elements of X is consistent. A poset B is ***pointed*** if it has a least element, referred to as ***bottom*** and denoted $\bot_B$ or just $\bot$; by anti-symmetry a least element is unique, if it exists. The empty set is consistent in a non-empty poset, and has bottom for its lub in a pointed poset[1].

2.1.1 Proposition

(1) A pointed poset B is bounded-complete iff every non-empty set has a glb.

(2) A pointed dcpo B is bounded-complete iff it has consistent joins.

Proof:

(1) If a set $X \subseteq B$ is consistent then the set Y of its upper bounds is non-empty, and if Y has a glb then it is the lub of X.

If a set $Y \subseteq B$ is non-empty then the set X of its lower bounds is consistent, and it is non-empty since the poset is pointed. If X has a lub then it is the glb of Y.

(2) Bounded completeness implies existence of consistent joins. Conversely, if a set $Y \subseteq B$ is non-empty and B has consistent joins then for every two lower bounds x and y of Y, $x \Uparrow y$, and $x \vee y$ exists and is itself a lower bound of Y. Bottom is a lower bound of Y, so that the set X of lower bounds of Y is non-empty and directed. By directed completeness, X has a lub, which is the glb of Y. Thus, every non-empty set has a glb, which is equivalent by (1) to bounded-completeness of B. ∎

An element x of a poset B is ***isolated*** iff, whenever it is dominated by a lub $\bigvee Y$ of some directed set Y, there exists some $y \in Y$ that dominates x. We write $\mathbf{K}\,B$ for the set of isolated elements of B, and for any subset $X \subseteq B$ we write $\mathbf{K}_B\,X$ for $X \cap \mathbf{K}\,B$, dropping the subscript when possible. (Isolated elements are also commonly called *finite* or *compact* elements.) A dcpo B is ***algebraic*** iff for every $x \in B$ the set of isolated elements below x is directed, and has x as its lub. A dcpo is ***ω-algebraic*** iff it is algebraic and $\mathbf{K}\,B$ is countable.

A poset has ***property (I)*** iff every isolated element dominates finitely many isolated elements. A poset is ***distributive*** iff it is bounded-complete and for every x and consistent pair $y \Uparrow z$,

$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z).$$

[1] In many formulations dcpos or posets in general are required to be non-empty, and bounded-complete posets are required to be pointed, so that *all* consistent sets, including the empty set, have a lub. A dcpo is also referred to in the literature as simply a *complete partial order* or *cpo*, and a bounded-complete poset is also referred to as being *consistently complete*.

A ***Scott domain*** is an ω-algebraic pointed bounded-complete and directed-complete poset. A ***dI-domain***, also called a ***Berry domain***, is a distributive Scott domain with property (I).

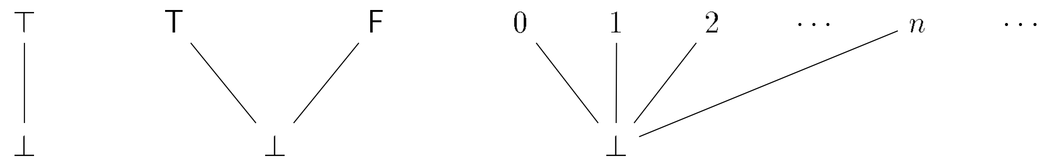

Figure 2.1: The posets **2**, **Bool** and **Nat**.

Typically, we will think of the partial order over a set as ordering the elements by their information content, making it possible to model partial information. A discrete partial order (where no distinct elements are related) would represent a set of elements that have a maximal or total information content. If one adjoins to such a set a new element, dominated by all existing elements, then the resulting poset — called a ***flat poset*** or a ***flat domain*** represents an all-or-nothing proposition: either there is no information, represented by the bottom element, or there is complete information, represented by one of the non-bottom, or ***proper*** elements (the information resides in the choice of one among several alternatives).

Some simple examples of flat domains are the poset **2**, the poset **Bool** of booleans or truth values, and the flat poset **Nat** of natural numbers, pictorially presented via their ***Hasse diagrams*** in figure 2.1. It is easy to check that all flat posets (with countably many proper elements) are dI-domains.

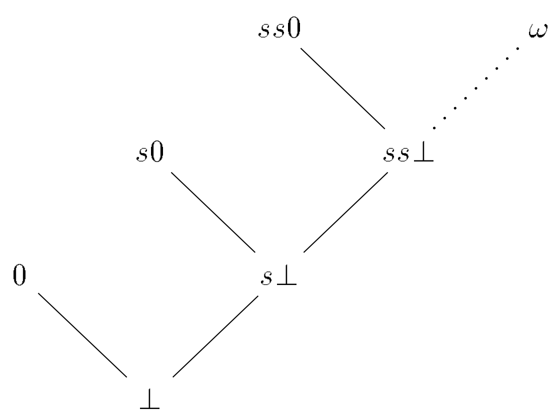

Figure 2.2: The poset of lazy naturals

Another example of a simple dI-domain is the poset **LNat** of *lazy naturals*, shown in figure 2.2.

The isolated elements of a poset can be thought of as elements with a finite content of information; this is especially true for posets with property (I), where finiteness of information content is identified with dominating a finite number of elements. Algebraicity of a poset guarantees that every element is (uniquely) obtained from its finite approximations, and hence the isolated elements may be seen as forming a basis.

A finite directed set always has a maximal element, so that finite posets are directed-complete and algebraic, with all of their elements isolated.

2.1.2 Product of Sets and Posets

The ***cartesian product*** $A_1 \times A_2$ of sets A_1 and A_2 consists of the ordered pairs (x_1, x_2) of $x_1 \in A_1$ and $x_2 \in A_2$; that is,

$$A_1 \times A_2 = \{(x_1, x_2) \mid x_1 \in A_1 \ \ \& \ \ x_2 \in A_2\}\,.$$

Together with the cartesian product of sets A_1 and A_2 we define several related functions and ways of constructing new functions from old. The ***product projections*** — a first projection $\pi_{1;A_1,A_2} : A_1 \times A_2 \to A_1$ and a second projection $\pi_{2;A_1,A_2} : A_1 \times A_2 \to A_2$ — are defined on $(x_1, x_2) \in A_1 \times A_2$ by

$$\begin{array}{rcl} \pi_1(x_1, x_2) & = & x_1 \\ \pi_2(x_1, x_2) & = & x_2 \end{array}\ .$$

For a set A and pair of functions $f_1 : A \to A_1$ and $f_2 : A \to A_2$ we define the ***pairing*** of f_1 and f_2 to be the function $\langle f_1\,, f_2\rangle : A \to A_1 \times A_2$ defined at $x \in A$ by

$$\langle f_1\,, f_2\rangle\, x = (f_1 x, f_2 x).$$

The ***product*** $g_1 \times g_2 : A_1 \times A_2 \to A_1' \times A_2'$ of functions $g_1 : A_1 \to A_1'$ and $g_2 : A_2 \to A_2'$ is defined by setting

$$g_1 \times g_2 = \langle g_1 \circ \pi_{1;A_1,A_2}\,, g_2 \circ \pi_{2;A_1,A_2}\rangle\,.$$

In other words, for $(x_1, x_2) \in A_1 \times A_2$ we set

$$(g_1 \times g_2)(x_1, x_2) = (g_1 x_1, g_2 x_2).$$

The product is associative and commutative up to isomorphism; this means that for all sets A_1, A_2 and A_3, (1) $A_1 \times (A_2 \times A_3)$ and $(A_1 \times A_2) \times A_3$ are isomorphic via the left-associative isomorphism

$$\mathsf{assl}_{A_1,A_2,A_3} = \langle\langle \pi_1\,, \pi_1 \circ \pi_2\rangle\,, \pi_2 \circ \pi_2\rangle : A_1 \times (A_2 \times A_3) \to (A_1 \times A_2) \times A_3,$$

which has for its inverse the right-associative isomorphism

$$\mathsf{assr}_{A_1,A_2,A_3} = \langle \pi_1 \circ \pi_1\,, \langle \pi_2 \circ \pi_1\,, \pi_2\rangle\rangle : (A_1 \times A_2) \times A_3 \to A_1 \times (A_2 \times A_3),$$

and (2) $A_1 \times A_2$ and $A_2 \times A_1$ are isomorphic via the commutative isomorphism

$$\mathsf{comm}_{A_1,A_2} = \langle \pi_2\,, \pi_1\rangle : A_1 \times A_2 \to A_2 \times A_1,$$

which has comm_{A_2,A_1} for an inverse. Since these isomorphisms are built up from the product projections using pairing, they belong to any class of functions that contains the product projections and is closed under pairing of functions.

The product $B_1 \times B_2$ of two posets $B_1 = (A_1, \leq_{B_1})$ and $B_2 = (A_2, \leq_{B_2})$ is taken to be the poset $(A_1 \times A_2, \leq_{B_1 \times B_2})$, that is, the cartesian product of their underlying sets, equipped with the ***componentwise order*** $\leq_{B_1 \times B_2}$ defined by:

$$(x_1, x_2) \leq_{B_1 \times B_2} (y_1, y_2) \quad \text{iff} \quad x_1 \leq_{B_1} y_1 \ \& \ x_2 \leq_{B_2} y_2.$$

2.1.2 Proposition *Upper bounds, lubs, lower bounds, glbs are determined componentwise in a product of posets.*

Projection $\pi_i X$ of a subset $X \subseteq B_1 \times B_2$ of the product preserves directedness and/or consistency of X.

Poset product preserves directed completeness, bounded completeness, distributivity, algebraicity, ω-algebraicity, pointedness and property (I). Also, $\mathbf{K}(B_1 \times B_2) = \mathbf{K}\, B_1 \times \mathbf{K}\, B_2$ and, for pointed posets, $\perp_{B_1 \times B_2} = (\perp_{B_1}, \perp_{B_2})$.

For pointed posets we define canonical counterparts to the product projections: the ***product injections*** are defined for pointed posets B_1 and B_2 by:

$$\begin{array}{rcl} \rotatebox[origin=c]{180}{π}_{1;B_1,B_2} & : & B_1 \to B_1 \times B_2 \\ & = & \langle \mathsf{id}_{B_1}, \mathsf{K}\!\perp_{B_2} \rangle \\ & = & \lambda x \in B_1 \,.\, (x, \perp_{B_2}) \end{array} \qquad \begin{array}{rcl} \rotatebox[origin=c]{180}{π}_{2;B_1,B_2} & : & B_2 \to B_1 \times B_2 \\ & = & \langle \mathsf{K}\!\perp_{B_1}, \mathsf{id}_{B_2} \rangle \\ & = & \lambda x \in B_2 \,.\, (\perp_{B_1}, x). \end{array}$$

The injections provide a natural way to inject a component of the product into the product. (The terminology and notation for the injections are not conventional. I suggest reading the symbols π and $\rotatebox[origin=c]{180}{π}$ as *project* and *inject* respectively.)

2.1.3 Continuous and Stable Functions

A function $f : B \to B'$ between posets is ***monotone*** iff for every $x, y \in B$, $x \leq_B y$ implies that $fx \leq_{B'} fy$. A function $f : B \to B'$ between dcpos is ***Scott continuous***, or just ***continuous***, iff it is monotone and it preserves directed lubs: for every directed $X \subseteq B$, $f(\bigvee X) = \bigvee(fX)$; note that the set $fX = \{fx \mid x \in X\}$ is directed by monotonicity of f, so that its lub is well defined. A function $f : B \to B'$ between dcpos with bounded meets is ***cm-stable***, or just ***stable***, iff it is continuous and it preserves bounded meets: for every $x, y \in B$, if $x \Uparrow y$ then $f(x \wedge y) = fx \wedge fy$; note that $fx \Uparrow fy$ by monotonicity of f, so that their meet is well defined[2]. A function $f : B \to B'$ between pointed posets B and B' is ***strict*** iff it preserves the least element: $f\perp_B = \perp_{B'}$.

[2]The definition of stable functions that we use is Berry's definition of *conditionally multiplicative* functions, or *cm*-functions (also called *consistently multiplicative*). This definition is equivalent to requiring that f be continuous and that for every $x \in B$ and $y \leq_{B'} fx$, the set $\{x' \leq_B x \mid y \leq_{B'} fx'\}$ be down-directed.

A function $f : B \to B'$ between dcpos is stable in Berry's sense iff it is continuous and the set $\{x' \leq_B x \mid y \leq_{B'} fx'\}$ has a least element, denoted $M(f, x, y)$, for every $x \in B$ and $y \leq_{B'} fx$. This *minimum point* definition and the cm definition of stability are not equivalent in general, but coincide on dI-domains, where they are of particular relevance.

Of the functions we have seen so far, it is easy to verify that, over appropriate posets[3], the identity functions, constant functions, the product projections and the product injections are all monotone, continuous and stable and all but the constant functions are strict; the only strict constant function is the constant bottom function $\mathsf{K}\bot$. Moreover, each of the classes of monotone, continuous, stable and strict functions is closed under composition, pairing and product of functions.

Directed completeness and pointedness guarantee the existence of least fixed points for continuous functions, playing a central rôle in the semantic treatment of recursion. For B a pointed dcpo and a continuous $f : B \to B$ define

$$\mathsf{fix}\, f = \bigvee_B \{f^n \bot_B \mid n \in \mathbb{N}\}.$$

The existence of the lub for this increasing chain follows from directed completeness. It can be shown that $\mathsf{fix}\, f$ is a fixed point of f, *i.e.*, $f(\mathsf{fix}\, f) = \mathsf{fix}\, f$, and moreover, $\mathsf{fix}\, f$ is dominated by every fixed point of f.

Note that if $f : B \to B'$ is a monotone function between posets and B is finite then f is continuous as well.

For an example of a function that is continuous but not stable we point to the parallel-or function $\mathsf{por} : \mathbf{Bool} \times \mathbf{Bool} \to \mathbf{Bool}$ which is taken to be the unique monotone function of this type such that

$$\begin{array}{lclcl} \mathsf{por}(\mathsf{T}, \bot) & = & \mathsf{por}(\bot, \mathsf{T}) & = & \mathsf{T} \\ \mathsf{por}(\mathsf{F}, \mathsf{F}) & = & \mathsf{F}. & & \end{array}$$

Stability fails at the consistent pair $(\mathsf{T}, \bot) \Uparrow (\bot, \mathsf{T})$:

$$\mathsf{por}((\mathsf{T}, \bot) \wedge (\bot, \mathsf{T})) = \mathsf{por}(\bot, \bot) = \bot < \mathsf{T} = \mathsf{por}(\mathsf{T}, \bot) \wedge \mathsf{por}(\bot, \mathsf{T}).$$

Functions Over a Product

When a function is defined over a product, *e.g.*, $f : B_1 \times B_2 \to B$, it may be seen both as a function of one argument, which is a pair $(x_1, x_2) \in B_1 \times B_2$, or as a function of two arguments, $x_1 \in B_1$ and $x_2 \in B_2$. One may then examine properties of the function at one of the components (or arguments). Thus, we say that f is strict in its left (or first) argument iff for every $x_2 \in B_2$ the function $\lambda x_1 \in B_1 \,.\, f(x_1, x_2)$ is strict (and symmetrically for strictness in the right (or second) argument, and similarly for properties other than strictness). We say that the function has a property in both of its arguments (simultaneously) when it has the property in the usual sense as a function of one argument (which happens to be a pair).

It is easy to check that if a function defined over a product is strict in one of its arguments then it is strict (in both arguments simultaneously), but the converse is not true: parallel-or is strict, but not strict in either of its arguments. The converse situation holds for stability: If a function over a product is stable then it is stable in each of its arguments separately, but parallel-or is stable in each of its arguments separately and it is not stable in both arguments

[3] "Appropriate posets" are necessary, because a property of functions must be examined over a class of structures where the property is meaningfully defined; thus, strictness of a function is assessed assuming that its source and target are pointed posets.

simultaneously. Finally, a function over a product is monotone or continuous, respectively, iff it is monotone or continuous, respectively, in each of its arguments. We show this for continuity.

2.1.3 Proposition *A monotone function $f : B_1 \times B_2 \to B$ between dcpos is continuous iff it is continuous in each of its arguments separately.*

Proof: Let Z be a directed set of $B_1 \times B_2$, and $X = \pi_1 Z$, $Y = \pi_2 Z$. X and Y are easily seen to be directed sets of B_1 and B_2, respectively, and determine componentwise the lub of Z: $\bigvee Z = (\bigvee X, \bigvee Y)$. Observe that if $x \in X$ and $y \in Y$ then, by directedness of Z, there must exist some $(x', y') \in Z$ with $(x, y) \leq (x', y')$.

By monotonicity, $\bigvee(fZ) \leq f(\bigvee Z)$. If we now assume that f is continuous in each argument separately, we have that

$$\begin{array}{rcll} f(\bigvee Z) & = & f(\bigvee X, \bigvee Y) & \\ & = & \bigvee\{f(x, \bigvee Y) \mid x \in X\} & \text{(Continuity in left argument)} \\ & = & \bigvee\{\bigvee\{f(x,y) \mid x \in X\} \mid y \in Y\} & \text{(Continuity in right argument)} \\ & = & \bigvee\{f(x,y) \mid x \in X \ \ \& \ \ y \in Y\} & \text{(A property of lub)} \\ & \leq & \bigvee\{f(x',y') \mid (x',y') \in Z\} & \text{(Observation above)} \\ & = & \bigvee(fZ), & \end{array}$$

and we may conclude that f is continuous. Conversely, if f is continuous then it is easily seen to be continuous in each argument separately (by considering directed sets that are fixed in one of the components of the product). ■

2.1.4 Proposition *The meet function $(\wedge_B) : B \times B \to B$ of an algebraic dcpo with meets B is a continuous function.*

Proof: If $x_1 \leq x_2$ and $y_1 \leq y_2$ then, by definition, $x_1 \wedge y_1 \leq x_2 \wedge y_2$, so that the meet is monotone. To show that the meet is continuous in the left argument let $X \leq B$ be a directed set and fix $y \in B$. By monotonicity, $\bigvee\{x \wedge y \mid x \in X\} \leq (\bigvee X) \wedge y$. For the reverse inequality, let z be an isolated approximation of $(\bigvee X) \wedge y$. Hence $z \leq \bigvee X$, and by isolatedness there exists $x \in X$ such that $z \leq x$. But $z \leq y$, so that $z \leq x \wedge y$, and $z \leq \bigvee\{x \wedge y \mid x \in X\}$. This is true for any isolated approximation z of $(\bigvee X) \wedge y$, and the desired inequality follows by algebraicity. This establishes continuity of the meet in the left argument; continuity in the right argument is shown symmetrically, and continuity of the meet follows from continuity in both arguments separately. ■

We say that a dcpo B ***has continuous meet*** iff its (binary) meet function $(\wedge_B) : B \times B \to B$ is well defined (*i.e.*, the poset has all meets) and continuous. Stability of a continuous meet is assured, even without assumption of algebraicity.

2.1.5 Proposition *If the meet function $(\wedge_B) : B \times B \to B$ of a dcpo B with meets is a continuous function then it is a stable function.*

Proof: For $(x_1, x_2) \Uparrow (y_1, y_2)$, it is a simple property of the meet that

$$(x_1 \wedge y_1) \wedge (x_2 \wedge y_2) = (x_1 \wedge x_2) \wedge (y_1 \wedge y_2).$$

But this is exactly the additional requirement for stability of a continuous $\wedge_B$, since it establishes that

$$(\wedge_B)((x_1, x_2) \wedge (y_1, y_2)) = ((\wedge_B)(x_1, x_2)) \wedge ((\wedge_B)(y_1, y_2)).$$

(Recall that meets over $B \times B$ are taken componentwise.) ∎

2.1.4 Function Spaces

The ***full function space*** $A \to A'$ between sets A and A' is the set of all (set-theoretic) functions from A to A'. Other function spaces may be obtained by taking a subset of the full function space; it may be necessary to consider sets with some extra structure for the purpose of identifying the desired functions. At this point we will mainly be interested in the monotone function space between posets, the continuous function space $B \to_{\mathrm{c}} B'$ between dcpos, and the stable function space $B \to_{\mathrm{s}} B'$ between dcpos with bounded meet.

We refer to functions defined over function spaces as functionals or higher-order functions. As a primary example, take application and composition:

$$\begin{array}{rcrcl} \mathsf{app}_{A,A'} & : & (A \to A') \times A & \to & A' \\ & & (f, x) & \mapsto & fx \end{array}$$

and

$$\begin{array}{rcrcl} (\circ_{A,A',A''}) & : & (A' \to A'') \times (A \to A') & \to & (A \to A'') \\ & & (g, f) & \mapsto & g \circ f = \lambda x \in A \,.\, g(fx). \end{array}$$

The fixed point operator is defined over spaces of continuous functions from a pointed dcpo B to itself:

$$\begin{array}{rcrcl} \mathsf{fix}_B & : & (B \to_{\mathrm{c}} B) & \to & B \\ & & f & \mapsto & \mathsf{fix}\, f = \bigvee_B \{f^n \bot_B \mid n \in \mathbb{N}\}\,. \end{array}$$

Functionals related to the product are the pairing and product of functions,

$$\begin{array}{rcrcl} \langle\rangle_{A;A_1,A_2} & : & (A \to A_1) \times (A \to A_2) & \to & (A \to A_1 \times A_2) \\ & & (f_1, f_2) & \mapsto & \langle f_1\,, f_2\rangle \end{array}$$

and

$$\begin{array}{rcrcl} (\times_{A_1,A_2;A_1',A_2'}) & : & (A_1 \to A_1') \times (A_2 \to A_2') & \to & (A_1 \times A_2 \to A_1' \times A_2') \\ & & (f_1, f_2) & \mapsto & f_1 \times f_2. \end{array}$$

The function spaces $A_1 \times A_2 \to A'$ and $A_1 \to (A_2 \to A')$ are isomorphic via the usual Currying and un-Currying maps, which formalize the way in which a function over a product may be seen as a function of two arguments:

$$\begin{array}{rcrcl} \mathsf{curry}_{A_1,A_2,A'} & : & (A_1 \times A_2 \to A') & \to & (A_1 \to A_2 \to A') \\ & & f & \mapsto & \lambda x \in A_1 \,.\, \lambda y \in A_2 \,.\, f(x, y) \\ \mathsf{uncurry}_{A_1,A_2,A'} & : & (A_1 \to A_2 \to A') & \to & (A_1 \times A_2 \to A') \\ & & g & \mapsto & \lambda z \in A_1 \times A_2 \,.\, g(\pi_1\, z)(\pi_2\, z). \end{array}$$

The constant-formation operation is defined by:

$$\begin{array}{rcccl} \mathsf{K}_{A,A'} & : & A' & \to & (A \to A') \\ & & y & \mapsto & \lambda x \in A \,.\, y, \end{array}$$

and $\mathsf{K}_{A,A'} = \mathsf{curry}\, \pi_{1;A',A}$. Here are a few additional identities involving currying and application:

2.1.6 Proposition *For every $f : A_1 \times A_2 \to A'$ and $g : A_1 \to (A_2 \to A')$,*

$$\begin{array}{rcl} \mathsf{app} \circ ((\mathsf{curry}\, f) \times \mathsf{id}) & = & f \\ \mathsf{app} \circ (g \times \mathsf{id}) & = & \mathsf{uncurry}\, g \\ \mathsf{app}_{A,A'} & = & \mathsf{uncurry}\, \mathsf{id}_{A \to A'} \,. \end{array}$$

The ***arrow*** or ***exponentiation*** functional $\to$ is defined by

$$\begin{array}{rcccl} (\to) & : & (A_1' \to A_1) \times (A_2 \to A_2') & \to & (A_1 \to A_2) \to (A_1' \to A_2') \\ & & (f, g) & \mapsto & f \to g = \lambda h \in A_1 \to A_2 \,.\, g \circ h \circ f. \end{array}$$

These functionals may be used unchanged over restricted function spaces over posets, but it is necessary to show that they are well defined; that is, that for operands in the restriction of their source, they produce results that are in the restriction of their target. For instance, the usual function composition is used when restricting attention to continuous function spaces, but it is necessary to show that the composition of two continuous functions is itself a continuous function. This is in fact the case: as we have already asserted, each of the classes of monotone, continuous and stable functions is closed under composition, pairing, and product.

For currying and uncurrying, however, the situation is more complicated. For instance, for a function $f : B_1 \times B_2 \to B'$, monotonicity of $\mathsf{curry}\, f : B_1 \to B_2 \to B'$ cannot be assessed if no partial order is given for $B_2 \to B'$ (or its desired restriction). Similarly, an order is required on the function space $B_2 \to B'$ (or its desired restriction) if we wish to check whether uncurrying preserves, say, the monotonicity of $g : B_1 \to B_2 \to B'$.

Thus, in order to determine closure of classes of functions under currying and uncurrying, and in order to determine whether higher-order functions have properties such as continuity, stability or strictness one must equip the function space with a partial order. This is known as ***order enriching*** the function space. It is then necessary to consider the closure of classes of posets under the order enriched function space construction, so that, for instance, we may be able to rely on directed completeness of a function space to determine continuity or to obtain fixed points. More generally than a partial order, one may need to enrich the function space with additional structure with respect to which the class of functions under consideration is defined.

The Pointwise Order

The ***pointwise order*** $\leq^{\mathrm{p}}_{B,B'}$ (also called the *extensional* order) is defined over the function space $B \to B'$ between posets B and B' by

$$f \leq^{\mathrm{p}}_{B,B'} g \quad \text{iff} \quad \forall x \in B \,.\, fx \leq_{B'} gx.$$

Write $B \to_{\mathrm{c}}^{\mathrm{p}} B'$ and $B \to_{\mathrm{s}}^{\mathrm{p}} B'$ for the pointwise ordered continuous and stable functions spaces, respectively, between B and B'.

For example, the function space $\mathbf{2} \to \mathbf{2}$ consists of the functions $\mathsf{K}\bot$, id and $\mathsf{K}\top$, which form a chain $\mathsf{K}\bot \leq^{\mathrm{p}} \mathsf{id} \leq^{\mathrm{p}} \mathsf{K}\top$ under the pointwise ordering. These functions are all monotone, continuous and stable.

If B' is pointed then the function $\mathsf{K}\bot_{B'}$ — a constant, strict, monotone, continuous and stable function — is least under the pointwise order on $B \to B'$, or any restricted function space to which it belongs. We write $\bot_{B \to B'} = \mathsf{K}\bot_{B'}$.

Associated with the pointwise order are a number of pointwise operations. For instance, the pointwise lub of a family F of functions between posets B and B', denoted $\bigvee_{B,B'}^{\mathrm{p}} F$, is defined pointwise by

$$(\bigvee\nolimits^{\mathrm{p}} F)x = \bigvee\nolimits_{B'}(Fx) = \bigvee\nolimits_{B'} \{fx \mid f \in F\}\,,$$

and its existence depends on the existence of the lubs in B'. Even if the pointwise lub $\bigvee^{\mathrm{p}} F$ is a well defined function, it may not belong to the function space under consideration — it may fail to be continuous or stable — and, if an order other than the pointwise order is used, it may fail to dominate the elements of F. For these reasons, it is important to distinguish between the pointwise lub and the lub in the order-enriched function space; our use of the notation $\bigvee^{\mathrm{p}}$ will always imply the pointwise operation as defined here. The pointwise glb, join and meet operations are defined similarly, and with the same proviso.

The next proposition provides examples of sufficient conditions for the pointwise lub to be well defined, as well as examples of pointwise ordered function spaces where the pointwise lub serves as the directed or bounded lub. Proposition 2.1.8 does the same for the pointwise meet operation.

2.1.7 Proposition *Sufficient conditions for the pointwise lub of a set F of functions between posets B and B' to be well defined are:*

(1) F is pointwise directed and B' is directed complete.

(2) F is pointwise bounded and B' is bounded complete.

The pointwise lub of a pointwise directed set F of continuous functions between dcpos B and B' is a continuous function, and it is the lub of F in $B \to_{\mathrm{c}}^{\mathrm{p}} B'$.

The pointwise lub of a pointwise bounded set F of continuous functions between bc-dcpos B and B' is a continuous function, and it is the lub of F in $B \to_{\mathrm{c}}^{\mathrm{p}} B'$.

The pointwise lub of a pointwise directed set F of stable functions between dcpos B and B' with continuous meet is a stable function, and it is the lub of F in $B \to_{\mathrm{s}}^{\mathrm{p}} B'$.

Proof: Let F be a set of functions between posets B and B'. If F is pointwise directed and B' is directed complete then for every $x \in B$, Fx is a directed set, and $(\bigvee^{\mathrm{p}} F)x = \bigvee_{B'}(Fx)$ is well defined. If F is pointwise bounded (by, say, a function g) and B' is bounded complete then for every $x \in B'$, Fx is a bounded set (with upper bound gx), and $(\bigvee^{\mathrm{p}} F)x = \bigvee_{B'}(Fx)$ is well defined.

When well-defined, set $f_0 = \bigvee^{\mathrm{p}} F$. Clearly, f_0 is least among functions that pointwise dominate all elements of F. Thus, it is the lub of F under the pointwise order on every function space to which it belongs. It now remains to show that if the functions in F are continuous then so is f_0, and if the functions in F are stable then so is f_0.

If all the functions in F are monotone then f_0 is monotone as well: for $x \leq_B y$ we have

$$f_0 x = \bigvee_{B'}(Fx) = \bigvee\{fx \mid f \in F\} \leq_{B'} \bigvee\{fy \mid f \in F\} = \bigvee_{B'}(Fy) = f_0 y.$$

Similarly, if all the functions in F are continuous then f_0 is continuous as well: for a directed set $X \subseteq B$ we have

$$\begin{array}{rcll} f_0(\bigvee_B X) & = & \bigvee_{B'}\{f(\bigvee_B X) \mid f \in F\} & \\ & = & \bigvee_{B'}\{\bigvee_{B'}\{fx \mid x \in X\} \mid f \in F\} & \text{(Continuity of } f \in F\text{)} \\ & = & \bigvee_{B'}\{fx \mid x \in X \;\;\&\;\; f \in F\} & \text{(properties of lub)} \\ & = & \bigvee_{B'}\{\bigvee_{B'}\{fx \mid f \in F\} \mid x \in X\} & \\ & = & \bigvee_{B'}(f_0 X). & \end{array}$$

This development applies both in the case of a pointwise directed set of continuous functions between dcpos, and a pointwise bounded set of continuous functions between bc-dcpos.

Moreover, for a pointwise directed set F of stable functions between dcpos with continuous meet, we have that for all $x \Uparrow_{B'} y$,

$$\begin{array}{rcll} f_0(x \wedge y) & = & \bigvee\{f(x \wedge_B y) \mid f \in F\} & \\ & = & \bigvee\{fx \wedge_{B'} fy \mid f \in F\} & \text{(Stability of } f \in F\text{)} \\ & = & (\bigvee\{fx \mid f \in F\}) \wedge_{B'} (\bigvee\{fy \mid f \in F\}) & (\wedge_{B'} \text{ continuous, } F \text{ directed}) \\ & = & f_0 x \wedge f_0 y, & \end{array}$$

and f_0 is stable. (Note that the directedness of F was used here; in fact the pointwise lub of a pointwise bounded set of stable functions need not be stable in general. See more on this in section 3.7.3.) ∎

2.1.8 Proposition *The pointwise meet of continuous (respectively stable) functions f_1 and f_2 between dcpos B and B' with continuous meet is a continuous function (respectively stable), and it is the meet of f_1 and f_2 in $B \to^{\mathrm{p}}_{\mathrm{c}} B'$ (respectively $B \to^{\mathrm{p}}_{\mathrm{s}} B'$).*

Proof: For continuous functions $f_1, f_2 : B \to B'$, the pointwise meet is obtained by

$$f_1 \wedge^{\mathrm{p}} f_2 = (\wedge_{B'}) \circ \langle f_1\,, f_2\rangle\,.$$

Since pairing is known to preserve continuity and the meet in B' is assumed continuous, the pointwise meet is itself continuous, since it is the composition of two continuous functions. It is immediately seen to be the meet of f_1 and f_2 in the pointwise ordered continuous function space.

Similarly, if f_1 and f_2 are stable then so is their pairing, and the continuous meet in B' is stable by 2.1.5, so that the pointwise meet is stable. Again, it is immediately seen to be the meet of f_1 and f_2 in the pointwise ordered stable function space. ∎

2.1.9 Corollary *Dcpos, dcpos with continuous meet, and bc-dcpos (and their pointed counterparts) are closed under the pointwise ordered continuous function space.*

Dcpos with continuous meet (and their pointed counterparts) are closed under the pointwise ordered stable function space.

Algebraicity of Pointwise Ordered Function Spaces

Algebraicity of dcpos is not preserved in general by the pointwise ordered continuous (or stable) function space, but it may be guaranteed by imposing an additional requirement on the domains. For instance, bounded completeness is sufficient to guarantee closure of algebraic dcpos under the pointwise ordered continuous function space. We show now that Scott domains — ω-algebraic pointed bc-dcpos — are in fact closed under the pointwise-ordered continuous function space. Note that by proposition 2.1.7 the pointwise-ordered continuous function space preserves bounded completeness, and the bounded lub is taken pointwise.

For isolated elements $x \in B$ and $y \in B'$ of pointed algebraic dcpos B and B', define the ***step function*** $[x \Rightarrow y] : B \to B'$ to be the function that maps $x' \in B$ to y if $x \leq x'$, and to $\perp_{B'}$ otherwise. Step functions are monotone, continuous and stable. They are constant iff $x = \perp$, *i.e.*, $[\perp \Rightarrow y] = \mathsf{K}y$, and they are strict iff $x > \perp$ or $y = \perp$. The notation $[x \Rightarrow y]$ will always be taken to imply isolatedness of x and y, and its use may thus require some justification.

2.1.10 Proposition *A step function between algebraic dcpos is isolated in a pointwise-ordered function space where directed lubs exist and are taken pointwise.*

If moreover the function space is bounded complete then it is algebraic, and the step functions are a sub-basis for it, meaning that (1) the set of joins of step functions dominated by a function f is directed and has f as its lub, and (2) every isolated element is the join of step functions. (Recall that we take "join" to mean the lub of a non-empty finite set.)

Proof: If $[x \Rightarrow y] \leq \bigvee^{\mathrm{p}} F$ for a directed F then $y \leq \bigvee(Fx)$ with Fx directed, and by isolatedness of y there exists $f_0 \in F$ with $y \leq f_0 x$. Thus $[x \Rightarrow y] \leq^{\mathrm{p}} f_0$, and $[x \Rightarrow y]$ is isolated.

Isolated elements are always closed under bounded joins in bounded-complete posets, so that if we assume bounded completeness of the function space then bounded joins of step functions exist and are isolated, and the joins of step functions approximating a function f form a directed set F. The lub of F is taken pointwise, and using algebraicity of B' and the pointwise order we can show that it is equal to f. Finally, if f is isolated then it must belong to the directed set of joins of step functions approximating it (it is dominated by one of these approximations, hence equals it), and it is thus a join of step functions. ∎

When restricted to pointwise ordered continuous function spaces between dcpos, currying and uncurrying are well defined and preserve continuity. It follows that the (restriction to the continuous function space of the) application $\mathsf{app}_{B,B'} = \mathsf{uncurry}\,\mathsf{id}_{B \to_c B'}$ is continuous.

Thanks mainly to these properties, Scott domains and continuous functions over them may be used to provide a semantics for PCF, known as the *continuous functions model.* See section 2.4.

The situation is different for the pointwise ordered stable function space. Firstly, imposing bounded completeness still does not yield a class of algebraic dcpos closed under this function space; we return to this deficiency in section 3.7.3. Secondly, (the restriction of) application may fail to be stable (in cases where the restricted function space is well defined). As an example consider $\mathsf{app}_{\mathbf{2},\mathbf{2}} : (\mathbf{2} \to^{\mathrm{p}}_{\mathrm{s}} \mathbf{2}) \times \mathbf{2} \to \mathbf{2}$, and take the bounded pair $(\mathsf{id}, \top) \Uparrow (\mathsf{K}\top, \bot)$. We have that $\mathsf{id} \leq^{\mathrm{p}} \mathsf{K}\top$, and thus

$$\mathsf{app}((\mathsf{id}, \top) \wedge (\mathsf{K}\top, \bot)) = \mathsf{app}(\mathsf{id}, \bot) = \bot \quad < \quad \top = \top \wedge \top = \mathsf{app}(\mathsf{id}, \top) \wedge \mathsf{app}(\mathsf{K}\top, \bot).$$

But, as Berry has shown, one may yet obtain stability of application, by employing an alternative ordering on the function space: the stable order.

The Stable Order

The ***stable order*** $\leq^{\mathrm{s}}_{B,B'}$ on the function space $B \to B'$ between posets with bounded meets is defined by

$$f \leq^{\mathrm{s}}_{B,B'} g \qquad \text{iff} \qquad f \leq^{\mathrm{p}}_{B,B'} g \;\;\&\;\; fx = fy \wedge_{B'} gx \text{ for every } x \leq_B y.$$

Equivalently,

$$f \leq^{\mathrm{s}}_{B,B'} g \qquad \text{iff} \qquad f \leq^{\mathrm{p}}_{B,B'} g \;\;\&\;\; f \sim_{B,B'} g,$$

where $f \sim_{B,B'} g$ is defined by

$$f \sim_{B,B'} g \qquad \text{iff} \qquad fx \wedge_{B'} gy = gx \wedge_{B'} fy \text{ for every } x \Uparrow_B y.$$

Yet another equivalent definition is given by

$$\begin{aligned} f \leq^{\mathrm{s}}_{B,B'} g \qquad \text{iff} \qquad f \leq^{\mathrm{p}}_{B,B'} g \;\;\&\;\; & \{x' \leq_B x \mid y \leq_{B'} fx'\} = \\ & \{x' \leq_B x \mid y \leq_{B'} gx'\} \text{ for every } x \in B, y \leq_{B'} fx. \end{aligned}$$

The stable order was introduced by Berry [Ber78, Ber79]. The use of the relation $\sim$ is a convenient way to encapsulate some of the results, but otherwise the development here follows Berry's.

We write $B \to^{\mathrm{s}}_{\mathrm{s}} B'$ for the stably ordered stable function space between B and B'.

As an example, under the stable order the function space $\mathbf{2} \to \mathbf{2}$ ceases to be a chain: $\mathsf{K}\bot \leq^{\mathrm{s}} \mathsf{id}$ and $\mathsf{K}\bot \leq^{\mathrm{s}} \mathsf{K}\top$, but $\mathsf{id} \not\leq^{\mathrm{s}} \mathsf{K}\top$ (and, in particular, the counter-example to stability of application under the pointwise order is not applicable any more).

The stable order is by definition finer than the pointwise order: $f \leq^{\mathrm{s}} g$ implies that $f \leq^{\mathrm{p}} g$, and if a set of functions is directed or bounded under the stable order then it is also directed or bounded under the pointwise order.

Stability is "inherited down" in the stable order:

2.1.11 Proposition *If a continuous function f is stably dominated by a stable function g — $f \leq^{\mathrm{s}} g$ — then f is stable.*

Proof: For every $x \Uparrow y$,

$$\begin{aligned} fx \wedge fy &= (f(x \wedge y) \wedge gx) \wedge (f(x \wedge y) \wedge gy) && (x \wedge y \leq x, y \text{ and } f \leq^{\mathrm{s}} g) \\ &= f(x \wedge y) \wedge (gx \wedge gy) && \\ &= f(x \wedge y) \wedge g(x \wedge y) && (g \text{ stable}) \\ &= f(x \wedge y) && (f \leq^{\mathrm{p}} g). \end{aligned}$$

∎

If B' is pointed then $\mathsf{K}\bot_{B'}$, a stable function, remains least under the stable order.

We write $f_1 \Uparrow^{s} f_2$ to indicate that f_1 and f_2 are bounded in the stable order (on the function space under consideration). Consistency in the stable order is characterized by the following propositions:

2.1.12 Proposition *If two functions $f_1, f_2 : B \to B'$ between dcpos with bounded meets have an upper bound in the stable order then $f_1 \sim f_2$.*

Proof: If f_1 and f_2 are stably bounded with upper bound g then $f_1 \sim g$ and $f_2 \sim g$, so that for every $x \Uparrow y$ we have that $f_1x \wedge gy = f_1y \wedge gx$ and $f_2y \wedge gx = f_2x \wedge gy$. Taking the meets of equals we have the equality:

$$(f_1x \wedge gy) \wedge (f_2y \wedge gx) = (f_1y \wedge gx) \wedge (f_2x \wedge gy),$$

and recalling that $f_1x \leq gx$, *etc*, this may be reduced to

$$f_1x \wedge f_2y = f_1y \wedge f_2x,$$

so that $f_1 \sim f_2$. ∎

Note however that in situations more general than the above proposition, the relation $\sim$ may fail to be transitive.

2.1.13 Proposition *If f_1 and f_2 are stable and $f_1 \sim f_2$ then for every $x \Uparrow y$, $f_1x \wedge f_2y = (f_1 \wedge^{\mathrm{p}} f_2)(x \wedge y)$.*

Proof: If both f_1 and f_2 are stable and $f_1 \sim f_2$ then for $x \Uparrow y$,

$$\begin{array}{rcll} f_1x \wedge f_2y & = & (f_1x \wedge f_2y) \wedge (f_1y \wedge f_2x) & (f_1 \sim f_2) \\ & = & f_1(x \wedge y) \wedge f_2(x \wedge y) & (\text{stability}) \\ & = & (f_1 \wedge^{\mathrm{p}} f_2)(x \wedge y). & \end{array}$$

∎

2.1.14 Proposition *For functions $f_1, f_2, h : B \to B'$ between dcpos, if $h \sim f_1$, $h \sim f_2$, and $f_1 \wedge^{\mathrm{p}} f_2$ is well defined (e.g., the domains have meets) then $h \sim (f_1 \wedge^{\mathrm{p}} f_2)$. If, moreover, B' is distributive and $f_1 \vee^{\mathrm{p}} f_2$ is well defined (e.g., the domains are bounded complete and f_1 and f_2 are pointwise bounded: $f_1x \Uparrow f_2x$ for every $x \in B$) then $h \sim (f_1 \vee^{\mathrm{p}} f_2)$.*

If F is a pointwise directed set of functions and the domains have a continuous meet, and if $h \sim f$ for every $f \in F$, then $h \sim \bigvee^{\mathrm{p}} F$.

Proof: Assume that $h \sim f_1$ and $h \sim f_2$. For every $x \Uparrow y$ we have:

$$\begin{array}{rcll} hx \wedge (f_1 \wedge^{\mathrm{p}} f_2)y & = & hx \wedge (f_1y \wedge f_2y) & \\ & = & (hx \wedge f_1y) \wedge (hx \wedge f_2y) & \\ & = & (hy \wedge f_1x) \wedge (hy \wedge f_2x) & (h \sim f_1,\ h \sim f_2) \\ & = & hy \wedge (f_1x \wedge f_2x) & \\ & = & hy \wedge (f_1 \wedge^{\mathrm{p}} f_2)x, & \end{array}$$

so that $h \sim (f_1 \wedge^{\mathrm{p}} f_2)$.

If we further assume that $f_1 \vee^{\mathrm{p}} f_2$ is well defined and that B' is distributive then for every $x \Uparrow y$,

$$\begin{array}{rcll} hx \wedge (f_1 \vee^{\mathrm{p}} f_2)y & = & hx \wedge (f_1 y \vee f_2 y) & \\ & = & (hx \wedge f_1 y) \vee (hx \wedge f_2 y) & \text{(distributivity)} \\ & = & (hy \wedge f_1 x) \vee (hy \wedge f_2 x) & (h \sim f_1,\, h \sim f_2) \\ & = & hy \wedge (f_1 x \vee f_2 x) & \text{(distributivity)} \\ & = & hy \wedge (f_1 \vee^{\mathrm{p}} f_2)x, & \end{array}$$

so that $h \sim (f_1 \vee^{\mathrm{p}} f_2)$.

If F is pointwise directed, so that $\bigvee^{\mathrm{p}} F$ is well defined, and $h \sim f$ for every $f \in F$, and we further assume that the domains have continuous meet then

$$\begin{array}{rcll} hx \wedge (\bigvee^{\mathrm{p}} F)y & = & hx \wedge \bigvee(Fy) & \\ & = & \bigvee \{hx \wedge fy \mid f \in F\} & \text{(continuity of meet)} \\ & = & \bigvee \{hy \wedge fx \mid f \in F\} & (h \sim f \text{ for } f \in F) \\ & = & hy \wedge \bigvee(Fx) = hy \wedge (\bigvee^{\mathrm{p}} F)x, & \end{array}$$

and $h \sim \bigvee^{\mathrm{p}} F$. ∎

2.1.15 Proposition *If $f_1, f_2 : B \to B'$ are stable functions with a well defined pointwise meet and $f_1 \sim f_2$ then their pointwise meet is their meet in $B \to^{\mathrm{s}}_{\mathrm{s}} B'$.*

Proof: By proposition 2.1.8, $g = f_1 \wedge^{\mathrm{p}} f_2$ is stable, and is the meet of f_1 and f_2 in the pointwise order. If we now assume that $f_1 \sim f_2$ then by proposition 2.1.14 (and $f_1 \sim f_1$) $f_1 \sim g$, and similarly for f_2 so that g is a lower bound in the stable order. Finally, if h is another lower bound then $h \leq^{\mathrm{p}} g$, and proposition 2.1.14 again gives $h \sim g$,so that $h \leq^{\mathrm{s}} g$, and g is the meet of f_1 and f_2 in $B \to^{\mathrm{s}}_{\mathrm{s}} B'$. ∎

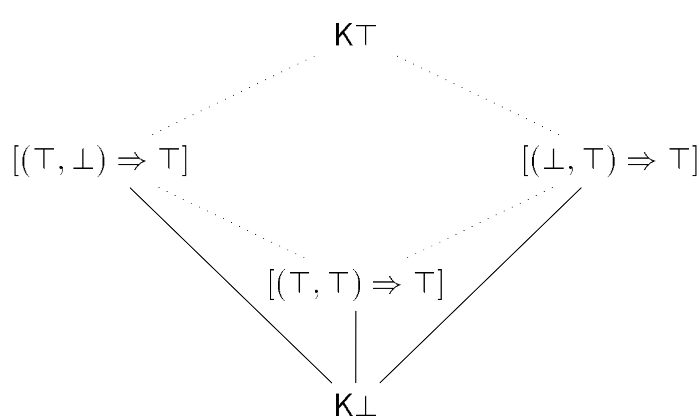

Figure 2.3: Pointwise meet vs. meet in the stable order

2.1.16 Example To see that the pointwise meet need not be the meet in the stable order for an arbitrary pair of stable functions consider $[(\top, \bot) \Rightarrow \top]$ and $[(\bot, \top) \Rightarrow \top]$ drawn from the stable function space $\mathbf{2} \times \mathbf{2} \to_{\mathrm{s}} \mathbf{2}$. These functions are pointwise consistent, but are not consistent in the stable order (hence they are not related by $\sim$, see next proposition). Their pointwise join $[(\top, \bot) \Rightarrow \top] \vee^{\mathrm{p}} [(\bot, \top) \Rightarrow \top]$ is not stable. Their pointwise meet $[(\top, \top) \Rightarrow \top]$ is stable, and is their meet in the pointwise order, but is not a lower bound in the stable order; their meet in the stable order is $\mathsf{K}\bot$.

Figure 2.3 shows the situation. Dotted lines correspond to the pointwise order, and solid lines to the stable order. •

2.1.17 Proposition *For stable functions $f_1, f_2 : B \to B'$ between distributive dcpos the following are equivalent:*

(1) f_1 and f_2 are bounded in $B \to_{\mathrm{s}}^{\mathrm{s}} B'$.

(2) f_1 and f_2 are pointwise bounded (i.e., $f_1 x \Uparrow_B f_2 x$ for all $x \in B$), and $f_1 \sim f_2$.

(3) The pointwise join $f_1 \vee^{\mathrm{p}} f_2$ is well-defined and stable, and is the join of f_1 and f_2 in $B \to_{\mathrm{s}}^{\mathrm{s}} B'$.

Proof: (1) implies (2) by definition of the order and proposition 2.1.12. (3) trivially implies (1). To see that (2) implies (3) let $g = f_1 \vee^{\mathrm{p}} f_2$, a well defined function since B' is bounded complete and f_1 and f_2 are pointwise consistent. g is already known to be continuous, and we show it is stable. If $x \Uparrow_B y$ then

$$\begin{array}{rcll}
gx \wedge gy & = & (f_1 x \vee f_2 x) \wedge (f_1 y \vee f_2 y) & \\
& = & (f_1 x \wedge f_1 y) \vee (f_2 x \wedge f_2 y) \vee & \text{(distributivity)} \\
& & (f_1 x \wedge f_2 y) \vee (f_2 x \wedge f_1 y) & \\
& = & f_1(x \wedge y) \vee f_2(x \wedge y) \vee (f_1 \wedge^{\mathrm{p}} f_2)(x \wedge y) & (f_1 \sim f_2 \text{ and } 2.1.13) \\
& = & f_1(x \wedge y) \vee f_2(x \wedge y) & \\
& = & g(x \wedge y). &
\end{array}$$

Clearly, $f_1 \leq^{\mathrm{p}} g$. By proposition 2.1.14 $f_1 \sim g$, since $f_1 \sim f_1$ and $f_1 \sim f_2$, so that $f_1 \sim (f_1 \vee^{\mathrm{p}} f_2)$. It follows that $f_1 \leq^{\mathrm{s}} g$, and similarly $f_2 \leq^{\mathrm{s}} g$, so that g is an upper bound of f_1 and f_2 in the stable order. To show it least, assume another upper bound h; clearly $g \leq^{\mathrm{p}} h$, and from $h \sim f_1$ and $h \sim f_2$ we obtain, again by proposition 2.1.14, that $h \sim g$, and thus $g \leq^{\mathrm{s}} h$. g is therefore the join of f_1 and f_2 in $B \to_{\mathrm{s}}^{\mathrm{s}} B'$, proving (3). ∎

2.1.18 Corollary *For stable functions between distributive dcpos, if $f \leq^{\mathrm{p}} g$ and $f \Uparrow^{\mathrm{s}} g$ then $f \leq^{\mathrm{s}} g$.*

2.1.19 Corollary *For a stably directed set F of stable functions between dcpos and every $f, f' \in F$, it is the case that $f \sim f'$.*

2.1.20 Proposition *The stably ordered stable function space $B \to_{\mathrm{s}}^{\mathrm{s}} B'$ between dcpos with continuous meet B and B' is directed complete, and the lub of a directed set F is given by its pointwise lub $\bigvee^{\mathrm{p}} F$.*

Proof: If F is a stably directed set of stable functions then it is pointwise directed and its pointwise lub $f_0 = \bigvee^{\mathrm{p}} F$ is known by proposition 2.1.7 to be stable and a lub of F under the pointwise order. To show that it is also the lub under the stable order, observe that for every $f' \in F$, $f' \sim f$ for every other $f \in F$, so that by proposition 2.1.14, $f' \sim \bigvee^{\mathrm{p}} F$, and $f' \leq^{\mathrm{s}} \bigvee^{\mathrm{p}} F$, so that f_0 is a bound of F in the stable order. To show that it is least, assume that g is another bound of F in the stable order; clearly, $f_0 \leq^{\mathrm{p}} g$, and since $g \sim f$ for every $f \in F$, $g \sim \bigvee^{\mathrm{p}} F$, again by 2.1.14, so that $f_0 \leq^{\mathrm{s}} g$, and f_0 is in fact the least upper bound of F in the stable order. ∎

2.1.21 Corollary *The stably ordered stable function space $B \to^{\mathrm{s}}_{\mathrm{s}} B'$ between distributive dcpos is a distributive dcpo, with bounded joins, bounded meets and directed lubs taken pointwise.*

Proof: Directed completeness has been shown in 2.1.20. Bounded completeness follows from 2.1.17 and 2.1.20. Distributivity of $B \to^{\mathrm{s}}_{\mathrm{s}} B'$ is inherited from B', since bounded meets and bounded joins are taken pointwise in the stably ordered function space (propositions 2.1.17 and 2.1.15). ∎

At this point one would like to add algebraicity to the list of properties preserved by the stably-ordered stable function space, and prove that dI-domains are closed under this function space. To avoid the need for a direct characterization of isolated elements in the function space, we delay the proof until the introduction of stable strong algebraicity, in the next section.

2.1.5 Strong Algebraicity

The class of strongly algebraic domains, or SFP domains, was introduced by Plotkin [Plo76]. It is a larger class than Scott domains, and unlike Scott domains it is closed under the Plotkin powerdomain construction. It arises by replacing the requirement of bounded-completeness of Scott domains by a weaker property of strong algebraicity. Strong algebraicity, in its stable form, will be necessary to present Berry's definition of bidomains [Ber79] in chapter 6. In the meantime, it provides some technical convenience for stating some results. The presentation here is based on [Plo76, Mil77, Ber79], and follows closely the last reference, especially for the stable version.

A ***continuous projection*** on a dcpo B is a continuous function $\psi : B \to B$ that is ***idempotent***, $\psi \circ \psi = \psi$, and ***pointwise deflating***, *i.e.*, it is pointwise dominated by the identity on B, $\psi \leq^{\mathrm{p}} \mathsf{id}$. We may omit the qualification of continuity when it is not ambiguous to do so. A projection ψ is ***finite*** iff ψB is a finite subset of B. An inverse ***sequence of finite continuous projections***, or ***SFP***, Ψ on B is a countable sequence $\psi_1, \psi_2, \ldots$ of finite projections that is increasing and has the identity as its lub in the pointwise order, *i.e.*, $\psi_i \leq^{\mathrm{p}} \psi_{i+1}$ and $\mathsf{id}_B = \bigvee^{\mathrm{p}} \{\psi_i \mid i \in \mathbb{N}\}$. B is ***strongly algebraic*** iff it has an SFP.

Note that the term *projection* is used in several senses. We have already defined the *product projections* and *continuous projections*, and we will subsequently define *stable projections* and *structural projections*. These meanings are distinct, but we will sometimes allow ourselves a less than complete qualification of the term used, when it does not seem to pose a problem.

Strong algebraicity implies ω-algebraicity.

2.1.22 Proposition *A strongly algebraic dcpo B with SFP Ψ is ω-algebraic and has*

$$\mathbf{K}\,B = \bigcup \{\psi_i B \mid i \in \mathbb{N}\}\,.$$

Proof: Start by showing $\psi_i x$ isolated for every i and $x \in B$. If $\psi_i x \leq \bigvee Z$ for some directed set Z then

$$\psi_i x = \psi_i(\psi_i x) \leq \psi_i(\bigvee Z) = \bigvee(\psi_i Z).$$

$\psi_i Z$, being finite and directed, has a maximum element $\psi_i z = \bigvee(\psi_i Z)$, for some $z \in Z$. But then $\psi_i x \leq \psi_i z \leq z$, and $\psi_i x$ must be isolated.

Conversely, if x is isolated then $x = \bigvee \{\psi_i x \mid i \in \mathbb{N}\}$, a directed lub, and being isolated, $x = \psi_i x$ for some i. This completes the proof of the equality $\mathbf{K}\,B = \bigcup \{\psi_i B \mid i \in \mathbb{N}\}$, and $\mathbf{K}\,B$ is countable, since it is the countable union of finite sets.

Finally, for every $x \in B$, $x = \bigvee \{\psi_i x \mid i \in \mathbb{N}\}$, so that $x = \bigvee(\mathbf{K} \downarrow x)$. To show that $\mathbf{K} \downarrow x$ is directed, observe that since $x = \bigvee \{\psi_i x \mid i \in \mathbb{N}\}$ is a directed lub, any two isolated approximations to x will have as an isolated upper bound $\psi_i x$ for some i. This proves algebraicity of B. ∎

Strongly algebraic domains are not necessarily bounded complete. The next proposition relates them to Scott domains.

2.1.23 Proposition *If B is a Scott domain then it is strongly algebraic, and it has an SFP defined by $\psi_i x = \bigvee \{x_j \leq x \mid j \leq i\}$ where $x_1, x_2, \ldots$ is some enumeration of $\mathbf{K}\,B$.*

Conversely, every pointed strongly algebraic dcpo with meets is a Scott domain.

Proof: Let B be a Scott domain and define ψ_i as above for some fixed enumeration of $\mathbf{K}\,B$. Every ψ_i is well-defined by bounded-completeness, and by definition it is monotone, idempotent and pointwise deflating. Moreover, $\psi_i B$ is finite since it is contained in the finite set $\{\vee X \mid X \subseteq \{x_1, \ldots, x_i\} \;\; \& \;\; X \neq \emptyset \;\; \& \Uparrow X\}$. It follows that ψ_i is continuous, and it is a continuous projection. It is now easy to check that $\psi_i \leq^{\mathrm{p}} \psi_{i+1}$, and $\mathsf{id}_B = \bigvee^{\mathrm{p}} \{\psi_i \mid i \in \mathbb{N}\}$.

For the converse direction, assume that B is a pointed algebraic dcpo with meets, with a SFP Ψ. B is ω-algebraic by 2.1.22, so that to establish that it is a Scott domain it suffices to prove the existence of glbs of all non-empty sets (by 2.1.1). Let $X \subseteq B$ be a non-empty set, so that for every i, $\psi_i X$ is a non-empty finite set, which has a meet $z_i = \wedge(\psi_i X)$ by assumption. Let z be the lub of $\{z_i \mid i \in \mathbb{N}\}$, a directed set. Then z is a lower bound of X, and if y is another lower bound of X then

$$z = \bigvee \{z_i \mid i \in \mathbb{N}\} \geq \bigvee \{\psi_i y \mid i \in \mathbb{N}\} = y,$$

so that z is the glb of X, as desired. ∎

The terminal poset $\{\bot\}$ is trivially strongly algebraic, and strong algebraicity is preserved by poset product and the continuous function space:

For strongly algebraic dcpos B_1 and B_2 with SFPs Ψ^1 and Ψ^2 respectively, define the ***product SFP*** $\Psi^1 \times \Psi^2$ by

$$(\Psi^1 \times \Psi^2)_i = (\psi_i^1 \times \psi_i^2).$$

2.1.24 Proposition *If B_1 and B_2 are strongly algebraic dcpos with SFPs Ψ^1 and Ψ^2, then $B_1 \times B_2$ is strongly algebraic with the product SFP.*

Proof: The proof relies on functoriality and continuity of the product, as well as

$$(\psi_i^1 \times \psi_i^2)(B_1 \times B_2) \subseteq (\psi_i^1 B_1) \times (\psi_i^2 B_2)$$

for establishing finiteness. ∎

For strongly algebraic dcpos B and B' with SFPs Ψ and Ψ', respectively, define the ***exponent SFP*** $\Psi \to \Psi'$ over $B \to_{\mathrm{c}} B'$ by

$$\begin{aligned}(\Psi \to \Psi')_i &= \psi_i \to \psi_i' \\ &= \lambda h \in B \to_{\mathrm{c}} B' \,.\, \psi_i' \circ h \circ \psi_i.\end{aligned}$$

2.1.25 Proposition *If B and B' are strongly algebraic dcpos with SFPs Ψ and Ψ', then $B \to_{\mathrm{c}} B'$ is strongly algebraic with the exponentiation SFP $\Psi \to \Psi'$.*

Proof: Continuity of each $\psi_i \to \psi_i'$ follows from continuity of composition. For pointwise deflation, use monotonicity of $\to$:

$$\psi_i \to \psi_i' \leq^{\mathrm{p}} \mathsf{id}_B \to \mathsf{id}_{B'} = \mathsf{id}_{B \to_{\mathrm{c}} B'},$$

hence $(\psi_i \to \psi_i')h \leq^{\mathrm{p}} h$, and $\psi_i \to \psi_i' \leq^{\mathrm{p}} \mathsf{id}_{B \to_{\mathrm{c}} B'}$. Idempotency follows from functoriality of $\to$:

$$(\psi_i \to \psi_i') \circ (\psi_i \to \psi_i') = (\psi_i \circ \psi_i) \to (\psi_i' \circ \psi_i') = \psi_i \to \psi_i'.$$

For every $h \in B \to_{\mathrm{c}} B'$, $(\psi_i \to \psi_i')h$ is (completely determined by) a mapping of the finite set $\psi_i B$ into the finite set $\psi_i' B'$, so that $(\psi_i \to \psi_i')(B \to_{\mathrm{c}} B')$ is finite. This establishes that $\psi_i \to \psi_i'$ is a finite continuous projection for every i.

Continuity of the composition implies that $(\psi_i \to \psi_i') \leq^{\mathrm{p}} (\psi_{i+1} \to \psi_{i+1}')$, and $\mathsf{id}_{B \to_{\mathrm{c}} B'} = \bigvee^{\mathrm{p}} \{\psi_i \to \psi_i' \mid i \in \mathbb{N}\}$, and hence $\Psi \to \Psi'$ is in fact an SFP. ∎

Stable Strong Algebraicity

We now present a stable version of strong algebraicity. The definitions and development in this section are mostly due to Berry [Ber79].

A ***stable projection*** on a distributive dcpo B is a stable function $\psi : B \to B$ that is idempotent, $\psi \circ \psi = \psi$, and stably deflating, *i.e.*, ψ is stably dominated by the identity on B, $\psi \leq^{s} \mathsf{id}$. A stable projection is easily seen to be a continuous projection. A stable projection is ***finite*** when it is finite as a continuous projection, *i.e.*, ψB is a finite subset of B. A ***stable SFP*** Ψ on B is a countable sequence $\psi_1, \psi_2, \ldots$ of finite stable projections that is increasing and has the identity as its lub in the stable order, *i.e.*, $\psi_i \leq^{s} \psi_{i+1}$ and $\mathsf{id} = \bigvee^{s} \{\psi_i \mid i \in \mathbb{N}\}$. B is ***stably strongly algebraic*** iff it has a stable SFP. A stable SFP for B is easily seen to be a (continuous) SFP for B, and a stably strongly algebraic distributive dcpo B is (continuously) strongly algebraic.

dI-domains may be characterized by means of stable strong algebraicity.

2.1.26 Proposition *A dcpo with a stable SFP verifies property (I).*

Proof: Let B be a dcpo with stable SFP Ψ. In particular, Ψ is a continuous SFP, so by 2.1.22 B is ω-algebraic and

$$\mathbf{K}\,B = \bigcup \{\psi_i B \mid i \in \mathbb{N}\}\,.$$

If $x \in \mathbf{K}\,B$ then there exists i such that $x = \psi_i x$. For every $y \leq x$,

$$\psi_i y = \mathsf{id}\, y \wedge \psi_i x = y \wedge x = y$$

since $\psi_i \leq^{s} \mathsf{id}$. Therefore $y \in \psi_i B$, a finite set, so that x dominates finitely many elements of B. ∎

2.1.27 Proposition *A pointed distributive dcpo is a dI-domain iff it has a stable SFP.*

Proof: If B has a stable SFP then it is ω-algebraic, by 2.1.22, and it has property (I) by 2.1.26, so that it is a dI-domain.

If B is a dI-domain then one can construct explicitly a stable SFP, based on an enumeration $x_1, x_2 \ldots$ of $\mathbf{K}\,B$: let $\psi_i x = \bigvee \{x \wedge x_j \mid j \leq i\}$. The remaining details are omitted (see [Ber79]). ∎

2.1.28 Proposition *The product and exponent of stable SFPs are stable SFPs.*

Proof: The product and exponent of SFPs are SFPs, and one needs to further check that the constructions preserve stability, and the desired relations for the stable order. ∎

As a corollary of the above two propositions and 2.1.21 one obtains an easy proof that dI-domains are closed under the stably-ordered stable function space:

2.1.29 Corollary *The stably-ordered stable function space between dI-domains is a dI-domain.*

Moreover, we obtain a convenient characterization of the isolated elements of this function space.

2.1.30 Proposition *If B and B' are dI-domains then the isolated elements of the stably-ordered stable function space $B \to_{\mathrm{s}}^{\mathrm{s}} B'$ are the stable functions that are pointwise joins of step functions. (Recall that joins are taken over* finite *non-empty sets.)*

In other words:

$$\mathbf{K}(B \to_{\mathrm{s}}^{\mathrm{s}} B') = (B \to_{\mathrm{s}} B') \cap \mathbf{K}(B \to_{\mathrm{c}}^{\mathrm{p}} B').$$

Proof: Let Ψ and Ψ' be stable SFPs for B and B'. Therefore, $B \to_{\mathrm{s}}^{\mathrm{s}} B'$ has $\Psi \to \Psi'$ as a stable SFP, with $\mathbf{K}(B \to_{\mathrm{s}}^{\mathrm{s}} B') = \bigcup \{(\psi_i \to \psi'_i)f \mid f \in B \to_{\mathrm{s}} B'\}$.

But by 2.1.25 it is also the case that $B \to_{\mathrm{c}}^{\mathrm{p}} B'$ has $\Psi \to \Psi'$ as an SFP, with

$$\mathbf{K}(B \to_{\mathrm{c}}^{\mathrm{p}} B') = \bigcup \{(\psi_i \to \psi'_i)f \mid f \in B \to_{\mathrm{c}} B'\}.$$

(Actually, this second construction of $\Psi \to \Psi'$ has a larger domain of definition, but, by its definition, when restricted to stable functions it agrees with the first construction.)

It is immediate then that

$$\begin{array}{rcl} \mathbf{K}(B \to_{\mathrm{s}}^{\mathrm{s}} B') & = & \bigcup \{(\psi_i \to \psi'_i)f \mid f \in B \to_{\mathrm{s}} B'\} \\ & \subseteq & \bigcup \{(\psi_i \to \psi'_i)f \mid f \in B \to_{\mathrm{c}} B'\} = \mathbf{K}(B \to_{\mathrm{c}}^{\mathrm{p}} B'), \end{array}$$

and

$$\mathbf{K}(B \to_{\mathrm{s}}^{\mathrm{s}} B') \subseteq (B \to_{\mathrm{s}} B') \cap \mathbf{K}(B \to_{\mathrm{c}}^{\mathrm{p}} B').$$

On the other hand, if $(\psi_i \to \psi'_i)f \in (B \to_{\mathrm{s}} B') \cap \mathbf{K}(B \to_{\mathrm{c}}^{\mathrm{p}} B')$ for some $f \in B \to_{\mathrm{c}} B'$ then $(\psi_i \to \psi'_i)((\psi_i \to \psi'_i)f) \in \mathbf{K}(B \to_{\mathrm{s}}^{\mathrm{s}} B')$, and by idempotency, $(\psi_i \to \psi'_i)f \in \mathbf{K}(B \to_{\mathrm{s}}^{\mathrm{s}} B')$, establishing the desired equality. ∎

2.1.6 Covering and Primes

This section defines and presents a few results concerning covering and primes. Most of the material here is standard. A fuller exposition may be found in [Zha91, Win87].

We have already defined the covering relation induced by a partial order: $x \prec y$ iff $x < y$ and there is no element strictly between x and y.

A related notion is that of primes. An element $x \in B$ is ***prime*** (sometimes called ***complete prime***) iff for every $X \subseteq B$, if $\bigvee X$ exists and $x \le \bigvee X$ then there exists $y \in X$ with $x \le y$. A poset B is ***prime algebraic*** iff every $x \in B$ is the lub of the prime elements below it. Write $\mathbf{Pr}\, B$ for the collection of prime elements of B.

It is a standard result that x is prime iff it covers a unique element. Define the ***predecessor*** of x to be $\mathsf{pred}\, x = \bigvee \{y \in B \mid y < x\}$. Then x is prime iff $\mathsf{pred}\, x$ is well defined (the lub exists) and, moreover, $\mathsf{pred}\, x \prec x$.

Covering and Primes over Product

It is straightforward to characterize covering and primes over a product:

2.1.31 Proposition *$(x,y) \prec (x',y')$ iff either $x \prec x'$ and $y = y'$, or $x = x'$ and $y \prec y'$.*

Proof: ($\Longleftarrow$) is straightforward. For ($\Longrightarrow$), we observe first that $x \leq x'$ and $y \leq y'$. If both of the inequalities are strict, then we have a contradiction, since $(x,y) < (x,y') < (x',y')$. Hence exactly one of the inequalities is strict, and the other is an equality. Assume without loss of generality that $x < x'$ and $y = y'$, and $x \prec x'$ follows. ∎

2.1.32 Proposition *(x',y') is prime in $B_1 \times B_2$ iff either x' is prime in B_1 and $y' = \bot$, or $x' = \bot$ and y' is prime in B_2.*

Proof: If x' is prime and $y' = \bot$ then

$$\mathsf{pred}(x',y') = \bigvee \{(x,y) < (x',y')\} = \bigvee \{(x,\bot) \mid x < x'\} = (\mathsf{pred}\, x', \bot),$$

so that the lub exists and $\mathsf{pred}(x',y') = (\mathsf{pred}\, x', \bot) \prec (x',y')$.

If (x',y') is prime and there exist $x < x'$ and $y < y'$ then we derive a contradiction to primality of (x',y'), since $(x',y') = (x,y') \vee (x',y)$. Therefore either $x' = \bot$ or $y' = \bot$, and it is easy to verify that the other component must be a prime itself. ∎

Covering and Primes in dI-domains

dI-domains are prime algebraic. Thanks to bounded completeness, $\mathsf{pred}\, x$ is well defined for every x. Importantly, every covering step $x \prec y$ corresponds to a unique prime, which may be seen as representing the increment in information from x to y. A basic fact about distributive posets is presented first.

2.1.33 Proposition *If B is a distributive bc-poset and x, y, z are three bounded elements such that $x \vee y = x \vee z$ and $x \wedge y = x \wedge z$ then $y = z$.*

Proof: A proof may be found in [DP90]. ∎

2.1.34 Proposition *For any two elements x, y of a dI-domain B, $x \prec y$ iff there exists $a \in \mathbf{Pr}\, B$ such that $y = x \vee a$, $a \not\leq x$, and $\mathsf{pred}\, a \leq x$. Moreover, such an a is unique, when it exists.*

Proof:

$\Longrightarrow$ Assume that $x \prec y$. By property (I), any descending chain in $\mathbf{Pr}\!\downarrow\! y$ must be finite. Therefore there must exist $a \in \mathbf{Pr}\!\downarrow\! y$ such that $a \not\leq x$ and $\mathsf{pred}\, a \leq x$. Moreover, $x < x \vee a \leq y$, and since y covers x, $y = x \vee a$.

To see that a is unique, assume that there exists a' with the same properties. Since a is prime and $a \not\leq x$, $a \leq y = x \vee a'$ implies that $a \leq a'$. The reverse inequality is obtained similarly, and therefore $a = a'$.

$\Leftarrow$ Assume that $y = x \vee a$ for some prime a with $a \not\leq x$ and $\mathsf{pred}\, a \leq x$. If $x \leq z \leq y$ then

$$\mathsf{pred}\, a = x \wedge a \leq z \wedge a \leq y \wedge a,$$

and since $\mathsf{pred}\, a \prec a$, either $z \wedge a = a$ or else $\mathsf{pred}\, a = z \wedge a$. In the first case $z \geq a$ and $z = y$, and in the second case $x = z$, by proposition 2.1.33 since the domain is distributive and $x \wedge a = z \wedge a (= \mathsf{pred}\, a)$ and $x \vee a = z \vee a (= y)$. Therefore $x \prec y$. ∎

This means that whenever $x \prec y$, we can talk about the "difference" between the two points, and give it a name — the unique prime a such that $y = x \vee a$ and $\mathsf{pred}\, a \leq x$. It is in this sense that the primes fill the role of "atomic" increments in information. A finite difference in information may also be described by a sequence of increments.

2.1.35 Proposition *For $x \leq y$ isolated in a dI-domain, there exists a finite covering chain from x to y, i.e., a finite chain $x = z_1 \prec \cdots \prec z_n = y$.*

Proof: Let A be the set of primes dominated by y, but not by x. The proof proceeds by induction on the cardinality of A, which is certainly finite. If A is empty then $x = y$, by prime algebraicity, and we take a trivial chain $x = z_1 = y$. If A is non-empty, it must contain a minimal element a, and it must be the case that $\mathsf{pred}\, a \leq x$. Therefore $x \prec x \vee a$, and by induction hypothesis there is a covering chain from $x \vee a$ to y. ∎

2.1.36 Proposition *In a dI-domain, if $x \prec y$, $x \leq z$, $y \Uparrow z$ and $y \not\leq z$ then $z \prec z \vee y$.*

Proof: Let a be the covering step $x \prec y$, as defined above. Then $\mathsf{pred}\, a \leq x \leq z$, but $a \not\leq z$ since $x \vee a = y \not\leq z$. Therefore $z \prec z \vee a = z \vee y$. ∎

By a double application of this proposition it follows that dI-domains have property (C), defined by:

(C) If $x \prec y$, $x \prec z$, $y \Uparrow z$ and $y \neq z$ then $y \prec y \vee z$ and $z \prec y \vee z$.

2.2 Topological Characterizations

Monotonicity, continuity and stability of functions, as well as the pointwise and the stable orders, have equivalent characterizations using a generalized topological approach. This is well known for the continuous functions following Scott's work [Sco70, Sco72]. A generalized topological characterization of stability has been proposed by Lamarche [Lam93] and Zhang [Zha91]. Our presentation here builds on Zhang's lead and generalizes it.

A ***generalized topology*** Ω over some class $\mathcal{A}$ of sets assigns to each set $A \in \mathcal{A}$ a family ΩA of subsets of A, called Ω-open sets of A, ordered by an order $\leq^{\Omega}$. Ω is a proper ***topology*** if for every A: (1) $\emptyset$ and A are Ω-open (2) Ω-open sets are closed under arbitrary unions (3) Ω-open sets are closed under finite intersections, and (4) the order used is inclusion. This may be expressed by saying that ΩA is a sub-frame of the powerset lattice of A, ordered by inclusion (see for instance [Vic89]). The generalizations that we employ here arise by relaxing the sub-frame requirement, and in particular relaxing requirement (2), so that Ω-open are not necessarily closed under arbitrary union, and/or choosing an ordering on Ω-open sets different from set inclusion, and relaxing the derived requirement that the poset of Ω-opens form a complete lattice.

The ***Ω-continuous*** functions from A to A' are the functions $f : A \to A'$ such that the inverse image $f^{-1}q$ of every $q \in \Omega A'$ is in ΩA. The ordering of the Ω-open sets induces an ordering on the Ω-continuous functions by $f \leq^{\Omega} g$ iff for every $q \in \Omega A'$, $f^{-1}q \leq^{\Omega} g^{-1}q$. The usual notion of continuous functions is the one induced by the Scott topology (defined below), and we reserve the unqualified term of continuous functions for these Scott continuous functions.

For every generalized topology Ω, identity functions are Ω-continuous, and Ω-continuous functions are closed under composition.

If Ω assigns to $\mathbf{2} = \{\bot, \top\}$ the Sierpinski space structure, *i.e.*, $\Omega\mathbf{2} = \{\emptyset, \{\top\}, \{\bot, \top\}\}$, then it is easy to see that for any A, $(\Omega A, \leq^{\Omega})$ is isomorphic to the Ω-continuous function space from A to $\mathbf{2}$, ordered by $\leq^{\Omega}$. This is because $f^{-1}\emptyset = \emptyset$ and $f^{-1}\{\bot, \top\} = A$ for any function f from A to $\mathbf{2}$, so that $f^{-1}\{\top\}$ should completely characterize f. Thus, $f \mapsto f^{-1}\{\top\}$ and $p \mapsto \chi_p$ (*i.e.*, map p to its characteristic function that returns $\top$ only on $x \in p$ and $\bot$ elsewhere) are the isomorphisms between the two posets; the isomorphisms are order-isomorphisms by virtue of the definition of $\leq^{\Omega}$ on the function space $A \to \mathbf{2}$.

We do not give a definition of generalized topologies, since all we need are a few examples. We only point out now some common traits of the generalized topologies defined here: for every A, the empty set $\emptyset$ and A itself are open sets, ΩA has T_0-separation and monotonicity, but not necessarily conformality (these separation properties are defined below), and $\mathbf{2}$ is in fact given the Sierpinski space structure. We only use two variations on the order: either set inclusion, or lobe inclusion, as defined below. While Ω-opens are not necessarily closed under arbitrary unions, they are closed under directed unions; some additional lubs exist, but they are not always given by union.

2.2.1 Separation Properties

There are several separation properties that may be defined for families $O \subseteq \mathcal{P}\,B$ of subsets of a poset. Given such a family O and $x \in B$, let $O_x = \{p \in O \mid x \in p\}$.

A family $O \subseteq \mathcal{P}\,A$ has ***T_0-separation*** iff for every $x, y \in A$,

$$x = y \quad \text{iff} \quad O_x = O_y,$$

or equivalently (since the "only if" direction here is always true),

$$x \neq y \quad \Longrightarrow \quad \exists p \in O \,.\, (x \in p \;\;\&\;\; y \notin p) \;\text{ or }\; (x \notin p \;\;\&\;\; y \in p).$$

(T_0-separation is also applicable to families of subsets of a set, since it makes no appeal to the partial order.)

A family $O \subseteq \mathcal{P}\,B$ of subsets of a poset B is ***monotone*** iff every $p \in O$ is up-closed, or, equivalently, when

$$x \leq y \quad \Longrightarrow \quad O_x \subseteq O_y.$$

$O \subseteq \mathcal{P}\,B$ is ***conformal*** when for every $x, y \in B$,

$$x \leq y \iff O_x \subseteq O_y.$$

Conformality should convey that O both *conforms* to the partial order (this is the monotonicity direction), and, moreover, that O completely *defines* the partial order. Conformality implies T_0-separation and monotonicity.

T_0-separation and monotonicity of $O \subseteq \mathcal{P}\,B$ imply an ***order separation*** property: for every $x, y \in B$,

$$x < y \implies O_x \subset O_y.$$

This may also be given a weaker formulation (equivalent under assumption of monotonicity, which is always the case for us) by requiring that for every $x, y \in B$,

$$x < y \implies \exists p \in O \,.\, x \notin p \;\;\&\;\; y \in p.$$

2.2.2 Inclusion and The Pointwise Order

The pointwise order on functions corresponds to inclusion on the powerset, in the following sense:

2.2.1 Proposition *For functions $f, g : B \to B'$ between posets, the following are equivalent:*

(1) $f \leq^{\mathrm{p}} g$,

(2) $f^{-1}q \subseteq g^{-1}q$ *for every up-closed* $q \subseteq B'$,

(3) $f^{-1}(\uparrow y) \subseteq g^{-1}(\uparrow y)$ *for every* $y \in B'$.

Proof: The implication (2) $\implies$ (3) is immediate.

(1) $\implies$ (2) Assume (1). If $x \in f^{-1}q$ for some up-closed q then $fx \in q$, and since $gx \geq fx$ and q is up-closed, $gx \in q$ and $x \in g^{-1}q$.

(3) $\implies$ (1) Assume (3). For every $x \in B$, $x \in f^{-1}(\uparrow fx) \subseteq g^{-1}(\uparrow fx)$, so that $gx \in \uparrow fx$ and $fx \leq gx$. ∎

2.2.3 The Scott Topology

A subset $p \subseteq B$ of a dcpo is ***Scott open*** iff it is up-closed and has the ***Scott property***: for every directed set X, if $\bigvee X \in p$ then $x \in p$ for some $x \in X$. The ***Scott topology*** Sc assigns to a dcpo B the set $\mathsf{Sc}B$ of Scott opens of B. This is in fact a proper topology, and the functions continuous with respect to the Scott topology coincide with the continuous functions we have previously defined (*i.e.*, functions that are monotone and preserve directed lubs). For full details see for instance [Vic89].

Algebraicity provides some convenient characterizations, summarized by the next proposition. For instance, the Scott opens of an algebraic dcpo are up-closed sets determined by their isolated elements, and continuity and the pointwise order are determined by the inverse images of principal filters $\uparrow y$ for *isolated* y.

2.2.2 Proposition *For algebraic dcpos B and B':*

(1) $p \subseteq B$ is Scott open iff $p = \uparrow \mathbf{K}\, p$.

(2) For every $x \in \mathbf{K}\, B$, $\uparrow x$ is Scott open; the Scott topology has T_0-separation.

(3) For every function $f : B \to B'$ and Scott open q of B',

$$f^{-1}q = \bigcup \{f^{-1}(\uparrow y) \mid y \in \mathbf{K}\, q\}\,.$$

(4) A function $f : B \to B'$ is continuous iff $f^{-1}(\uparrow y)$ is Scott open for every $y \in \mathbf{K}\, B'$.

(5) For $f, g : B \to B'$, $f \leq^{\mathrm{p}} g$ iff $f^{-1}(\uparrow y) \subseteq g^{-1}(\uparrow y)$ for every $y \in \mathbf{K}\, B'$.

Proof: (1) and (3) are immediate, as is the "only if" direction of (4) and (5). (2) follows from (1); note that $x < y$ are separated by the Scott open $\uparrow z$ iff $z \in \mathbf{K}\downarrow y \setminus \mathbf{K}\downarrow x$. If $f^{-1}(\uparrow y)$ is Scott open for every $y \in \mathbf{K}\, B'$ then for every $q \in \mathsf{Sc}B'$, $f^{-1}q$ is a union of Scott opens, by (3), hence it is Scott open and f continuous. If $f^{-1}(\uparrow y) \subseteq g^{-1}(\uparrow y)$ for every $y \in \mathbf{K}\, B'$ then every isolated approximation y to fx is also an approximation to gx, so that $fx \leq gx$ for every x. ∎

Here's a convenient characterization of the interaction of directed lub of Scott opens with the inverse image by a pointwise lub:

2.2.3 Proposition *If F is a pointwise-directed set of continuous functions from B to B' then for every Scott open $q \in \mathsf{Sc}B'$,*

$$(\bigvee\nolimits^{\mathrm{p}} F)^{-1}q = \bigcup \{f^{-1}q \mid f \in F\}\,.$$

Proof: By Scott property of q and directedness of Fx for every x, $(\bigvee^{\mathrm{p}} F)x = \bigvee(Fx) \in q$ iff there exists $f \in F$ with $fx \in q$. ∎

2.2.4 The Stable Generalized Topology

A subset $p \subseteq B$ of a poset with consistent meets is ***stable*** iff it closed under consistent meets; that is, if $x, y \in p$ and $x \Uparrow y$ then $x \wedge y \in p$. Clearly, closure under binary consistent meets is equivalent to closure under non-empty finite consistent meets. A ***stable open*** of a dcpo with consistent meets is a Scott open that is stable. The ***stable generalized topology*** St assigns to every dcpo with consistent meets B the collection $\mathsf{St}B$ of its stable opens.

The Scott open $\{(\top, \bot), (\bot, \top), (\top, \top)\}$ of $\mathbf{2} \times \mathbf{2}$ is not stable, because it does not contain the meet of the consistent pair $(\top, \bot)$ and $(\bot, \top)$. Since it is the union of the two stable opens $\{(\top, \bot), (\top, \top)\}$ and $\{(\bot, \top), (\top, \top)\}$, this shows that stable opens are not closed under arbitrary unions, and thus do not form a proper topology. Stable opens are nevertheless closed under directed unions:

2.2.4 Proposition *If P is a family of stable opens directed under inclusion then $\bigcup P$ is a stable open.*

Proof: $p_0 = \bigcup P$ is a Scott open. If p_0 contains some consistent pair then by directedness some $p \in P$ must contain that pair, so that the meet is in p and hence in p_0, as desired for stability of p. ∎

2.2.5 Proposition *For algebraic dcpos B and B' with consistent meets,*

(1) $\uparrow x \in \mathsf{St}B$ for every $x \in \mathbf{K}\, B$, and the stable generalized topology has T_0-separation.

(2) $f : B \to B'$ is stable iff it is continuous with respect to the stable generalized topology, i.e., $f^{-1}q$ is stable open for every $y \in \mathbf{K}\, B'$.

(3) $f : B \to B'$ is stable iff $f^{-1}(\uparrow y)$ is stable open for every $y \in \mathbf{K}\, B'$.

Proof: For (1) note that a principal filter $\uparrow x$ for x isolated is closed under all meets, so that it is stable as well as being a Scott open. T_0-separation now follows from algebraicity.

For one direction of (2) and (3), assume that f is stable and $q \in \mathsf{St}B'$, and let $p = f^{-1}q$. p is Scott open, by continuity of f. If p contains a consistent pair $\{x, y\}$ then q contains the pair $\{fx, fy\}$, which is consistent by monotonicity of f, and by stability of q, $fx \wedge fy \in q$. By stability of f, $f(x \wedge y) = fx \wedge fy \in q$, so that $x \wedge y \in p = f^{-1}q$. p is thus stable open.

For the converse direction of (2), assume that $f^{-1}q \in \mathsf{St}B$ for every $q \in \mathsf{St}B'$. f is continuous by part (4) of proposition 2.2.2. If $x \Uparrow y$ then by monotonicity $f(x \wedge y) \leq fx \wedge fy$. Now, let q be a stable open of B' that contains $fx \wedge fy$. Therefore $x, y \in f^{-1}q$, and since $f^{-1}q$ is stable, $x \wedge y \in f^{-1}q$, so that $f(x \wedge y) \in q$. Therefore no stable open separates $f(x \wedge y)$ and $fx \wedge fy$; since stable opens have T_0-separation, this means that $f(x \wedge y) = fx \wedge fy$, and stability of f follows.

To complete the proof of (3), note that the selection of q in the previous paragraph may be restricted to principal filters $\uparrow y$ for isolated y, since these suffice to establish T_0-separation; thus, if $f^{-1}(\uparrow y)$ is stable open for isolated y then the previous paragraph shows that f is stable. ∎

2.2.5 Lobes, Lobe Inclusion, and the Stable Order

We show now that, in analogy with the correspondence between the pointwise order on functions and the inclusion over the powerset (in the sense of propositions 2.2.1 and 2.2.2 (5)), there is a correspondence between the stable order on the function space and a different order on the powerset: lobe inclusion. This also gives rise to the concept of lobes.

A specialized form of lobe inclusion was introduced by Zhang. Other than that the material in this section is new.

We define the ***lobe inclusion*** order $\sqsubseteq$ on subsets of a poset B,

$$X \sqsubseteq Y \quad \text{iff} \quad \forall x \in \uparrow X \,.\, X \cap \downarrow x = Y \cap \downarrow x.$$

An equivalent definition is obtained via a decomposition of subsets $X \subseteq B$ of a poset B into their lobes. If we consider a poset B as a graph, with edges connecting ordered elements, then the lobes of an up-closed set $X \subseteq B$ are the connected regions of X in the graph (seen as undirected); that is, the equivalence classes of X under the relation of "being connected in X". The formal definition, then, is this: if X is a subset of a poset B, then $\Uparrow_X$ is the restriction to X of the consistency predicate of B, namely, for $x, y \in X$, let $x \Uparrow_X y$ iff there exists $z \in B$ with $x, y \leq z$ (and necessarily $z \in X$, if X is up-closed). The transitive closure $\Uparrow_X^*$ of $\Uparrow_X$ is the equivalence relation of "being connected in $\uparrow X$", and the equivalence classes of X with respect to $\Uparrow_X^*$ are called the ***lobes*** of X. Write $\mathsf{lobes}\, X$ for the set of lobes of X. The lobes of a subset of a poset are important conceptually, since they are indivisible constituents of the set for representation purposes.

2.2.6 Proposition *For $X, Y \subseteq B$, lobe inclusion coincides with inclusion of lobes; that is,*

$$X \sqsubseteq Y \qquad \text{iff} \qquad \mathsf{lobes}\, X \subseteq \mathsf{lobes}\, Y.$$

Proof: Each of the conditions implies $X \subseteq Y$.

$\Longrightarrow$ Assume $X \sqsubseteq Y$, and let $r \in \mathsf{lobes}\, X$. Certainly $X \subseteq Y$, so that $r \subseteq Y$. Since r is a single lobe, there must exist a unique lobe $r' \in \mathsf{lobes}\, Y$ such that $r \subseteq r'$. But now if $y \in r' \setminus r$ we may derive a contradiction to $X \sqsubseteq Y$. Thus $r = r' \in \mathsf{lobes}\, Y$, and $\mathsf{lobes}\, X \subseteq \mathsf{lobes}\, Y$.

$\Longleftarrow$ Assume that $\mathsf{lobes}\, X \subseteq \mathsf{lobes}\, Y$, and let $x \in \uparrow X$. Then there is a unique lobe $r \in \mathsf{lobes}\, X$ with $r \cap \downarrow x \neq \emptyset$, and $X \cap \downarrow x = r \cap \downarrow x$. Hence $r \in \mathsf{lobes}\, Y$, and, again by uniqueness, this time for Y, $Y \cap \downarrow x = r \cap \downarrow y$. The desired result follows. ∎

Lobe inclusion implies inclusion. Inclusion of minimal points is another equivalent definition for sets that are determined by their minimal points (sets X such that $X = \uparrow(\mathsf{min}\, X)$: if $X = \uparrow(\mathsf{min}\, X)$ and $Y = \uparrow(\mathsf{min}\, Y)$ then $X \sqsubseteq Y$ iff $\mathsf{min}\, X \subseteq \mathsf{min}\, Y$). Inclusion of minimal points was used by Zhang.

In contrast to the inclusion ordering, the powerset is not a lattice when ordered by lobe inclusion. In particular, A itself is not necessarily a largest element. Nevertheless, the union still serves as a join for subsets that are consistent with respect to lobe inclusion.

An analogue of 2.2.1 holds for the stable order and lobe inclusion:

2.2.7 Proposition *For functions f, g between dcpos with meet, the following are equivalent:*

(1) $f \leq^{\mathrm{s}} g$,

(2) $f^{-1}q \sqsubseteq g^{-1}q$ *for every up-closed stable* $q \subseteq B'$,

(3) $f^{-1}(\uparrow y) \sqsubseteq g^{-1}(\uparrow y)$ *for every* $y \in B'$.

Proof: The implication (2) $\Longrightarrow$ (3) is immediate.

(1) $\Longrightarrow$ (2) Assume $f \leq^{\mathrm{s}} g$ and $q \subseteq B'$ up-closed. This implies $f \leq^{\mathrm{p}} g$, so that by 2.2.1 $f^{-1}q \subseteq g^{-1}q$. Now, let $y \in f^{-1}q$, so that $fy \in q$ and $gy \in q$. If $x \in f^{-1}q \cap \downarrow y$ then $x \leq y$ and $fx \in q$, so that $gx \in q$, and $x \in g^{-1}q \cap \downarrow y$. Conversely, if $x \in g^{-1}q \cap \downarrow y$ then $x \leq y$

and $gx \in q$, and since $fy, gx \in q$ are consistent and q is closed under consistent meets, $fx = fy \wedge gx \in q$, so that $x \in f^{-1}q \cap \downarrow y$. Therefore

$$f^{-1}q \cap \downarrow y = g^{-1}q \cap \downarrow y$$

for every $y \in f^{-1}q$, and $f^{-1}q \sqsubseteq g^{-1}q$.

(3) $\Longrightarrow$ (1) Assume (3). This implies that $f \leq^{\mathrm{p}} g$ by 2.2.1. Let $x \leq y$ and $z = fy \wedge gx$, a consistent meet in B'. Clearly $fx \leq z$. Now, $y \in f^{-1}(\uparrow z)$, and since $f^{-1}(\uparrow z) \sqsubseteq g^{-1}(\uparrow z)$ and $x \in g^{-1}(\uparrow z) \cap \downarrow y$, it must be the case that $x \in f^{-1}(\uparrow z)$, so that $fx \geq z$. Therefore $fx = z$ and $f \leq^{\mathrm{s}} g$. ∎

2.2.8 Proposition *The union of lobe-inclusion consistent sets p_1 and p_2 is their join under lobe inclusion.*

Proof: Let $p_1, p_2 \sqsubseteq p$, that is, $\mathsf{lobes}\, p_1, \mathsf{lobes}\, p_2 \subseteq \mathsf{lobes}\, p$. Every lobe of p_1 is also a lobe of p, so it is either a lobe of p_2, or is disjoint from p_2. The same argument may be applied in the converse direction. Thus, $\mathsf{lobes}(p_1 \cup p_2) = \mathsf{lobes}\, p_1 \cup \mathsf{lobes}\, p_2$, and it follows that $p_1 \cup p_2$ is lobe-inclusion dominated by p, and it is thus the lobe-inclusion join of p_1 and p_2. ∎

We may also talk of lobes in isolation, and not only with respect to some decomposition of a subset. Define a ***lobe*** to be a subset of a poset that has a single lobe, *i.e.*, X such that $\mathsf{lobes}\, X = \{X\}$. It is easy to see that a lobe is a set X such that for any pair of elements $x, y \in X$ there exists a finite sequence of elements of X that contains x and y, in which all adjacent pairs of elements are consistent. We call such a sequence a ***consistent sequence***.

Stable lobes — lobes closed under consistent meets — will be of particular interest. The next proposition shows that monotone stable lobes are filters.

2.2.9 Proposition *A monotone stable lobe is closed under meets. (Recall that meets are taken over non-empty finite sets.)*

Proof: We prove by induction on the length of a consistent sequence c contained in a monotone stable lobe p that its meet $\wedge c$ is in p. For c of length one this is immediate. For c of length $n+1$, c is obtained from some length n consistent sequence c' and an additional element, y that is consistent with an element of c', and hence with $\wedge c'$. Now, $\wedge c' \in p$ by the induction hypothesis, and by stability the bounded meet $(\wedge c') \wedge y = \wedge c$ is in p.

Now, if X is a non-empty finite subset of a monotone stable lobe then there is a consistent sequence c in p that contains X (just piece together consistent sequences for pairs of elements of X), and by the above $\wedge c \in p$, with $\wedge X \in p$ following by monotonicity. ∎

2.3 Categories

We use here some basic concepts from category theory as aids to the organization and presentation. Our use of categorical concepts and terminology will be at a very shallow level; we are mainly interested in categories as a concise way of stating what framework or model we are talking about at a given moment, and as a way of stating some basic (packages of) properties about frameworks of this sort. This is very much a book-keeping application, but it is essential in work such as this that revolves about varying the notions of functions employed, the underlying objects, and the orders used on the function space. For some introductory reading in the subject we suggest [Pie91, BW90], and the authoritative reference is [ML71].

A ***category*** consists of a collection of ***objects*** and a collection of ***arrows*** or ***morphisms***, such that each morphism is assigned a domain object and a codomain object. We may write $f : A \to_{\mathcal{A}} A'$ to indicate that the morphism f of the category $\mathcal{A}$ has domain A and codomain A'. A category also has a composition operation $\circ^{\mathcal{A}}$ on arrows, such that for $f : A \to_{\mathcal{A}} A'$ and $f' : A' \to_{\mathcal{A}} A''$, $f' \circ^{\mathcal{A}} f : A \to A''$. The composition must be associative, so that $f'' \circ^{\mathcal{A}} (f' \circ^{\mathcal{A}} f) = (f'' \circ^{\mathcal{A}} f') \circ^{\mathcal{A}} f$ for all f, f' and f'' with suitable domains and codomains. Finally, for every object A there must be an identity morphism $\mathsf{id}^{\mathcal{A}}_A : A \to_{\mathcal{A}} A$ that serves as unit for composition: $\mathsf{id}^{\mathcal{A}}_{A'} \circ^{\mathcal{A}} f = f = f \circ^{\mathcal{A}} \mathsf{id}^{\mathcal{A}}_A$ for every $f : A \to A'$.

A ***subcategory*** of a category $\mathcal{A}$ is a category $\mathcal{B}$ such that every object of $\mathcal{B}$ is an object of $\mathcal{A}$, and every morphism of $\mathcal{B}$ is a morphism of $\mathcal{A}$ and is assigned the same domain and codomain in $\mathcal{A}$ and $\mathcal{B}$, with composition and identities in $\mathcal{B}$ coinciding with composition and identities in $\mathcal{A}$, so that, $\mathsf{id}^{\mathcal{B}}_A = \mathsf{id}^{\mathcal{A}}_A$ for every object A of $\mathcal{B}$, and $f' \circ^{\mathcal{B}} f = f' \circ^{\mathcal{A}} f$ for all $f : A \to_{\mathcal{B}} A'$ and $f' : A' \to_{\mathcal{B}} A''$. $\mathcal{B}$ is a ***full*** subcategory of $\mathcal{A}$ if in addition every $\mathcal{A}$ arrow $f : A \to_{\mathcal{A}} A'$ between objects A and A' that are $\mathcal{B}$-objects is also a $\mathcal{B}$ arrow.

As an example, take the category SET whose objects are sets and whose morphisms are functions; composition is taken to be ordinary function composition, and the identity morphism for a set is the identity function over that set. Another category is PO, the category of posets and monotone functions between them, with set-theoretic composition and identities. A third category is CT, the category of dcpos and continuous functions between them, again with set-theoretic composition and identities. Note that CT is a subcategory of PO, but PO is not, strictly speaking, a subcategory of SET, since its objects have additional structure not present in the objects of SET.

The above are examples of ***concrete categories*** — categories whose objects are sets, possibly endowed with extra structure, and whose morphisms are functions, with set-theoretic composition and identities. We will work here exclusively with concrete categories, and will employ exclusively the set-theoretic operations, including operations connected with products and function spaces.

Additional concrete categories that we define at this point are DCT, the category of Scott domains and continuous functions; ST, the category of dcpos with consistent meets and stable functions; and DI, the category of dI-domains and stable functions.

A simple example of a typical category-theoretical definition is the definition of isomorphism between objects of a category. An arrow $f : A \to_{\mathcal{A}} A'$ is said to be an ***isomorphism*** between A and A' in $\mathcal{A}$, denoted $f : A \cong_{\mathcal{A}} A'$, iff there exists an arrow $g : A' \to_{\mathcal{A}} A$ such that

$g \circ f = \mathsf{id}_A^{\mathcal{A}}$ and $f \circ g = \mathsf{id}_{A'}^{\mathcal{A}}$. A and A' are said to be ***isomorphic*** in $\mathcal{A}$, denoted $A \cong_{\mathcal{A}} A'$, iff there is an isomorphism $f : A \cong_{\mathcal{A}} A'$. It is easy to check that an isomorphism in SET is simply an isomorphism of sets, *i.e.*, a 1-1 correspondence, and an isomorphism in PO is a conventional order-isomorphism, *i.e.*, an isomorphism of sets that respects the partial order (in the sense of being monotone).

Rather than presenting categorical definitions in their full glory, we will only relate here to their incarnation for concrete categories.

2.3.1 Categorical Product

A concrete category is said to have products iff there is a product operation $\times$ on objects that assigns to every pair of objects A_1 and A_2 their product $A_1 \times A_2$, which is the cartesian product of the underlying set, together with additional structure as appropriate; and moreover, the (set-theoretical) product projections $\pi_{1;A_1,A_2}$ and $\pi_{2;A_1,A_2}$ are morphisms in the category; and moreover, for any object A and two morphisms $f_1 : A \to A_1$ and $f_2 : A \to A_2$, the pairing $\langle f_1 \,, f_2 \rangle = \lambda x \in A \,.\, (f_1 x, f_2 x)$ is a morphism of the category.

For instance, in PO the standard product object is taken to be the cartesian product of the underlying sets, ordered componentwise; and it is easy to check that the projections are monotone, and that pairing of functions preserves monotonicity.

The definition is usually presented via the product diagram:

$$
\begin{tikzcd}
A_1 & A_1 \times A_2 \arrow[l, "\pi_{1;A_1,A_2}"'] \arrow[r, "\pi_{2;A_1,A_2}"] & A_2 \\
& A \arrow[ul, "f_1"] \arrow[u, "\langle f_1 \,, f_2 \rangle"] \arrow[ur, "f_2"'] &
\end{tikzcd}
$$

The diagram is required to commute, meaning that every pair of directed paths between two objects in the diagram must yield an equal composition. Because we use concrete categories, we inherit this property from SET (where it can be verified for the usual set-theoretic definitions), so all we need to worry about is that the product object be an object of the category, and that the projections and pairing be morphisms of the category.

The above is the binary product, and finite products can be built up by iterating it. Thanks to associativity (the isomorphisms assl and assr), the order in which the iteration takes place is immaterial. The nullary product in a concrete category is a singleton set $\mathbf{1}$ (with structure as appropriate), such that for every object A, the unique constant function from A to $\mathbf{1}$ is a morphism of the category.

A concrete category that has products is said to be ***cartesian***. The product structure is not determined uniquely — there is some latitude in the choice of a terminal object, or possibly in the choice of extra structure for the product object — but it is unique up to isomorphism.

2.3.2 Exponentiation

A cartesian concrete category $\mathcal{C}$ (with a fixed choice of product structure) has exponentiation iff there is a binary operation $\to_{\mathcal{C}}$ on objects that maps any two objects A and A' to an exponentiation object $A \to_{\mathcal{C}} A'$ that is some function space between the sets underlying A and A', with extra structure as appropriate; and, moreover, the application function $\mathsf{app} : (A \to_{\mathcal{C}} A') \times A \to A'$ is a morphism in the category; and, moreover, for every object B and function $f : B \times A \to A'$, the function $\mathsf{curry}\, f : B \to (A \to_{\mathcal{C}} A')$ is a morphism of the category.

Equivalently, one may ask that $A \to_{\mathcal{C}} A'$ exist, for any two objects, and that currying and uncurrying be well defined when restricted to morphisms of the category (of appropriate types), *i.e.*, that currying and uncurrying preserve the property of being a morphism.

For example, if one takes the category DCT of Scott domains and continuous functions, then we have already noted that the pointwise ordered continuous function space between Scott domains is a Scott domain, application is continuous, and currying and uncurrying preserve continuity.

The diagram corresponding to exponentiation is the following:

$$\begin{tikzcd}
(A \to_{\mathcal{C}} A') \times A \arrow[r, "\mathsf{app}"] & A' \\
B \times A \arrow[u, "(\mathsf{curry}\, f) \times \mathsf{id}"] \arrow[ur, "f"'] &
\end{tikzcd}$$

A cartesian category that has exponentiation is called a ***cartesian closed category***, or a CCC for short.

2.3.3 Self-enriched Categories

The ***homset*** of two objects A and A' in a category $\mathcal{C}$ is the set of $\mathcal{C}$-morphisms from A to A'.

Since the homset is just a set, it will not have the extra structure that may be needed for objects in our concrete category. Therefore, it will not usually be possible to say whether operations defined on homsets — composition, pairing and product of functions, application, currying and uncurrying, and the exponent of functions — are in the category. For instance, it will not be possible to say whether composition is monotone if no order is defined on the homset.

In conventional treatments the next step after the definition of a CCC is then to define an *order-enriched* category, or a *dcpo-enriched* category, a CCC where the homset between any two objects is a poset or a dcpo, respectively. Once this is done, it is possible to judge properties such as monotonicity, continuity, stability and strictness of operations on homsets. This is also useful for the solution of recursive domain equations.

We are interested, however, in more restricted notions of functions, for which extra structure beyond the order is necessary — for instance, a bidomain would require an additional order. What we really need, then, is to treat the homset — which is just a function space — as an object of the category, by specifying the extra structure needed. For this reason we need to show that the category is closed under the function space. This will usually be done when we define the exponentiation, so that not much effort is involved now (thanks to our concrete setting): we add to the homset the structure for an exponentiation object.

Thus, rather than just order-enriching or dcpo-enriching the category, we need to enrich it over itself: a homset needs to be treated as an object of the category. This is known as a ***self-enriched*** category. (This is a special case of the more general concept of *enriched categories*. See [ML71], or for much more information [Kel82].)

2.4 PCF and the Full Abstraction Problem

PCF is a programming language for computable functions, originating with LCF, Scott's logic of computable functions [Sco69, Plo77]. It is a simply typed lambda calculus extended with recursion and with operations over primitive types. It is usually considered as a paradigmatic sequential programming language.

2.4.1 The PCF Language

Let ρ range over a pre-determined collection of primitive, or ground, types, and τ range over types, defined to be the elements of the least set that contains the primitive types and is closed under the arrow constructor: if τ and τ' are types then so is $\tau \to \tau'$. We let the arrow associate to the right so that $\tau \to \tau' \to \tau'' = \tau \to (\tau' \to \tau'')$.

Let c range over some collection of constants, and v range over a collection of identifiers (or, variables). Let M range over raw terms, defined to be the elements of the least set that contains the constants and the identifiers, and that is closed under formation of application and abstraction terms: an application is a term of the form MM' and an abstraction is a term of the form $\lambda v : \tau \,.\, M$, for terms M and M', identifier v and type τ. We let application associate to the left, so that $MM'M'' = (MM')M''$.

An occurrence of an identifier v in a term M may be either *free* or *bound*; the occurrence is free if it is not bound, and the occurrence is bound if it is bound in some subterm of M in which it occurs, or if $M = \lambda v \,.\, M'$. An identifier is free (or bound) in a term iff it has a free (or bound, respectively) occurrence in the term. An identifier may be both free and bound, if it has both free and bound occurrences in the same term. A closed term is a term that has no free identifiers.

Every constant is assumed to have a uniquely determined type. In contrast with early presentations (*e.g.*, in [Plo77]), we do not assume that identifiers have an intrinsic type, but rather, we will use type contexts that assign types to identifiers. This is convenient when giving a categorical model for the language. A type context Γ is a finite ordered list of identifier-type pairs, and we write $\Gamma, v : \tau$ for the type context obtained by extending Γ so that it assigns type τ to v. Identifiers may occur more than once in a type context, and the rightmost occurrence always takes precedence for the purpose of determining the type assigned to an identifier by a type context.

The typing system for PCF is defined below by axioms and inference rules for judgements of the form $\Gamma \vdash M : \tau$, to be read as: the term M has type τ in type context Γ.

- Constants:

$$\frac{}{\Gamma \vdash c : \tau}\,\mathsf{const}$$

for every constant c of type τ.

- Variables:

$$\frac{}{\Gamma, v : \tau \vdash v : \tau}\,\mathsf{var}$$

$$\frac{\Gamma \vdash v : \tau}{\Gamma, v' : \tau' \vdash v : \tau}\,\mathsf{var}' \quad (\text{provided } v \neq v')$$

- Application:

$$\frac{\Gamma \vdash M : \tau_0 \to \tau \quad \Gamma \vdash M_0 : \tau_0}{\Gamma \vdash M M_0 : \tau}\,\mathsf{app}$$

- Abstraction:

$$\frac{\Gamma, v : \tau \vdash M : \tau'}{\Gamma \vdash (\lambda v : \tau \,.\, M) : \tau \to \tau'}\,\mathsf{abs}$$

The terms of PCF are those raw terms M for which a judgement $\Gamma \vdash M : \tau$ is derivable. It can be shown by induction on the structure of a term that a term has at most one type per type context, and there is a unique type derivation for the corresponding judgement. Also, it can be checked that if Γ_1 and Γ_2 are type contexts that agree over the free identifiers of a term M then $\Gamma_1 \vdash M : \tau$ is derivable iff so is $\Gamma_2 \vdash M : \tau$. Moreover, a term is well-typed and closed iff it may be given a type in the empty type context, and that type is then the unique type derivable for the term.

2.4.2 Categorical Models for PCF

If $\mathcal{C}$ is a concrete cartesian-closed category then an interpretation $[\![-]\!]_0$ of PCF's primitive types as $\mathcal{C}$ objects is lifted to an interpretation $[\![-]\!]$ of all PCF types as $\mathcal{C}$ objects by taking $[\![\rho]\!] = [\![\rho]\!]_0$, and other types are interpreted inductively by mapping the arrow constructor to the exponentiation in $\mathcal{C}$, *i.e.*, $[\![\tau \to \tau']\!] = [\![\tau]\!] \to_{\mathcal{C}} [\![\tau']\!]$. (The interpretation may be extended to product types by mapping the product constructor to the product on objects.)

In a second step, an interpretation $[\![-]\!]_0$ of PCF constants, such that $[\![c]\!]_0 \in [\![\tau]\!]$ for every PCF constant of type τ, may be lifted to provide a model $[\![-]\!]$ of PCF in $\mathcal{C}$. We define a meaning function $[\![-]\!]$ over derivable judgements $\Gamma \vdash M : \tau$ by induction on the derivation of the judgement, so that

$$[\![\Gamma \vdash M : \tau]\!] : [\![\Gamma]\!] \to_{\mathcal{C}} [\![\tau]\!],$$

where $[\![\Gamma]\!]$ is an object representing environments (mappings of identifiers to values) "compatible" with type context Γ, as defined by $[\![\Gamma]\!] = \mathbf{1}$ if Γ is the empty type context, and

$$[\![\Gamma, v : \gamma]\!] = [\![\Gamma]\!] \times [\![\gamma]\!].$$

The meaning function $[\![-]\!]$ is defined as follows:

$$\begin{array}{rcll}
[\![\Gamma \vdash c : \tau]\!] & = & \mathsf{K}[\![c]\!]_0 & \\
[\![\Gamma, v : \tau \vdash v : \tau]\!] & = & \pi_2 & \\
[\![\Gamma, v' : \tau' \vdash v : \tau]\!] & = & [\![\Gamma ; x : \tau]\!] \circ \pi_1 & (v \neq v') \\
[\![\Gamma \vdash M M_0 : \tau]\!] & = & \mathsf{app} \circ \langle [\![\Gamma \vdash M : \tau_0 \to \tau]\!], [\![\Gamma \vdash M_0 : \tau_0]\!] \rangle & \\
 & = & \lambda \eta \in [\![\Gamma]\!] \,.\, [\![\Gamma \vdash M : \tau_0 \to \tau]\!] \eta ([\![\Gamma \vdash M_0 : \tau_0]\!] \eta) & \\
[\![\Gamma \vdash (\lambda v : \tau \,.\, M) : \tau \to \tau']\!] & = & \mathsf{curry}[\![\Gamma, v : \tau \vdash M : \tau']\!]. &
\end{array}$$

It can be checked that the function $[\![-]\!]$ is well defined on all derivable judgements. Note that every derivable judgement has a unique derivation, and, in particular, τ_0 mentioned in the application clause is uniquely determined.

2.4.3 The Standard Base Interpretation

Thus far we have assumed a parameterized collection of primitive types and primitive operations. We present now the standard collections used, and their standard interpretations. These particular choices impose some requirements on any prospective model.

The standard primitive types are the type `Nat` of natural numbers and the type `Bool` of booleans, interpreted by $[\![-]\!]_0$ as the flat posets **Nat** and **Bool** of naturals and booleans, respectively (see figure 2.1).

The standard constants and their types and meanings by $[\![-]\!]_0$ are listed next.

- A constant of type ρ for each element of each ground type ρ. That is,
  - A numeral $\bar{n}$ of type `Nat` for each natural number n.
  - Constants T and F of type `Bool`, denoting the corresponding truth values.
  - For each ground type ρ a constant Ω^ρ of that type, denoting the bottom element of $[\![\rho]\!]_0$.
- For each ground type ρ, a conditional constant $\mathtt{if}^\rho$ of type $\mathtt{Bool} \to \rho \to \rho \to \rho$ such that

$$[\![\mathtt{if}^\rho]\!]_0 bxy = \begin{cases} x & b = \mathsf{T} \\ y & b = \mathsf{F} \\ \bot_{[\![\rho]\!]_0} & b = \bot. \end{cases}$$

  To improve readability, we may write applications of the conditional constant such as $\mathtt{if} M M' M''$ using keywords `then` and `else`, as in `if` M `then` M' `else` M''.

- Arithmetic constants: `(+1)` and `(-1)` of type $\mathtt{Nat} \to \mathtt{Nat}$, `(=0)` of type $\mathtt{Nat} \to \mathtt{Bool}$, denoting the successor and predecessor functions and the equal-to-zero predicate, respectively. For the sake of definiteness we decree that the predecessor of 0 is undefined, and set $[\![\mathtt{(-1)}]\!]_0 0 = \bot$ (but any other value could have been chosen).

Finally, recursion is added by means of a recursion constant $\mathtt{Y}^\tau$ for every type τ. $\mathtt{Y}^\tau$ has type $(\tau \to \tau) \to \tau$, is interpreted by the fixed-point operation:

$$[\![\mathtt{Y}^\tau]\!]_0 = \mathsf{fix}_{[\![\tau]\!]} : [\![\tau \to \tau]\!] \to [\![\tau]\!].$$

Note that in order to interpret the fixed-point combinator, the fixed-point operation must be well defined over the chosen exponentiation — it is well-defined, for instance, over continuous functions between dcpos — and, in the formulation given here, must itself be a morphism in the category.

Also, note that the standard interpretation of the fixed-point constant already relies on the definition of the function space in the category in which we wish to define a model.

The Ω constants are superfluous in the presence of recursion, since $\mathtt{Y}^\rho(\lambda v : \rho \,.\, v)$ is a term denoting the bottom element of $[\![\rho]\!]_0$.

The first and best known model of PCF is the continuous functions model [Sco69, Plo77], which is essentially the model constructed as above in the category of Scott domains and Scott-continuous functions between them.

2.4.4 Adequacy and Full Abstraction

By convention, one may only observe the behavior of ***PCF programs***, which are taken to be the closed PCF terms of ground type. Observation here means that one obtains the value of the term (potentially including an indication that the value is undefined).

Let a behavior be a meaning function defined on programs that maps programs of ground type ρ into $[\![\rho]\!]_0$ — a flat poset in the standard interpretation, where a value of $\bot_{[\![\rho]\!]_0}$ indicates an undefined value. Such a behavior $\mathcal{O}[\![-]\!]$ induces an observational equivalence and an observational pre-order on terms of PCF. Intuitively, we say that a term M observationally approximates a term M' iff for every PCF program, replacing occurrence(s) of M by M' yields a program that is defined, and has the same value, whenever the original program is defined.

The replacement is formalized by means of program contexts. A program context $P[-]$ is a term P that has exactly one free identifier, referred to as the hole and denoted by $-$ (this is usually considered to be distinguished and cannot be bound). For a program context $P[-]$ and a term M, the term $P[M]$ is the term obtained by replacing[4] the free occurrences of $-$ in P by the term M; if $P[M]$ is a program (with ground type ρ) then we say that M is compatible for $P[-]$ (with type ρ).

Thus, given a behavior $\mathcal{O}[\![-]\!]$, we say that M observationally approximates M', denoted $M \leq_{\mathcal{O}} M'$, iff for every program context $P[-]$, if M and M' are compatible for $P[-]$ (both

[4] Free variables of M may be captured by bindings in P, in contrast to the usual capture-avoiding substitution. This is not essential, however, since the context lemma [Mil77, Ber79, JM91] entails that applicative contexts $Q[-] = Q-$ (with Q closed) suffice for defining the observational pre-order.

with type ρ) then $\mathcal{O}[\![P[M]]\!] \leq_{[\![\rho]\!]_0} \mathcal{O}[\![P[M']]\!]$. M and M' are observationally equivalent, denoted $M =_{\mathcal{O}} M'$, iff each of them observationally approximates the other: $M \leq_{\mathcal{O}} M'$ and $M' \leq_{\mathcal{O}} M$.

A semantics $[\![-]\!]$ is ***adequate*** with respect to a behavior $\mathcal{O}[\![-]\!]$ iff observational similarity of terms implies similarity of their meanings, and ***fully abstract*** if observational similarity of terms coincides with similarity of their meanings. For similarity one may take either equality (yielding *equational* adequacy and full abstraction) or inequality (yielding *inequational* adequacy and full abstraction, also referred to as *order* or *approximation* adequacy and full abstraction). The inequational notions imply their equational counterparts, and following most work in the field we will only consider here the more demanding inequational version[5]. Note that this pre-supposes that the objects of $\mathcal{C}$ are equipped with partial orders. We thus have:

- Adequacy:

$$M \leq_{\mathcal{O}} M' \implies [\![\Gamma \vdash M : \tau]\!] \leq_{[\![\Gamma]\!] \to_{\mathcal{C}} [\![\tau]\!]} [\![\Gamma \vdash M' : \tau]\!]$$

  for all terms M and M' and type context Γ under which both have type τ.

- Full abstraction:

$$M \leq_{\mathcal{O}} M' \text{ iff } [\![\Gamma \vdash M : \tau]\!] \leq_{[\![\Gamma]\!] \to_{\mathcal{C}} [\![\tau]\!]} [\![\Gamma \vdash M' : \tau]\!]$$

  for all terms M and M' and type context Γ under which both have type τ.

The Full Abstraction Problem

Plotkin showed that the continuous functions model, while being adequate, fails to be fully abstract (with respect to the standard notion of behavior) because it contains non-definable parallel elements such as parallel-or [Plo77]. One may write in PCF higher-order functionals whose meanings differ in their action on parallel-or, and yet, such that no context would be able to tell the terms apart, precisely because parallel-or is not definable, as was observed by Scott [Sco69].

Milner has shown that a fully abstract model does exist, and that (up to isomorphism) there is a unique (inequationally) fully abstract model that is order-extensional and continuous (*i.e.*, such that the function space is ordered pointwise, and consists of continuous functions) [Mil77]. Moreover, full abstraction for PCF may be reduced to definability of isolated elements: An order-extensional continuous model is fully abstract iff every isolated element is definable.

Milner's construction of the fully abstract model was syntactic in nature, however, and our ultimate goal in this work is to provide a language independent, synthetic construction of the fully abstract model, that identifies and contributes to the understanding of sequential computation.

For a recent survey of work on the problem of full abstraction for PCF, as well as comprehensive bibliography, one is referred to [Ong95]. Full abstraction in a context wider than PCF is considered in [Sto88].

[5] For a treatment of equational full abstraction see for instance [Sto90].

2.4.5 Operational Semantics of PCF

In the standard treatment, PCF is given a behavior via an operational semantics that determines how to evaluate PCF terms. A program that evaluates to a (uniquely determined) value of the appropriate ground type is assigned (the meaning of) that value as its meaning, and a program that fails to evaluate to a value is assigned bottom as its meaning. This defines a behavior $\mathcal{O}[\![-]\!]$, against which one may judge adequacy and full abstraction of some model $[\![-]\!]$ in a category $\mathcal{C}$.

We remark that this assessment may be broken into two separate steps. First, one may consider the model $[\![-]\!]$ in isolation, and ask whether it is adequate or fully abstract with respect to *itself* as a notion of behavior. This, in itself, functions as a test for the model, independently of the operational semantics. Secondly, one may show that the notion of behavior induced by the model coincides with that obtained from the operational semantics, and thus the results of the first step translate directly into results with respect to the standard notion of behavior.

For completeness we list here a transition relation that defines the standard operational semantics of PCF. We write $M[N/v]$ for the term obtained by substituting N for free occurrences of v in M (using usual capture-avoiding substitution).

$$
\begin{array}{rcl@{\qquad}rcl}
(\lambda v:\tau\,.\,M)N & \to & M[N/v] & \mathtt{Y}M & \to & M(\mathtt{Y}M) \\
\mathtt{if}\ \mathsf{T}\ \mathtt{then}\ M\ \mathtt{else}\ M' & \to & M & \mathtt{if}\ \mathsf{F}\ \mathtt{then}\ M\ \mathtt{else}\ M' & \to & M' \\
(\mathtt{+1})\bar{n} & \to & \overline{n+1} & (\mathtt{-1})\overline{n+1} & \to & \bar{n} \\
(\mathtt{=0})\bar{0} & \to & \mathsf{T} & (\mathtt{=0})\overline{n+1} & \to & \mathsf{F}
\end{array}
$$

$$
\frac{M \to M'}{MN \to M'N} \qquad \frac{M \to M'}{cM \to cM'}
$$

The operational semantics is now defined for programs of type `Nat` by setting $\mathcal{O}[\![P]\!] = n$ if $P \to^* \bar{n}$, and $\mathcal{O}[\![P]\!] = \bot$ if no such n exists. Similarly for programs of boolean type.

Chapter 3

Indexed Computation

This chapter presents a general framework for the semantic treatment of sequentiality, derived from an abstract operational view of computation as an incremental process. The individual steps of computation over a domain are formalized as *indices*, drawn from an *index structure* for the given domain. Index structures on domains may be used to define a notion of *indexable functions* — essentially functions that use only valid indices, and thus obey the computational assumptions formalized by the index structures chosen. With this framework of indexed computation in place, the search for a suitable notion of sequential functions may be rephrased as a search for suitable index structures. This chapter presents the framework of indexed computation, and subsequent chapters engage in the search for index structures suitable for formalizing sequential computation in PCF.

The nature of this chapter is abstract, and the two examples of index structures in this chapter are trivial and do not formalize sequentiality in any way. A first non-trivial example is given in chapter 4, where the link is made with the Kahn-Plotkin definition of sequentiality. The reader might wish to return and re-read this chapter following a reading of chapter 4.

3.1 Incremental Computation

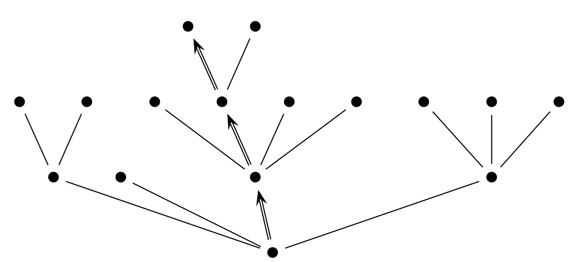

Figure 3.1: A schematic incremental computation

Central to our approach is an abstract operational view of computation as an *incremental process* of obtaining, generating, or communicating information. A computation proceeds over a set of values, each of which represents some information. The values may be partially ordered by their information content. As the computation of a value proceeds, the information available about the value is increased. Thus, there is a need to be able to talk about partial values, or approximations to values; we will make no distinction here between values and partial values. The important points to note are:

- At each point during the course of the computation, there is some notion of a *current approximation* of the value being computed; this characterizes the *state* of the computation.
- The computation proceeds by incrementing this approximation in *discrete steps*.
- The *value* computed is completely described as the sum of its pieces: it is the least upper bound in the information ordering of the individual pieces of information encountered during the computation.

It is important that thing may go wrong: if one of the steps of a computation fails or diverges, so should the entire computation. After all, if nothing can go wrong, then every way of carrying out the computation is equally applicable, and the characterization of sequentiality ceases to be a concern.

Any computation over the booleans, for example, starts with no information about the value being computed, and trivially computes either T or F in one computational step (or diverges). One possible sequential computation over ordered pairs of booleans may start out with no information, compute the left component, and then the right component, in two steps. A parallel computation over the same domain may compute both the left and the right components in one step.

Computation as described so far can be seen abstractly as taking place over some domain of values, and presumably performed by some computational agent. Somewhat more concretely, it is useful to see such computation as taking place at the interface between two computational agents in a producer/consumer relationship: information about the value being computed is passed from the producer to the consumer. It is for this reason that we may regard computation as either *obtaining*, *generating* or *communicating* information: the difference is solely in the vantage point of an external observer that may describe the computation. The domain of values being computed over is, in a sense, a specification of the interface between the producer and the consumer.

A distinction may now be made between *data-driven* and *demand-driven* computation. In a data-driven computation, it is the producer that determines how to go about incrementing the information, while in demand-driven computation, this is the consumer's decision. The demand-driven view is more adequate for our needs.

A function $f : A \to A'$ (or rather, a computational agent computing such a function — but we shall not be careful with this distinction) may be regarded as a filter: it consumes an input over A and produces an output over A'. As we shall show, the relationship between the computations that the function carries out over its input and output (as the consumer and producer, respectively) is instrumental in identifying the compliance of the function with certain computational disciplines or assumptions. Under the demand-driven discipline, this essentially means that a function is computable under certain assumptions if, for every compliant computation over its output, there is a compliant computation that may be carried out over its input so as to enable the agent computing the function to respond to the demands of its consumer. Thus, it may be possible to characterize some classes of functions by designating classes of permissible computations, together with a designation of a desired relationship between computations over the functions input and output.

For instance, the assumptions that only a finite amount of information may be obtained in finite time (and represented in finite space), and that values with an infinite content of information may be said to be computed as the limits of finite computations, lead to a characterization of the Scott continuous functions; the permissible computations are those that maintain a current approximation with a finite information content, and every finite piece of (the computation of) the output must depend on a finite piece of (the computation of) the input. Under these assumptions, the computation of a finite element (*i.e.*, one with a finite content of information) is essentially trivial. (Note that this does not imply that computation of a continuous function must map a finite-information input to a finite-information output!)

As this example points out, the view presented here of computation as an incremental process of information exchange is present, to differing degrees of explicitness, in previous work, starting with the basic intuitions of the Scott-Strachey approach [Sco70, Sto77], and including work on dataflow networks [KM77, PS87], Kahn and Plotkin's definition of sequentiality [KP78], subsequent work by Berry and Curien on sequential algorithms [BC82, Cur93], and work by the Siberian school [Saz76c, Saz76b, Tra85].

3.2 Indices in Computation

In this section we consider what it takes to formalize or capture sequential computation. The considerations leading to the imposition of Scott continuity remain valid, so that we will start from Scott continuous functions and computation, and attempt to obtain a suitable restriction. To do so, however, requires that computations have much more structure. In this section we proceed to specialize the general view of demand-driven computation, and obtain a framework of *indexed computation*, which we will subsequently show to be suitable for the desired formalization.

We start by asking how a consumer may direct the progress of a demand-driven computation. As we have described demand-driven computation, the consumer should communicate to the producer some query about the value being computed, and the producer should reply with an answer to the query. Failure of the producer to give an answer causes divergence of the computation. The queries that may be posed by the consumer correspond intuitively to the primitives available for computation in a language or model of computation. Thus, we view each step of the computation as an invocation of one of a set of relevant primitives resulting in a choice from a set of alternative potential increments.

For instance, over pairs of booleans we might have the primitives "compute the left component" and "compute the right component", and the latter may result in either "the right component is T", or "the right component is F". Over a domain of values that have a color attribute, a possible query might be "what is the color?", and possible answers include "black" and "blue". We formalize this by designating a permissible collection of such atomic computational actions — an individual *index* presents a set of alternative increments, with the computation of an index corresponding to a choice of an actual increment among the potential ones; and an *index structure* designates the valid indices over a domain.

Intuitively, an index structure ought to reflect restrictions on the primitives available in the language or in the model — corresponding, say, to the restriction of PCF to employ only sequential primitives; in fact, a main effort of this work is the identification of classes of index structures that capture sequential computation in PCF.

Under the assumption of sequentiality, each computation exhibits a *sequence* of discrete computational steps. The sequencing conveys timing information (or perhaps causality or precedence information) for the queries issued. (This is in contrast to, say, concurrent or distributed computation, where linear sequencing of the individual steps of the computation may need to be relaxed.)

We remark here that looking in isolation at the traces — sequences of approximations encountered during a computation — is not a sufficient criterion for the validity of computational agents. To see this, note that while each individual trace of parallel-or may be exhibited by some sequential function, there is no sequential function that exhibits all of parallel-or's traces. Instead, one needs to give the computation a more structured description by explicitly considering the queries issued during the computation. Taking the ensemble of potential computations, one obtains a tree, whose nodes are indices, representing queries, with a sub-tree at each of the alternative answers to the query. Such a tree may be referred to informally as a *(sequential) computation strategy*. Again, this concept is present to differing degrees of explicitness or in different guises in previous work on sequential computation, for instance [Mil77, Saz76c, Cur93].

Note also that increments to the current approximation during a sequential computation must be atomic, in the sense that they may indeed be computed in a single computational step (but not necessarily in the sense of being minimal or indecomposable). Atomicity precludes the simultaneous issuing of multiple queries, or "hiding" the parallel occurrence of several distinct increments as a single compound increment; this would rule out the computation of both components of a pair of booleans in a single step. In contrast to the continuous paradigm, this means that even values with a finite content of information may require computation by non-trivial sequences of atomic increments (although one could expect the sequence to be finite for finite values). Thus, one of the roles of index structures will be to provide a formalization of the atomicity of the increment at each computational step. Again, this is in analogy with the primitive operations of a language or a model, that define its atoms of computation.

With index structures in place, we obtain the framework of *indexed computation*, where we restrict attention to computations that use permissible indices. The class of valid indexed computations is of course relative to the choice of index structures.

A word is in order concerning the distinction between *intrinsic* and *extrinsic* index structures. There are cases when domains may be uniformly equipped with an index structure; for instance, concrete domains have an intrinsically determined index structure, namely, their cells. Chapter 4 will show that this may be uniformly and conservatively extended to dI-domains, yielding a notion of sequentiality that captures first-order computation in PCF, and computation at all orders in Berry and Curien's CDS framework. Section 4.4 will show, however, that for the proper treatment of higher-order PCF-sequentiality we need (at least initially) to parameterize the index structure, so that the objects we work with are domains equipped with an extrinsic index structure and, importantly, the function space may be equipped with an index structure that is adequate for higher-order computation. It is to accommodate this need that we take an index structure as not necessarily being fully determined by the domain.

3.3 Index Structures

This section defines index structures, following the intuitions discussed above. Some intuitive requirements are imposed on index structures at this abstract level, making it possible to give a general development.

Indices are taken to be abstract entities that represent queries. We regard a query as being completely described by the possible answers it may be given, and represent this by a decomposition of indices into prime constituents, appropriately called answers.

Each index has an associated *extent* — the set of elements of the underlying set that answer the query posed by the index, or, using terminology borrowed from CDSs, the elements that *fill* the index. For instance, the extent of the index representing a query about the color consists of all elements that have some color attribute. The extent is defined by means of a relation $\checkmark$ of *filling* between indices and the elements of the domain. For most purposes, an index may be regarded as synonymous with its extent; but in fact we do not make this identification formally, and keep indices as abstract objects. This allows us to give a more intensional presentation of indices, that includes details beyond their extent. For higher-order computation, for instance, indices will include information specifying their building blocks.

3.3.1 Definition An ***index structure*** $\mathcal{I} = (I, \checkmark, \mathbf{Ans})$ over a set A consists of a set I of ***indices*** over A, together with a ***filling*** relation $\checkmark \subseteq I \times A$ between indices and elements of A, and an ***answer*** map $\mathbf{Ans} : I \to \mathcal{P}\, I$, subject to the constraints (IS1), (IS2) and (IS3) given below.

r is said to be ***filled*** by x when $r \checkmark x$. This indicates that r may be seen as having been already computed if x is known, or, in other words, that x answers the question posed by r. The ***extent*** of $r \in I$ in A is taken to be

$$(r \checkmark\ ) = \{x \in A \mid r \checkmark x\}\,.$$

We also use the notation

$$(\ \checkmark x) = \{r \in I \mid r \checkmark x\}$$

for the set of indices filled by x. We write $\not\checkmark$ for the negation of the filling relation.

The ***answer*** map $\mathbf{Ans} : I \to \mathcal{P}\, I$ identifies the alternative potential increments offered by an index, *i.e.*, the different possible answers to the query that the index represents. We call those indices in the range $\bigcup(\mathbf{Ans}\, I)$ of this map ***answer indices***, or just ***answers*** if no confusion arises.

We now impose a few requirements having to do with the way in which indices may be viewed as questions whose answers are the answer indices. (IS1) essentially says that a question is described by its possible answers; *e.g.*, rather than asking "what is the color?" we should ask "is the color red, green or blue?" (or some other list of colors), with the alternative answers — the colors — represented by answer indices. (IS2) says that omitting some of the answers to a valid question results in another valid question, with fewer answers; *e.g.*, if one may ask "is the color red, green or blue?" then it should also be permissible to ask "is the color red or green?".

(IS1) Indices' extents are determined by their answers' extents: for every $r \in I$,

$$(r \checkmark\ ) = \bigcup \{(s \checkmark\ ) \mid s \in \mathbf{Ans}\, r\}\ .$$

(IS2) Indices are closed under answer subsets: for every index r_0 and non-empty subset S of $\mathbf{Ans}\, r_0$, there is an index r such that $\mathbf{Ans}\, r = S$.

(For the sake of definiteness, we rule out the case $S = \emptyset$, and we do not allow $\mathbf{Ans}\, r = \emptyset$. However, it is possible to carry out the development with the addition of an empty index — representing a query that always leads to divergence — so that this decision has no impact one way or the other.)

An important requirement is that different answers to a question really be different. Technically, we require that the answers of an index must present a choice between distinct alternatives:

(IS3) For every r, the extents of answers of r are pairwise disjoint:

$$\forall s_1, s_2 \in \mathbf{Ans}\, r\ .\ (s_1 \checkmark\ ) \cap (s_2 \checkmark\ ) \neq \emptyset \implies (s_1 \checkmark\ ) = (s_2 \checkmark\ ).$$

Note that (IS3) is stated so as to avoid a need to compare indices; we also make no attempt at this abstract level to impose succinctness on the indices and the answer indices.

An index structure $\mathcal{I}$ for a set A equipped with a partial order $\leq$ is said to have the various separation properties — T_0-separation, monotonicity, order separation and conformity — iff the family $\{(r \checkmark\ ) \mid r \in \mathcal{I}\}$ of its extents has the corresponding properties with respect to $(A, \leq)$.

We require that an index structure over a set have T_0-separation. Index structures over posets are required, additionally, to be monotone.

Take an ***indexed poset*** to be a poset B with an index structure $\mathcal{I}$ for the poset B.

An index structure over a dcpo (with consistent meets) is continuous (and stable) if all its extents are Scott-open (and stable-open). •

Together with monotonicity, (IS3) restricts the way in which extents may be built up from answer extents: each lobe of an extent $(r \checkmark\ )$ must be completely contained in $(s \checkmark\ )$ for some answer $s \in \mathbf{Ans}\, r$ with a uniquely determined extent. This may be seen as saying that an index over a poset offers a choice between *pairwise inconsistent* alternatives.

Beyond monotonicity, the continuity of an index structure expresses a Scott property for the extents of an index: if $r \checkmark (\bigvee X)$ for a directed X then $r \checkmark x$ for some $x \in X$. Moreover, in an algebraic dcpo, if $r \checkmark z$ then $r \checkmark y$ for some isolated approximation y to z.

We will restrict our attention to continuous index structures.

3.3.2 Example Two trivial examples of index structures that we may define at this point are the trivial continuous index structure, consisting of all Scott opens, and the trivial stable index structure, consisting of all stable opens. In both cases the extent of an open set p is taken to be p itself, and the answers are taken to be single lobes, with $\mathbf{Ans}\, p = \mathsf{lobes}\, p$. It can be checked that the two index structures satisfy the requirements — (IS3) is satisfied thanks to the disjointness of the lobes of a set — and the two index structures are continuous and stable, respectively.

The operational intuition behind these index structure views a set as a choice between its lobes, with the additional stipulation, in the stable case, that each of the choices is justified by some least amount of information. •

3.4 Indexable Functions

In order to extend the framework of indexed computation to provide a restricted notion of functions that obey the computational assumptions embodied in the choice of index structures, it is necessary to look only at functions that use valid computations over their input and output, and, in addition, we need to identify a criterion for the necessary relationship between the computations of a function's input and output. We will call the functions that are well behaved in this sense *indexable functions*. The computational assumptions leading to Scott continuity remain valid, so that we will start with Scott continuous functions, and identify additional requirements.

The intuitive idea here is that for every query presented regarding the function's output, there must be a (finite height) computation strategy over the input that acquires sufficient information to answer the query; this guarantees that the (computational agent computing the) function can reply within a finite amount of time to every query presented by the consumer of its output. A further refinement is obtained by the following apparent weakening: rather than requiring that there exists a complete (finite height) computation strategy over the input that acquires sufficient information, we will only ask that there be a query over the input that leads to sufficient information — this would be the first query of the computation strategy, or the first step of the computation over the input in order to answer the query posed over the output. If we require in addition that this first query make actual progress — increment the current approximation of the output — then we can glue together the different first steps with respect to different current approximations, to get a computation strategy as desired. It remains to be shown — via a proof of definability — that the computation strategy obtained will be of finite height.

This approach, incidentally, is not restricted to identifying (computational agents that compute) indexable (or sequential) functions; more general agents may be considered. In any case, the construction of a demand-driven dataflow network with a producer/consumer relationship local to each channel, and consisting of sequential computational agents, results in a coroutine-like computation [KM77]. As we shall show in more detail in section 4.2, this, and the framework of indexed computation, is a generalization of the computational paradigm present in Kahn and Plotkin's definition of sequential functions on CDSs [KP78] and employed in Berry and Curien's work on sequential algorithms between concrete data structures (CDSs) [BC82, Cur93]: The *cells* of CDSs serve as indices, and computation proceeds by a consumer querying the producer with a cell, to which the producer replies with a choice of a way to fill that cell, possibly after obtaining additional information about its inputs in a coroutine-like fashion.

The requirement that indices make an actual increment in the information content is captured technically by the auxiliary notion of *separation* by an index. Moreover, we only consider queries that help to decide among a non-empty and finite set of alternatives. Technically, finiteness is necessary here in order to obtain directed completeness of the indexable function spaces.

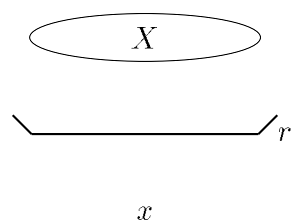

Figure 3.2: r separates x from X

3.4.1 Definition An index $r \in \mathcal{I}$ of an indexed poset $(B, \mathcal{I})$ is said to ***separate*** $x \in B$ from a subset $X \subseteq B$ iff $r \not\checkmark x$ and $r \checkmark X$ (*i.e.*, $r \checkmark y$ for every $y \in X$). We are interested in the case when x approximates all elements of X, and X is non-empty and finite. The notation $r \in \mathcal{I}(x, X)$ will be used to indicate that r separates x from X, and its use will imply that $X \subseteq \uparrow x$ and X is non-empty and finite. We may also state this relationship by saying that *r is an index at x for X*. If $f : A \to (B, \mathcal{I})$, we introduce the notation $r \in \mathcal{I}(f; x, X)$ as an abbreviation for $r \in \mathcal{I}(fx, fX)$, *i.e.*, r is an index at fx for fX, for $x \in A$ and $X \subseteq A$ such that $fX \subseteq \uparrow fx$ and fX is non-empty and finite.

A function $f : (B, \mathcal{I}) \to (B', \mathcal{I}')$ between indexed dcpos is ***indexable*** iff it is continuous and for every $x \in B$, non-empty finite $X \subseteq \uparrow x$ and index $r' \in \mathcal{I}'(f; x, X)$ there exists an index $r \in \mathcal{I}(x, X)$. Such an r is called *an index of f at (x, X) for r'*. •

Note that the notion of separation embodies within it a *proper increment* in the information content; *e.g.*, no index may separate x from itself. Also, an index with an empty extent would not separate x from X for any non-empty X, so that such an index would have no effect on indexability of functions. Hence, it makes no difference whether such an empty index is allowed or not, as we have already indicated.

As a result of the proper increment aspect of separation, the quantification over $r' \in \mathcal{I}(f; x, X)$ is vacuous if f is a constant function, because $fX = \{fx\}$. Thus constant functions are trivially indexable.

Clearly, the identity is indexable, since an index of the identity for $r' \in \mathcal{I}(\mathsf{id}; x, X)$ is given by r' itself. Indexable functions are closed under composition: if $r'' \in \mathcal{I}''(f' \circ f; x, X)$ then by indexability of f' there exists $r' \in \mathcal{I}'(f; x, X)$, and by indexability of f there exists $r \in \mathcal{I}(x, X)$.

A step function $[z \Rightarrow y]$ is indexable, because if $r' \in \mathcal{I}([z \Rightarrow y]; x, X)$ then $z \not\leq x$ and $z \leq x'$ for every $x' \in X$. Thus $\wedge X$ must properly dominate x, and by T_0-separation of the index structure there must exist an index separating $\wedge X$ (and X) from x.

3.4.1 Full Uncurrying

We remark that the definition of indexable functions relies on a sharp distinction between the input and output of a function, and is therefore quite sensitive to currying and uncurrying.

In particular, note that any continuous function $f : B \to B'$ between indexed dcpos is indexable if B is a trivially indexed dcpo, in the sense that, for every $x \in B$ and non-empty finite $X \leq \uparrow x$ there exists an index separating x and X. Any flat poset has this property if it is given an index structure where the extent of some index consists of all proper elements.

Thus, if we assume that the flat domain of booleans has an such an index, then the curried form of the parallel-or function $\mathsf{cpor} : \mathbf{Bool} \to (\mathbf{Bool} \to \mathbf{Bool})$ will be indexable, no matter what index structure is employed over its target. It is only by looking at the uncurried form $\mathsf{por} : \mathbf{Bool} \times \mathbf{Bool} \to \mathbf{Bool}$ that we can hope to identify index structures that formalize the non-indexability (and non-definability) of parallel-or.

The moral is that already when looking at first-order sequentiality, we need to assess the sequentiality of functions by looking at their fully uncurried forms. ("Full" uncurrying essentially refers to repeated uncurrying until the target is not a function space.) This is common to our approach, and to its progenitor, the Kahn-Plotkin definition of sequentiality [KP78]. Technically, this is done either by interpreting types and terms in their fully uncurried forms (in chapter 5), or by interpreting them in the usual way, but assessing indexability by examining the fully uncurried form of a function (in chapter 7, where this leads to considerable complications).

It is interesting to note that this issue and resulting technical difficulties are avoided by more intensional approaches, such as Berry and Curien's sequential algorithms model [Cur93] and (the intensional level of) the recent game-theoretic approaches [AJM94, HO94], where it seems that, so to speak, the division between input and output is not as pronounced, and as a result the property of sequentiality remains invariant under currying and uncurrying.

3.5 Alternative Definitions of Indexability

We present now two alternative formulations of the direct definition of indexability. The first is a generalized topological definition, with a generalized topology of *indexable sets*. The second relies on preservation of *critical sets*, analogous to the conditionally-multiplicative definition of stable functions by means of preservation of consistent sets. The critical sets for an index structure are closely related to Bucciarelli and Ehrhard's *coherence structures*, as we show in section 3.6.3.

3.5.1 Indexable Sets and Indexable Opens

A subset $p \subseteq B$ of an indexed poset B is ***indexable*** iff for every $x \notin p$ and every non-empty finite $X \subseteq p \cap \uparrow x$ there exists an index $r \in \mathcal{I}(x, X)$. In other words, if p separates x from a non-empty finite $X \subseteq \uparrow x$ then there exists an index that separates x from X. An ***indexable open*** of an indexed dcpo B is a Scott open of B that is indexable.

3.5.1 Proposition *The extent of an index is indexable, and it is indexable open if the index structure is continuous.*

Proof: Immediate from the definitions. ∎

We will assume in the remainder of this chapter that the index structures under consideration are continuous.

Order separation of index structures has the following corollary:

3.5.2 Proposition *$\uparrow y$ is indexable for every element y of an indexed dcpo with meets. If y is isolated then $\uparrow y$ is an indexable open.*

Proof: If $\uparrow y$ separates x from a non-empty and finite $X \subseteq \uparrow x$ then $y \leq \wedge X$, so that $\uparrow y$ separates x from $\wedge X$, and by order separation of index structures there exists an index that separates x from $\wedge X$, and hence x from X. ∎

Indexability is equivalent to continuity with respect to the generalized topology of indexable sets:

3.5.3 Proposition *A Scott continuous function $f : B \to B'$ between indexed dcpos is indexable iff it is continuous with respect to indexable sets, i.e., for every indexable set q of B', $f^{-1}q$ is an indexable set of B.*

Proof: Assume that f is indexable, and q is an indexable set of B'. Let $p = f^{-1}q$. To show indexability of p, choose some $x \notin p$ and let $X \subseteq p \cap \uparrow x$ be non-empty and finite. Now, $fx \notin q$ and $fX \subseteq q \cap \uparrow fx$, and because q is indexable there must be some index $r' \in \mathcal{I}(f; x, X)$. Since f is assumed indexable, there must exist some $r \in \mathcal{I}(x, X)$.

Conversely, assume that f is continuous with respect to indexable sets. Choose now some $x \in B$ and non-empty and finite $X \subseteq \uparrow x$ with some $r' \in \mathcal{I}(f; x, X)$. By 3.5.1 and continuity of the index structure, $(r' \checkmark\ )$ is an indexable open of B', so that, by assumption, $p = f^{-1}(r' \checkmark\ )$ is indexable set. But $x \notin p$ and $X \subseteq p \cap \uparrow x$, so that there must exist an index $r \in \mathcal{I}(x, X)$, as desired. ∎

A corollary allows us to combine Scott continuity and continuity with respect to indexable *sets* into a single notion of continuity with respect to indexable *opens*. But the corollary relies on proposition 2.2.2 which requires algebraicity of the domains, and is thus of slightly more restricted applicability.

3.5.4 Corollary *A function $f : B \to B'$ between indexed algebraic dcpos with meets is indexable iff it is continuous with respect to indexable opens.*

Proof: If f is indexable then it is Scott continuous, and, by the preceding proposition, it is also continuous with respect to indexed sets; hence it is continuous with respect to indexable Scott opens, which are precisely the indexable opens.

If f is continuous with respect to indexable opens then $\uparrow y$ and hence $f^{-1}(\uparrow y)$ are indexable opens for every isolated $y \in \mathbf{K}\, B'$, by 3.5.2, so that by proposition 2.2.2 f is Scott continuous. To show now that f is indexable we simply repeat the second part of the proof of the preceding proposition; this applies even if f is only continuous with respect to indexable opens, since we only relied on its generalized continuity to establish that $f'^{-1}(r' \checkmark\ )$ is indexable, and in fact $(r' \checkmark\ )$ is indexable open by 3.5.1 and continuity of the index structure. ∎

3.5.2 Critical Sets

The proof of proposition 3.5.2 suggests that tests of indexability of a set p need only be carried out at the meet of every non-empty finite $X \subseteq p$, thanks to the order separation property of index structures. The next proposition formalizes this, leading to a definition of *critical sets* and yet another characterization of indexability.

3.5.5 Proposition *In an indexed dcpo with meets, p is indexable iff for every non-empty finite $X \subseteq p$ with $\wedge X \notin p$ there exists an index at $\wedge X$ for X.*

Proof: ($\Longrightarrow$) is immediate from definition. For ($\Longleftarrow$), let $x \notin p$ and let $X \subseteq p \cap \uparrow x$ be non-empty and finite. If $x < \wedge X$ then, by order separation of indices, there exists an index at x for $\wedge X$, and hence for X. Otherwise, $x = \wedge X$ and by assumption there exists an index at x for X. ∎

This suggests the following definition: A non-empty finite subset $X \subseteq B$ of an indexed dcpo with meets is ***critical*** iff there is no index at $\wedge X$ for X.

3.5.6 Proposition *In an indexed dcpo B with meets, $p \subseteq B$ is indexable iff p is closed under critical meets.*

Proof: If p is indexable and a non-empty and finite subset $X \subseteq p$ is critical then $\wedge X \notin p$ would lead to a contradiction of indexability of p. Therefore p must be closed under critical meets.

If p is not indexable then by 3.5.5 there must exist some non-empty finite $X \subseteq p$ with $\wedge X \notin p$ that has no index at its meet; p is then not closed under critical meets. ∎

Some proofs may be simplified if critical sets are used, rather than alternative definitions. For instance, proposition 3.5.2 may now be proved as an immediate corollary of the closure of a principal filter $\uparrow y$ under *all* meets.

Just as critical sets completely characterize indexable sets, in the sense of the preceding proposition, so do indexable sets (or indexable opens) completely characterize critical sets:

3.5.7 Proposition *A non-empty finite set X is critical iff every indexable open that contains X also contains its meet $\wedge X$.*

Proof: If X is critical then certainly every indexable set that contains it also contains its meet. If X is not critical and has $r \in \mathcal{I}(\wedge X, X)$ then, by 3.5.1 and continuity of the index structure, $(r \checkmark)$ is an indexable open that contains X, but not its meet. ∎

Note that every non-empty finite set with a least element, and, in particular, every singleton, is critical.

We turn to a characterization of indexable functions through criticality.

3.5.8 Proposition *A continuous function $f : B \to B'$ between indexed dcpos with meets is indexable iff it preserves criticality and meets of critical sets, i.e., for every critical $X \subseteq B$, fX is a critical set of B', and $\wedge(fX) = f(\wedge X)$.*

Proof: Assume that f is indexable, and let X be a critical set of B. If $r' \in \mathcal{I}(f; \wedge X, X)$ then by indexability of f there exists $r \in \mathcal{I}(\wedge X, X)$, a contradiction. Now, by monotonicity of f, $f(\wedge X) \leq \wedge(fX)$. If this is a proper inequality then by order separation there exists an index r' at $f(\wedge X)$ for $\wedge(fX)$, and hence for fX, and the above contradiction may again be derived. Thus, to avoid a contradiction, it must be the case that $f(\wedge X) = \wedge(fX)$ and fX is critical.

Assume that f preserves criticality and critical meets, and let q be an indexed set of B' and $p = f^{-1}q$. If a non-empty finite $X \subseteq p$ is critical then $fX \subseteq q$ is critical, so that $f(\wedge X) = \wedge(fX) \in q$, and $\wedge X \in p$. p is therefore closed under critical meets, hence is indexable, and we may conclude by 3.5.3 that f is indexable. ∎

This characterization of indexable functions is reminiscent of the characterization of stable functions as continuous functions that preserve bounded meets (the conditionally-multiplicative formulation). We will use this similarity to relate indexability and stability. Note that there is no need to require stable functions to preserve boundedness of bounded sets, since boundedness is automatically preserved by any monotone function. This is not so for criticality (except under certain circumstances, see proposition 3.6.5 below), so that this characterization of indexability explicitly requires preservation of criticality, in addition to preservation of critical meets.

3.5.9 Example The critical sets with respect to the trivial continuous index structure of all Scott opens (defined in example 3.3.2) are exactly the non-empty finite sets that have a least element. Hence, the functions indexable with respect to this index structure are just the Scott-continuous functions, and the indexable opens are just the Scott opens. •

3.6 Stable Index Structures

Recall that a stable index structure for a dcpo with meets is an index structure whose indices have stable extents, *i.e.*, are closed under (non-empty finite) bounded meets. This means that no index can separate a non-empty finite bounded set X from its meet.

We claim that sequential computation employs stable index structures. This can only be an informal claim at this point. As evidence we offer the stability of index structures that will be defined in the sequel to capture (some notions and aspects of) sequential computation. Stability of the morphisms is, of course, a requirement for any purported definition of a fully abstract model for PCF (with respect to the stable order, see for instance [Ber79]).

Note that an index structure is stable iff its answers have stable extents, since extents are unions of disjoint answer extents, by (IS1) and (IS3), and disjoint union preserves stability.

3.6.1 Relating Indexability and Stability

Indexability implies stability when stable index structures are employed. This is simple to show using the critical set characterization of indexability, and exploiting its similarity to the conditionally-multiplicative characterization of stability.

3.6.1 Proposition *A non-empty finite bounded set is critical with respect to a stable index structure.*

Proof: If X is a non-empty finite bounded set and indices have stable extents, no index can separate $\wedge X$ from X. ∎

3.6.2 Corollary *For a stable index structure, an indexable set is stable, and an indexable open is stable open.*

An indexable function between stably indexed dcpos with meets is stable.

Proof: Since an indexable set (or indexable open) is closed under critical meets and bounded sets are critical, it is closed under bounded meets.

Similarly, an indexable function — stipulated to be continuous — preserves critical meets, and since the index structures are assumed stable, all non-empty finite bounded sets are critical, and their meets are preserved. ∎

In fact, the converse is also true: stable functions are indexable with respect to the trivial stable index structure of all stable opens (defined in example 3.3.2).

3.6.3 Proposition *The stable functions between dcpos with meets are precisely the functions indexable with respect to the trivial stable index structures on the dcpos.*

Proof: If f is indexable then it is stable, by proposition 3.6.2 and stability of the index structures.

If f is stable and $X \subseteq \uparrow x$ is non-empty and finite and there exists a stable open q that partitions fx and fX then $f^{-1}q$ is a stable open that partitions x and X. Therefore f is indexable with respect to the trivial stable index structures. ∎

3.6.2 Critical Sets of Stable Index Structures

The ***Egli-Milner*** order $\preceq$ over subsets of a poset B is defined for $X, Y \subseteq B$ by

$$X \preceq Y \qquad \text{iff} \qquad X \subseteq \downarrow Y \;\; \& \;\; Y \subseteq \uparrow X.$$

Of course, $Y \subseteq \uparrow X$ implies that $\wedge X \leq \wedge Y$ when the meets exist.

When stable index structures are employed, criticality has a "downwards motion" property with respect to the Egli-Milner order:

3.6.4 Proposition *If X and Y are non-empty finite subsets of a stably indexed dcpo with meets, Y is critical and $X \preceq Y$ then X is critical as well.*

Proof: Assume that r is an index of X at $\wedge X$. If $r \not\checkmark (\wedge Y)$ then r is an index of Y at $\wedge Y$, a contradiction to criticality of Y. If $r \checkmark (\wedge Y)$, let p be the lobe of $(r \checkmark\ )$ that contains $\wedge Y$. This lobe must then contain all points of X, since each one of them is consistent with $\wedge Y$. Since p is monotone and stable, it follows by proposition 2.2.9 that $\wedge X \in p$, and hence $r \checkmark (\wedge X)$. But this contradicts $r \in \mathcal{I}(\wedge X, X)$. Hence no such r exists, and X is critical. ∎

As a corollary, a function pointwise dominated by an indexable function preserves criticality of critical sets. This is a weak analogue to the preservation of boundedness by monotone functions; thus, to establish indexability of a function pointwise dominated by an indexable function it suffices to show that it preserves critical meets.

3.6.5 Proposition *If a function between stably indexed dcpos with meet is pointwise dominated by some indexable function then it preserves criticality of critical sets.*

Proof: Let $f : B \to B'$ be a function between stably indexed dcpos with meet, pointwise dominated by an indexable function h. Then for every critical set $X \subseteq B$, $fX \preceq hX$, and since hX is critical, by indexability of h, fX must also be critical. ∎

3.6.3 The Coherence Structure of Critical Sets

Critical sets make it possible to relate index structures and indexable functions to Bucciarelli and Ehrhard's coherence structures and strongly stable functions [BE91b].

3.6.6 Definition A ***coherence*** for a domain B is a subset C of $\mathcal{P}\, B$ which satisfies the following conditions:

(CH1) Each $X \in C$ is a finite set.

(CH2) For every $x \in B$, $\{x\} \in C$.

(CH3) If X and Y are finite subsets of B with $Y \in C$ and $X \preceq Y$ then $X \in C$ as well.

(CH4) If $X_1, \ldots, X_n$ are directed subsets of B and for every family $x_1 \in X_1, \ldots, x_n \in X_n$ we have $\{x_1, \ldots, x_n\} \in C$, then $\{\bigvee X_1, \ldots, \bigvee X_n\} \in C$.

A continuous function $f : (B, C) \to (B', C')$ between domains with coherence is ***strongly stable*** iff $fX \in C'$ and $\wedge(fX) = f(\wedge X)$ for every $X \in C$. •

Bucciarelli and Ehrhard phrased their definition with qualitative domains (a sub-class of dI-domains) as the underlying domains, but this definition may clearly be applied to larger classes of domains.

The critical sets of a stably indexed domain form a coherence structure. (CH1) is true by definition, and (CH2) is trivial. (CH3) has been shown by proposition 3.6.4 (and this is the place where stability of the index structure is required). (CH4) is easy to prove, using continuity of the index structure:

3.6.7 Proposition *If $X_1, \dots, X_n$ are directed subsets of an indexed domain B such that for every family $x_1 \in X_1, \dots, x_n \in X_n$ the set $\{x_1, \dots, x_n\} \in C$ is critical then $\{\bigvee X_1, \dots, \bigvee X_n\}$ is critical as well.*

Proof: Let $Y = \{\bigvee X_1, \dots, \bigvee X_n\}$. Assume that Y is not critical, and that there exists an index r that separates $\wedge Y$ from Y, *i.e.*, $r \checkmark Y$ and $r \not\checkmark \wedge Y$. For every $i \leq n$, $\bigvee X_i \in (r \checkmark\ )$ so that by continuity of $(r \checkmark\ )$ there exists some $z_i \in X_i$ with $z_i \in (r \checkmark\ )$. But now, fixing a choice of z_i for every $i \leq n$, $Z = \{z_1, \dots, z_n\}$ is itself not critical, since $r \checkmark Z$, and $r \not\checkmark \wedge Z$ (because $\wedge Z \leq \wedge Y$ and $r \not\checkmark \wedge Y$). But this contradicts the premises, and therefore Y must be critical. ∎

It immediately follows that the indexable functions coincide with the strongly stable functions between domains respectively equipped with stable index structures or the coherence formed by the critical sets for the index structures.

It is not the case, however, that every coherence structure is derived as the family of critical sets for some (stable) index structure. For instance, the coherence structure axioms do not force non-empty finite sets possessing a least element to be coherent (in the sense of belonging to the coherence structure), but such sets must always be critical. We attribute this difference to the operational content of the index structures, that causes the coherences they generate to be a special case of the more general framework of coherences.

While critical sets and indexable opens fully determine each other (propositions 3.5.6 and 3.5.7), it is not clear at this point if it is possible to axiomatize them — presumably by adding to the axioms of a coherence — so that they determine a (stable) index structure.

3.7 Indexable Function Spaces

We now consider the pointwise-ordered and stably-ordered indexable function spaces, corresponding to the set inclusion and lobe inclusion orderings on the indexable opens.

We obtain here closure results, along the general lines of showing that a certain class of domains equipped with stable index structures is closed under the construction of the indexable function space, when a certain order is used over the function space. That is, for any two domains in the class, and for every choice of stable index structures over the domains, the indexable function space, ordered as specified, is a domain in the class. We do not consider yet an index structure over the function space itself.

We write $B \rightarrow^{\mathrm{p}}_{\mathrm{x}} B'$ and $B \rightarrow^{\mathrm{s}}_{\mathrm{x}} B'$ for the pointwise- and stably-ordered indexable functions spaces, respectively, between the indexed domains B and B'.

3.7.1 Directed Completeness

Directed completeness under the pointwise order follows easily from the quantification over *finite* sets X as candidates for separation in the definitions of indexability.

3.7.1 Proposition *Indexable sets are closed under inclusion-directed unions.*

Proof: If P is an inclusion-directed family of indexable sets and a non-empty finite $X \subseteq \bigcup P$ is critical then, by finiteness of X and directedness of P, there exists $p \in P$ that contains X. Since p is indexable it contains $\wedge X$, and $\bigcup P$ is therefore closed under critical meets, and is indexable. ∎

3.7.2 Proposition *The pointwise lub of a pointwise directed set F of indexable functions between indexed dcpos with meet is indexable, and it is the lub of F in the pointwise ordered indexable function space.*

Proof: Let $f_0 = \bigvee^{\mathrm{p}} F$ be the pointwise lub of a pointwise directed set F of indexable functions between indexed dcpos with meet B and B'. f_0 is certainly continuous. For every indexable open q of B', $F^{-1}q = \{f^{-1}q \mid f \in F\}$ is, by 2.2.3, an inclusion-directed family of indexable opens, and by 3.7.1 $f_0^{-1}q = \bigcup F^{-1}q$ is indexable. ∎

This translates directly to directed completeness under the stable order.

3.7.3 Proposition *The pointwise lub of a pointwise directed set F of indexable functions between indexed dcpos is the lub of F in the stably ordered indexable function space.*

Proof: A set of functions directed under the stable ordering is also pointwise directed, so that its pointwise lub is already known to be indexable, and it remains to show that this is in fact a lub in the stable order (and likewise for the union of a lobe-inclusion-directed family of indexable sets).

If P is a lobe-inclusion-directed family of indexable sets then $\bigcup P$ is indexable, by 3.7.1. Moreover, by directedness, $\mathsf{lobes}(\bigcup P) = \bigcup(\mathsf{lobes}\, P)$, which implies both that $\bigcup P$ dominates P under lobe inclusion, and that it is the least indexable open that does so.

Using the terminology of proof 3.7.2 for the case when F is stably directed, we have that $f_0 = \bigvee^{\mathrm{p}} F$ is indexable, and for every indexable open q of B',

$$F^{-1}q \sqsubseteq \bigcup F^{-1}q = f_0^{-1}q,$$

so that f_0 stably dominates F. Moreover, since $f_0^{-1}q$ is the lub of $F^{-1}q$ under lobe-inclusion, if g is an upper bound of F in the stable ordering then it is certainly the case that $f_0^{-1}q \sqsubseteq g^{-1}q$, and it follows that f_0 is the lub of F in the stable order. ∎

3.7.2 The Stably Ordered Indexable Function Space

Indexable functions are down-closed under the stable order in the following sense:

3.7.4 Proposition *If p' is an indexable open of an indexed dcpo with meets then for every $p \sqsubseteq p'$, p is also an indexable open.*

Proof: If $p \sqsubseteq p'$ then it is easy to see that p is Scott open. For indexability, let $X \subseteq p \subseteq p'$ be critical, so that $\wedge X \in p'$. There must therefore be a single lobe of p' that contains $\wedge X$ and X, and this must therefore also be a lobe of p, so that $\wedge X \in p$. ∎

3.7.5 Proposition *For functions $f, f' : B \to B'$ between stably indexed dcpos with meets, if f' is indexable and $f \leq^{s} f'$ then f is indexable.*

Proof: For every indexable open q of B', $f^{-1}q \sqsubseteq f'^{-1}q$ by 2.2.7 and stability of the index structure, and since f' is indexable, $f'^{-1}q$ and hence also $f^{-1}q$ are indexable. ∎

Note that we need to require stability of the index structures. This already guarantees that the indexable functions form a restricted subset of the stable functions. Down closure has convenient corollaries for this restriction when the stable order is used on the function spaces.

3.7.6 Corollary *Indexable functions are down-closed in the stably-ordered stable function space between stably indexed dI-domains.*

Existing lubs and glbs in the stably-ordered indexable function space coincide with lubs and glbs taken in the stable function space. (In particular, existing lubs and consistent meets are taken pointwise.)

The isolated elements of the indexable function space are the isolated elements of the stable function space that are indexable.

The stably-ordered indexable function space between stably-indexed dI-domains is a dI-domain.

Proof: Down closure of the indexable functions in the stably ordered function space is immediate from 3.6.2 and 3.7.5.

The stably ordered stable function space is bounded complete (by 2.1.21). An indexable upper bound g of a set F in the indexable function space will stably dominate the lub f_0 of F in the stable function space, and by down closure f_0 itself will be indexable, and it will necessarily be the lub of F in the stably ordered indexable function space. Thus the indexable function space is itself bounded complete, and any lub that exists in the indexable function space coincides with the lub taken in the stable function space. Existence of all glbs in the indexable function space and their coincidence with glbs in the stable function space follows directly from down closure. All existing lubs and bounded meets are determined pointwise, since they are determined pointwise in the stable function space (by proposition 2.1.21).

If f is indexable and isolated in the stably-ordered stable function space then it is certainly isolated in the stably ordered indexable function space, by definition. If f

is isolated in the stably ordered indexable function space then, by algebraicity of the stable function space (proposition 2.1.29), f is the directed lub of the set F of isolated stable approximations to f. But F, being stably dominated by f, consists entirely of indexable functions, so that, by assumption, $f \in F$, and hence f is isolated in the stable function space.

Distributivity of the indexable function space, countability of the isolated elements, and property (I), all follow from the same properties for the stably ordered stable function space and the corollaries already shown. This completes the proof that the stably ordered indexable function space is a dI-domain. ∎

Note, in particular, that lubs and glbs taken pointwise in the stably ordered stable function space are also taken pointwise in the stably ordered indexable function space. We state explicitly the following:

3.7.7 Corollary *The pointwise join of stably consistent indexable functions between stably indexed dI-domains is indexable, and it is their join in the stably-ordered indexable function space.*

Proof: By 3.7.6 and 2.1.17. ∎

Another corollary to note here is that all finite projections drawn from stable SFPs are indexable, since they are stably dominated by the identity, an indexable function.

3.7.3 The Pointwise Order Revisited

Since application will generally fail to be stable when the pointwise order is employed on the function space, the pointwise order has not been considered for stable function spaces[1]. We show now that it is possible to obtain a limited closure result for pointwise-ordered indexable (and stable) function spaces. Of course, application will fail to be stable or indexable; but in chapter 5 we use this closure result to construct a restricted model for a sub-language of PCF, which, appropriately, does not contain application. For the closure result it is necessary to impose an additional *finite meet* property — that the isolated elements of the domains be closed under finite meets — so that the closure is shown for FM-domains, a class of domains intermediate between Scott domains and dI-domains. What we are able to show is that the pointwise ordered indexable function space between stably-indexed FM-domains is an FM-domain itself if the source and target domain are distributive, or if the target domain is flat (or more generally, a tree).

Stable opens and special functions may be seen as a special case of indexable opens and indexable functions when stable index structures are used, by example 3.3.2. Therefore, we use stable opens and stable functions for motivating the introduction of the finite meet property and giving some examples, and the development is carried out generally for indexable opens and indexable functions.

[1] Except perhaps for implicit corollaries that could be drawn from Berry's work on bidomains [Ber79], where both the stable and pointwise order were used over the function space, see chapter 6. One such corollary is that the pointwise-ordered stable function space between dI-domains is a Scott domain.

Meets in the Pointwise Order

We start by looking at the meet of indexable functions under the pointwise order. Glbs of indexable sets under inclusion are obtained by intersection.

3.7.8 Proposition *Indexable sets are closed under (arbitrary non-empty) intersection.*

Proof: Let P be a non-empty family of indexable sets, and $p_0 = \bigcap P$. To show indexability of p_0, choose $x \notin p_0$ and let $X \subseteq p_0 \cap \uparrow x$ be non-empty and finite. There must be some $p \in P$ such that $x \notin p$, and of course $X \subseteq p$. By indexability of p there exists an index at x for X. Indexability of p_0 follows. ∎

The Scott property is only preserved for finite intersection so that for indexable opens the above proposition needs to be restricted to non-empty finite intersections or, equivalently, binary intersections.

The pointwise meet of functions is known to preserve continuity and stability of functions; does it preserve indexability? In contrast to the analogous progression for directed completeness, this is not an immediate corollary of the closure of indexable opens under binary intersection, since in general the inclusion

$$(f_1 \wedge^{\mathrm{p}} f_2)^{-1} q \subseteq f_1^{-1} q \cap f_2^{-1} q$$

is not an equality. Under the assumption of stable indexing, however, this inclusion is strengthened to lobe-inclusion, which suffices to establish the desired closure.

3.7.9 Proposition *If $f_1, f_2 : B \to B'$ are indexable functions between stably indexed dcpos with meets then for every indexable open q of B',*

$$(f_1 \wedge^{\mathrm{p}} f_2)^{-1} q \sqsubseteq f_1^{-1} q \cap f_2^{-1} q$$

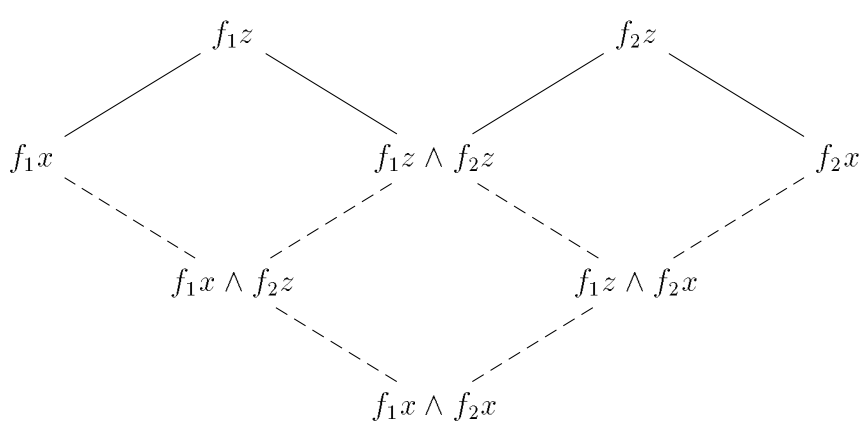

Figure 3.3: A zigzag in a stable open q

Proof: If $z \in (f_1 \wedge^{\mathrm{p}} f_2)^{-1}q$ and for some $x \leq z$, $x \in f_1^{-1}q \cap f_2^{-1}q$, then we have a zigzag

$$f_1x \leq f_1z \geq f_1z \wedge f_2z \leq f_2z \geq f_2x$$

in q, presented graphically in figure 3.3. Since B' is assumed to be stably indexed, q is stable, so that it must include all the points in the figure, and in particular $f_1x \wedge f_2x \in q$, implying that $x \in (f_1 \wedge^{\mathrm{p}} f_2)^{-1}q$. The desired lobe-inclusion follows. ∎

3.7.10 Proposition *The pointwise meet of indexable functions $f_1, f_2 : B \to B'$ between stably indexed dcpos is indexable, and it is their meet in the pointwise ordered indexable function space.*

Proof: Indexability of the pointwise meet is a corollary of 3.7.8, 3.7.9 and 3.7.4. That it is the meet in the pointwise order is immediate. ∎

Bounded Completeness in the Pointwise Order

We consider now the bounded completeness of pointwise ordered indexable function spaces, or, as an interim stepping stone, the collection of indexable opens, ordered by inclusion. The pointwise lub of indexable functions or union of indexable opens is known to be continuous, but will not be indexable in general.

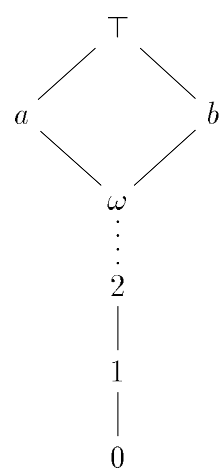

Figure 3.4: A Scott domain that is not FM

3.7.11 Example For a counter-example (in the stable case), consider the Scott domain in figure 3.4, where ω is the lub of an infinite ascending chain, and all other elements are isolated.

The stable opens $\uparrow a$ and $\uparrow b$ are bounded under inclusion. Their union $\uparrow a \cup \uparrow b$ is Scott open, but not stable since it is not closed under the consistent meet $\omega = a \wedge b$. On the other hand, $\uparrow \omega$ is closed under consistent meets, but is not Scott open, since it contains the lub ω of a chain, but no element of the chain. Thus, $\uparrow a$ and $\uparrow b$ are a counter-example to bounded completeness of the stable opens of a Scott domain ordered by inclusion, and, by extension, of the pointwise ordered stable function space between Scott domains (just replace $p \subseteq B$ by its characteristic function). •

In the example, $\uparrow \omega$ may be seen as an attempt to add to the union $\uparrow a \cup \uparrow b$ the bounded meet $\omega = a \wedge b$ that prevents the union from being stable. This does not work, because ω is the meet of two isolated elements, yet is not isolated itself. One may ask, then, if this may be fixed by removing the pathology — that is, requiring that isolated elements be closed under finite glbs. We show that this is in fact the case.

We say that an algebraic dcpo with meets has the ***finite meet property*** or ***property (FM)*** iff its isolated elements are closed under meets; that is, for any non-empty finite set X of isolated elements, $\wedge X$ is itself isolated. We designate as ***FM-domains*** the Scott domains that have property (FM). (The term *arithmetic* is used for this property in [GHK+80].)

FM-domains are a class of domains intermediate between Scott domains and dI-domains. The example above shows a Scott domain that is not an FM-domain. It is easy to see that dI-domains are FM-domains, because their isolated elements are down-closed; and the poset

$$\perp < \cdots < n < \cdots < 2 < 1 < 0$$

(the dual to the vertically ordered naturals, with an added bottom) is an example of an FM-domain that is not a dI-domain.

We may now employ iterated *critical meet completion* operations which add the necessary preservation or closure properties with respect to critical sets. A similar operation of completion by bounded meets was defined by Zhang [Zha91] for stable opens.

For a Scott open p of an indexed domain, define the ***critical meet completion*** of p to be $\mathsf{cmc}^* \, p = \bigcup \{\mathsf{cmc}^n \, p \mid n \in \mathbb{N}\}$, where

$$\mathsf{cmc}\, p = \uparrow \{\wedge X \mid X \subseteq p \ \ \& \ \ X \text{ critical } \}\, .$$

3.7.12 Proposition *For a Scott open p of a stably indexed FM-domain B, $\mathsf{cmc}\, p$ is Scott open, $p \subseteq \mathsf{cmc}\, p$, and $\mathsf{cmc}^* \, p$ is the least indexable open that contains p.*

Proof: Without loss of generality, $\mathsf{cmc}\, p$ is determined by its isolated elements: if X is a critical set contained in p then by the Scott property of p there must be some X_0 contained in p consisting of isolated approximations to elements of X, with $X_0 \preceq X$, and by downwards motion of criticality (proposition 3.6.4, requiring stability of the index structure) X_0 is critical, so that $\wedge X$ may be safely replaced in the construction of $\mathsf{cmc}\, p$ by its approximation $\wedge X_0$, which is isolated, by property FM. $\mathsf{cmc}\, p$ is therefore Scott open. Moreover, $p \subseteq \mathsf{cmc}\, p$, since every singleton is critical.

It is easy to check that $\mathsf{cmc}^* \, p$ is closed under critical meets, since every critical set it contains must have been introduced at some finite stage of the iteration, and its meet must have been introduced at the next stage. Finally, any indexable open that contains p must contain $\mathsf{cmc}^* \, p$, since every point added to p by the completion process must be present in any indexable open that contains p ∎

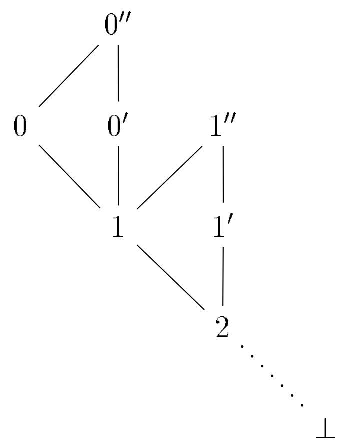

Figure 3.5: An FM-domain where completion may be infinite

3.7.13 Corollary *The indexable opens of a stably indexed FM-domain B, ordered by inclusion, are bounded complete (and a complete lattice with maximal element B), with* $\mathsf{cmc}^*(\bigcup P)$ *as the lub of a family P of indexable opens.*

3.7.14 Example An example of a critical completion that cannot be carried out in a finite number of steps is given by the Scott domain formed by the set $\{\bot\} \cup \{k, k', k'' \mid k \in \mathbb{N}\}$, ordered by $\bot < k{+}1 < k, k' < k''$ for every $k \in \mathbb{N}$. See figure 3.5. Take the trivial stable index structure over this set (see section 3.6.1; in this case the critical meet completion amounts to completion by bounded meets), and let p be the Scott open $\{0\} \cup \{k', k'' \mid k \in \mathbb{N}\}$. Then

$$\begin{aligned} \mathsf{cmc}^n\, p &= \{k; k \leq n\} \cup \{k', k'' \mid k \in \mathbb{N}\} \\ \mathsf{cmc}^*\, p &= \{k, k', k'' \mid k \in \mathbb{N}\}\,. \end{aligned}$$

•

Note, however, that the above example necessitates an infinite p. In fact, $\mathsf{cmc}^*\, p$ always stabilizes at some finite point if p is finite. To see this, note that $\mathsf{cmc}\, p \subseteq \mathsf{mc}\, p'$ for $p \subseteq p'$, where the meet closure mc is defined by

$$\mathsf{mc}\, p = \uparrow\{\wedge X \mid X \subseteq p \;\;\&\;\; X \text{ non-empty and finite }\}\,,$$

so that $\mathsf{cmc}^*\, p \subseteq \mathsf{mc}^*\, p$ for every p. But mc is idempotent, so $\mathsf{mc}^*\, p = \mathsf{mc}\, p$, and hence, for finite p, $\mathsf{cmc}^*\, p$ is contained in the finite set $\mathsf{mc}\, p$, and it must be the case that $\mathsf{cmc}^*\, p = \mathsf{cmc}^N\, p$ for some N.

Moving on to function spaces, we exploit the corollary to down closure of criticality: the pointwise lub of a family of indexable functions with an indexable upper bound preserves criticality, by 3.6.5, so that we only need to worry about preservation of critical meets.

For a function $f : B \to B'$ define the ***critical meet completion*** of f to be $\mathsf{cmc}^*\, f = \bigvee^{\mathrm{p}}\{\mathsf{cmc}^n\, f \mid n \in \mathbb{N}\}$, where

$$\mathsf{cmc}\, f = \lambda x \in B\,.\, \bigvee\{\wedge(fX) \mid \wedge X \leq x \;\;\&\;\; X \text{ critical}\}\,.$$

3.7.15 Proposition *If a continuous function $f : B \to B'$ between stably indexed FM-domains is pointwise dominated by an indexable function h then $\mathsf{cmc}\, f$ is well defined and continuous, $f \leq^{\mathrm{p}} \mathsf{cmc}\, f \leq^{\mathrm{p}} h$, and $\mathsf{cmc}^*\, f$ is well defined and is the least indexable function that dominates f.*

Proof: For every x and critical X with $\wedge X \leq x$,

$$\wedge(fX) \leq \wedge(hX) = h(\wedge X) \leq hx,$$

since $f \leq^{\mathrm{p}} h$ and h is indexable, so that $\mathsf{cmc}\, fx$ is well defined as the lub of a bounded set (with upper bound hx) in B'. Thus $\mathsf{cmc}\, f$ is well defined and $\mathsf{cmc}\, f \leq^{\mathrm{p}} h$. To see that $f \leq^{\mathrm{p}} \mathsf{cmc}\, f$ take $X = \{x\}$.

Monotonicity of $\mathsf{cmc}\, f$ is immediate. For continuity of $\mathsf{cmc}\, f$, we first need to show that the value of $\mathsf{cmc}\, fx$ may be determined by quantifying only over critical sets consisting of isolated elements, that is, that for every critical X,

$$(*) \qquad \wedge\,(fX) = \bigvee \{\wedge(fX') \mid X' \in \Xi'\}\,,$$

where Ξ' is the collection of non-empty finite sets of isolated elements that are Egli-Milner dominated by X (and thus are critical, by downwards motion of criticality). This equation holds by algebraicity of the domain and continuity of f and the meet. To establish continuity now, we show that f preserves the lub of a directed subset $Y \subseteq B$:

$$\begin{array}{rcl}
\mathsf{cmc}\, f(\bigvee Y) & = & \bigvee \{\wedge(fX) \mid \wedge X \leq \bigvee Y \ \ \&\ \ X \text{ critical}\} \\
& = & \bigvee \{\wedge(fX) \mid \wedge X \leq \bigvee Y \ \ \&\ \ X \text{ critical} \ \ \&\ \ X \subseteq \mathbf{K}\, B\} \\
& & \qquad \text{(by } (*)) \\
& = & \bigvee \{\wedge(fX) \mid \exists y \in Y \,.\, \wedge X \leq y \ \ \&\ \ X \text{ critical} \ \ \&\ \ X \subseteq \mathbf{K}\, B\} \\
& = & \bigvee \{\bigvee \{\wedge(fX) \mid \wedge X \leq y \ \ \&\ \ X \text{ critical} \ \ \&\ \ X \subseteq \mathbf{K}\, B\} \mid y \in Y\} \\
& = & \bigvee \{\bigvee \{\wedge(fX) \mid \wedge X \leq y \ \ \&\ \ X \text{ critical}\,\} \mid y \in Y\} \\
& & \qquad \text{(by } (*)) \\
& = & \bigvee \{\mathsf{cmc}\, fy \mid y \in Y\}\,.
\end{array}$$

Now turn to $\mathsf{cmc}^*\, f$. It is well defined and continuous, being the pointwise lub of an increasing chain of continuous functions. It preserves criticality of critical sets, since it is dominated by an indexable function h (proposition 3.6.5). To see that it preserves critical meets, let X_0 be a critical set, and set $x_0 = \wedge X_0$. Then:

$$\begin{array}{rcll}
\mathsf{cmc}^*\, fx_0 & = & \bigvee \{\mathsf{cmc}^n\, fx_0 \mid n \geq 0\} & \\
& = & \bigvee \{\wedge(\mathsf{cmc}^n\, fX) \mid n \geq 0 \wedge X \leq x_0 \ \ \&\ \ X \text{ critical}\} & \\
& \geq & \bigvee \{\wedge(\mathsf{cmc}^n\, fX_0) \mid n \geq 0\} & \\
& = & \wedge(\mathsf{cmc}^*\, fX_0). & \text{(meet continuous)}
\end{array}$$

The reverse inequality $\mathsf{cmc}^*\, fx_0 \leq \wedge(\mathsf{cmc}^*\, fX_0)$ follows from monotonicity. Thus $\mathsf{cmc}^*\, f$ preserves critical meets, and it is indexable.

Finally, $\mathsf{cmc}^*\, f \leq^{\mathrm{p}} h$, and since h is chosen arbitrarily among the indexable functions dominating f, $\mathsf{cmc}^*\, f$ is the least indexable function dominating f. ∎

3.7.16 Corollary *The pointwise ordered indexable function space between stably indexed FM-domains is bounded complete, and the lub of a bounded set F in this function space is* $\mathsf{cmc}^*(\bigvee^{\mathrm{p}} F)$.

The Pointwise Ordered Indexable Function Space

We have by now established the directed completeness and bounded completeness of the pointwise ordered indexable function space between stably indexed FM-domains. We complete this by showing that the function space is itself, in some circumstances, an FM-domain. We do this via a strongly algebraic characterization of FM-domains, analogous to those of Scott domains and dI-domains in section 2.1.5.

An ***FM-SFP*** for a dcpo B with meets is an SFP Ψ for B, such that for every isolated $x, y \in \mathbf{K}\, B$ there exists k_0 such that for every $i \geq k_0$, the projection ψ_i preserves the meet of x and y: $\psi_i(x \wedge y) = \psi_i x \wedge \psi_i y$. B is ***FM-strongly algebraic*** iff it has an FM-SFP.

3.7.17 Proposition *A pointed FM-strongly algebraic dcpo with meets is an FM-domain.*

Proof: Let B be a pointed FM-strongly algebraic dcpo with meets. An FM-SFP is an SFP, so that by propositions 2.1.23 and 2.1.22 B is a Scott domain, and its isolated elements are given by

$$\mathbf{K}\, B = \bigcup \{\psi_i B \mid i \in \mathbb{N}\}.$$

From this characterization of isolated elements and the additional property of FM-SFPs, if x and y are isolated then for i sufficiently large,

$$\psi_i(x \wedge y) = \psi_i x \wedge \psi_i y = x \wedge y,$$

and it follows that $x \wedge y$ is itself isolated. B thus has property (FM), and it is an FM-domain. ∎

3.7.18 Proposition *An FM-domain B has an FM-SFP Ψ given by*

$$\psi_i x = \bigvee \{z \leq x \mid \exists Z \subseteq \{x_1, \ldots, x_i\} \,.\, Z \neq \emptyset \;\;\&\;\; z = \wedge Z\},$$

where $x_1, x_2, \ldots$ is some enumeration of $\mathbf{K}\, B$.

If, moreover, B is distributive then for every i, ψ_i is indexable with respect to every stable index structure over B.

Proof: It is easy to check that Ψ is an SFP over B. The same arguments as in the proof of proposition 2.1.23 apply. Note that Ψ as defined here with respect to $x_1, x_2, \ldots$ is a sub-sequence of the SFP Ψ defined there with respect to the modified enumeration $z_1, z_2, \ldots$, where we insert after each x_i all the meets of non-empty finite subsets of previously enumerated elements. Property (FM) of the domain is relied upon to guarantee that the isolated elements are in fact closed under meets.

That Ψ is an FM-SFP follows from the fact that for every isolated elements x and y, for every sufficiently large i (once x and y have been enumerated),

$$\psi_i(x \wedge y) = x \wedge y = \psi_i x \wedge \psi_i y.$$

Assume now that B is distributive and it is equipped with some stable index structure, and choose some i. Let Z be the set $\{\wedge X \mid X \subseteq \{x_1, \ldots, x_i\} \ \& \ X \neq \emptyset\}$, so that $\psi_i x = \bigvee \{z \in Z \mid z \leq x\}$ for every x. To show that ψ_i is indexable, note first that, being dominated by the identity, ψ_i preserves criticality of critical sets, by proposition 3.6.5. To see that ψ_i preserves critical meets (actually, all meets), let $Y = \{y_1, \ldots, y_n\}$. Then by monotonicity $\psi_i(\wedge Y) \leq \wedge(\psi_i Y)$, and on the other hand,

$$\begin{array}{rcll} \wedge(\psi_i Y) & = & (\bigvee \{z \in Z \mid z \leq y_1\}) \wedge \cdots \wedge (\bigvee \{z \in Z \mid z \leq y_n\}) & \\ & = & \bigvee \{z_1 \wedge \cdots \wedge z_n \mid \forall i \leq n \,.\, z_i \in Z \ \& \ z_i \leq y_i\} & \text{(distributivity)} \\ & \leq & \bigvee \{z \in Z \mid z \leq \wedge Y\} & \\ & & \quad (Z \text{ closed under meets}). & \end{array}$$

Therefore ψ_i preserves all meets, and it is indexable. Note that we have used distributivity here. ∎

We may now employ this characterization to obtain property (FM) for function spaces. Recall that the exponential SFP $\Psi \to \Psi'$ over a function space $B \to B'$ is defined by

$$(\Psi \to \Psi')_i f = (\psi_i \to \psi_i') f = \psi_i' \circ f \circ \psi_i.$$

First, we show that FM-domains are closed under the continuous function space:

3.7.19 Proposition *If B and B' are FM-domains with FM-SFPs Ψ and Ψ', respectively, then $B \to_{\mathrm{c}}^{\mathrm{p}} B'$ is an FM-domain with the exponent FM-SFP $\Psi \to \Psi'$.*

Proof: All we need to show is that $\Psi \to \Psi'$ is an FM-SFP — it has been shown in proposition 2.1.25 to be an SFP. Let f and g be two isolated elements of the continuous function space. Their meet in $\mathcal{F}$ is known to be taken pointwise, so that, for every $x \in B$, and every i large enough,

$$\begin{array}{rcll} (\psi_i \to \psi_i')(f \wedge^{\mathrm{p}} g)x & = & \psi_i'((f \wedge^{\mathrm{p}} g)(\psi_i x)) & \\ & = & \psi_i'((f(\psi_i x)) \wedge (g(\psi_i x))) & \\ & = & \psi_i'(f(\psi_i x) \wedge \psi_i'(g(\psi_i x))) & (i \text{ large enough}) \\ & = & (\psi_i \to \psi_i') f x \wedge (\psi_i \to \psi_i') g x & \\ & = & ((\psi_i \to \psi_i') f \wedge^{\mathrm{p}} (\psi_i \to \psi_i') g) x & \end{array}$$

.

∎

For the indexable function space we need to worry about well definedness of the exponent, and indexability of the projections.

3.7.20 Proposition *If B and B' are stably-indexed FM-domains with FM-SFPs Ψ and Ψ', respectively, and the projections of Ψ and Ψ' are indexable, then $B \rightarrow^{\mathrm{p}}_{\mathrm{x}} B'$ is an FM-domain with the exponent FM-SFP $\Psi \rightarrow \Psi'$.*

Proof: Since the projections in Ψ and Ψ' are assumed indexable, the exponent projections $\psi_i \rightarrow \psi'_i$ are well defined — $(\psi_i \rightarrow \psi'_i)f$ is indexable whenever so is f, since indexable functions are closed under composition. The rest is as above — we rely on the fact that meets in the pointwise ordered indexable function space are taken pointwise. ■

3.7.21 Corollary *The pointwise ordered indexable function space between stably indexed distributive FM-domains is an FM-domain.*

Proof: Distributivity of the source and target domains guarantees the existence of an FM-SFP consisting of indexable functions. ■

Thus, a sufficient condition for the pointwise ordered indexable function space between stably-indexed FM-domains being itself an FM-domain is that the source and target domains be distributive. This corollary is of limited applicability, however, since in general the pointwise-ordered indexable function space fails to preserve distributivity, as shown by the next example. In the next section we proceed to determine another, more useful, sufficient condition.

3.7.22 Example Consider the stable function space $\mathbf{Bool}^3 \rightarrow \mathbf{2}$, ordered pointwise. Recall that for stable functions the critical sets are the non-empty finite sets. This example is also valid for functions that are sequential in Kahn and Plotkin's sense, or functions indexable with respect to the data index structures (to be defined in chapter 4).

Let $f = [(\mathsf{F}, \mathsf{F}, \mathsf{T}) \Rightarrow \top]$, $g_1 = [(\mathsf{T}, \bot, \bot) \Rightarrow \top]$, $g_2 = [(\bot, \mathsf{T}, \bot) \Rightarrow \top]$, and $g = g_1 \vee^{\mathrm{p}} g_2$. f, g_1 and g_2 are stable, but g, essentially a parallel-or of the first two components, is not stable, and taking completion by bounded meets, $\mathsf{cmc}^* g = [(\bot, \bot, \bot) \Rightarrow \top] = \mathsf{K}\top$. Therefore we have

$$\begin{array}{rcccl} f \wedge^{\mathrm{p}} g & = & \mathsf{K}\bot & = & \mathsf{cmc}^*(f \wedge^{\mathrm{p}} g) \\ f \wedge^{\mathrm{p}} \mathsf{cmc}^* g & = & f, & & \end{array}$$

so that

$$\mathsf{cmc}^*(f \wedge^{\mathrm{p}} g) < f \wedge^{\mathrm{p}} \mathsf{cmc}^* g.$$

This leads to a failure of distributivity of the pointwise-ordered stable function space:

$$\begin{array}{rcl} \mathsf{cmc}^*((f \wedge^{\mathrm{p}} g_1) \vee^{\mathrm{p}} (f \wedge^{\mathrm{p}} g_2)) & = & \mathsf{cmc}^*(f \wedge^{\mathrm{p}} (g_1 \vee^{\mathrm{p}} g_2)) \\ & < & f \wedge^{\mathrm{p}} \mathsf{cmc}^*(g_1 \vee^{\mathrm{p}} g_2), \end{array}$$

using distributivity of the continuous function space. Recall that the join of consistent indexable functions f_1 and f_2 in the pointwise-ordered indexable function space is given by $\mathsf{cmc}^*(f_1 \vee^{\mathrm{p}} f_2)$. •

Isolated Elements of the Pointwise Ordered Indexable Function Space

We may give a somewhat more explicit characterization of the isolated elements in the pointwise ordered indexable function space:

3.7.23 Proposition *If f is isolated in the continuous function space $B \to_{\mathrm{c}}^{\mathrm{p}} B'$ between two stably-indexed FM-domain and f is dominated by an indexable function then $\mathsf{cmc}^* f$ is isolated in the indexable function space $B \to_{\mathrm{x}}^{\mathrm{p}} B'$.*

If g is isolated in the indexable function space $B \to_{\mathrm{x}}^{\mathrm{p}} B'$ then $g = \mathsf{cmc}^* f$ for some isolated function in the continuous function space $B \to_{\mathrm{c}}^{\mathrm{p}} B'$.

The indexable function space $B \to_{\mathrm{x}}^{\mathrm{p}} B'$ is ω-algebraic.

Proof: Assume that f is isolated in the continuous function space and dominated by an indexable function, so that $\mathsf{cmc}^* f$ is well defined. Let G be a directed set of indexable functions with $\mathsf{cmc}^* f \leq^{\mathrm{p}} \bigvee^{\mathrm{p}} G$. But then $f \leq^{\mathrm{p}} \mathsf{cmc}^* f \leq^{\mathrm{p}} \bigvee^{\mathrm{p}} G$, and since f is isolated in the continuous function space there exists $g \in G$ with $f \leq^{\mathrm{p}} g$. Since g is indexable, $\mathsf{cmc}^* f \leq^{\mathrm{p}} g$. We may now conclude that $\mathsf{cmc}^* f$ is isolated in the indexable function space.

Conversely, assume that g is isolated in the indexable function space, and let F be the directed set of isolated approximations to g in the continuous function space, with $g = \bigvee^{\mathrm{p}} F$. Since g dominates F, we may apply cmc^* to F and by monotonicity of cmc^*, $\mathsf{cmc}^* F = \{\mathsf{cmc}^* f \mid f \in F\}$ is directed and $g = \bigvee^{\mathrm{p}} \mathsf{cmc}^* F$. Since g is isolated in the indexable function space, $g = \mathsf{cmc}^* f$ for some $f \in F$.

For algebraicity of the indexable function space, let g be indexable, and as above let F the directed set of isolated approximations to g in the continuous function space. Then $\mathsf{cmc}^* F$ is, as above, a directed set of isolated approximations to g in the indexable function space, with $g = \bigvee^{\mathrm{p}} \mathsf{cmc}^* F$. Moreover, by the first part of the proposition, every isolated approximation g_0 to g in the indexable function space is derived from some $f_0 \leq^{\mathrm{p}} g_0 \leq^{\mathrm{p}} g$ isolated in the continuous function space, by $g_0 = \mathsf{cmc}^* f_0$, so that $\mathsf{cmc}^* F$ in fact contains all isolated approximations to g in $B \to_{\mathrm{x}}^{\mathrm{p}} B'$.

As a further corollary, countability of $\mathbf{K}(B \to_{\mathrm{c}}^{\mathrm{p}} B')$ implies countability of $\mathbf{K}(B \to_{\mathrm{x}}^{\mathrm{p}} B')$. ∎

Thus, the pointwise-ordered indexable function space between FM-domains is a Scott domain, and we now seek to show that it has property FM. To do so, we would like to show that the cmc operation preserves isolatedness (in the continuous function space). This is essentially a simple combinatorial consideration: if f is isolated then it is completely described by a mapping of some finite set of isolated elements of the source into a finite set of isolated elements of the target, and $\mathsf{cmc}\, f$ is described by a derived finite mapping. To formalize this, however, we need to resort to a machinery of representation by step functions.

Define a ***representation*** of f to be a set ρ of step functions such that $f = \bigvee^{\mathrm{p}} \rho$, and a ***directed representation*** of f to be a representation of f such that for every z the set $\{y \mid \exists x \leq z \,.\, [x \Rightarrow y] \in \rho\}$ is directed. For every continuous function f, f has a directed representation, by taking all step functions approximating f. Technically, directedness of a representation will permit us to use the continuity of the meet operation.

The isolated elements of the continuous function space are those functions that have a finite representation. Note that an isolated function may dominate infinitely many step functions, but is the lub of a finite subset thereof.

For a set ρ of step functions, we use the abbreviations $\pi_1\,\rho = \{x \mid \exists y\,.\,[x \Rightarrow y] \in \rho\}$ and $\pi_2\,\rho = \{y \mid \exists x\,.\,[x \Rightarrow y] \in \rho\}$.

We define the operations of ***bounded join completion***, ***meet completion*** and ***critical meet completion*** over sets of step functions, denoted respectively by bjc, mc and cmc, as follows:

$$\begin{array}{rcl}
\mathsf{bjc}\,\rho & = & \{[\vee(\pi_1\,\rho') \Rightarrow \vee(\pi_2\,\rho')] \mid \rho' \subseteq \rho \;\;\&\;\; \rho' \text{ non-empty and finite } \;\&\; \Uparrow \pi_1\,\rho'\} \\
\mathsf{mc}\,\rho & = & \{[\wedge(\pi_1\,\rho') \Rightarrow \wedge(\pi_2\,\rho')] \mid \rho' \subseteq \rho \;\;\&\;\; \rho' \text{ non-empty and finite }\} \\
\mathsf{cmc}\,\rho & = & \{[\wedge(\pi_1\,\rho') \Rightarrow \wedge(\pi_2\,\rho')] \mid \rho' \subseteq \rho \;\;\&\;\; \rho' \text{ non-empty and finite } \;\&\; \pi_1\,\rho' \text{ critical}\}\,.
\end{array}$$

The meet completion is defined over the function space between FM-domains, and the critical meet completion is defined over the function space between stably indexed FM-domains. This guarantees that the step functions employed in the definitions are well defined (*i.e.*, are defined by isolated elements).

Note that bjc, mc and cmc preserve finiteness of representations.

Bounded join completion is a way of obtaining a directed representation for a function from a given representation for the same function.

3.7.24 Proposition *If f has a representation ρ then it has a directed representation $\mathsf{bjc}\,\rho$. If f has a finite representation then it has a directed finite representation.*

Proof: Let ρ be a finite representation of a function f. $\mathsf{bjc}\,\rho$ is well defined, since for every non-empty and finite $\rho' \subseteq \rho$, the projection $\pi_2\,\rho'$ is upper-bounded by $f(\vee\,\pi_1\,\rho')$, ρ' is finite, and isolated elements are closed under finite joins, so that we have defined valid step functions.

Let $\rho_0 = \mathsf{bjc}\,\rho$. For every z,

$$\begin{array}{rcl}
(\bigvee^{\mathrm{p}}\rho_0)z & = & \bigvee\{\vee\,\pi_2\,\rho' \mid \rho' \subseteq \rho \text{ non-empty and finite } \;\&\; \vee\,\pi_1\,\rho' \leq z\} \\
& = & \bigvee\{y \mid \exists \rho' \subseteq \rho \text{ non-empty and finite }\;.\,\forall x\,.\,[x \Rightarrow y] \in \rho' \implies x \leq z\} \\
& \leq & \bigvee\{y \mid [x \Rightarrow y] \in \rho \;\;\&\;\; x \leq z\} \\
& = & (\bigvee^{\mathrm{p}}\rho)z,
\end{array}$$

so that $f = \bigvee^{\mathrm{p}}\rho = \bigvee^{\mathrm{p}}\rho_0$, and ρ_0 is a representation of f.

Finally, if ρ is finite then so is ρ_0, and ρ_0 can be seen to be a directed representation: for every z,

$$\{y \mid \exists x \leq z\,.\,[x \Rightarrow y] \in \rho_0\} = \{\vee\,\pi_2(\rho') \mid \rho' \subseteq \rho \text{ non-empty and finite } \;\&\; \vee\,\pi_1\,\rho' \leq z\}\,,$$

is a directed set. ■

The critical meet completion on representations mirrors the critical meet completion on functions.

3.7.25 Proposition *For a continuous function $f : B \to B'$ between stably indexed FM-domains, if ρ is a directed representation of f then* $\mathsf{cmc}\,\rho$ *is a representation for* $\mathsf{cmc}\,f$.

Proof: The equality $\mathsf{cmc}\,f = \bigvee^{\mathrm{p}}(\mathsf{cmc}\,\rho)$ is shown as follows, for every $z \in B$:

$$\begin{array}{rcl}
\mathsf{cmc}\,f z & = & \bigvee\{\bigwedge(fX) \mid \bigwedge X \leq z \ \ \& \ \ X \text{ critical}\} \\
& = & \bigvee\{\bigwedge\{\bigvee\{y \mid \exists x' \leq x \,.\, [x' \Rightarrow y] \in \rho\} \mid x \in X\} \\
& & \qquad \mid \bigwedge X \leq z \ \ \& \ \ X \text{ critical}\} \\
& = & \bigvee\{\bigvee\{\bigwedge(\pi_2(\iota X)) \mid \iota : X \to \rho \ \ \& \ \ \forall x \in X \,.\, \pi_1(\iota x) \leq x\} \\
& & \qquad \mid \bigwedge X \leq z \ \ \& \ \ X \text{ critical}\} \\
& & \qquad \quad \text{(continuity of meet)} \\
& = & \bigvee\{\bigwedge(\pi_2\,\rho') \mid \rho' \subseteq \rho \text{ is finite } \ \& \ \ \pi_1\,\rho' \text{ is critical } \ \& \ \ \bigwedge(\pi_1\,\rho') \leq z\} \quad (**) \\
& = & \bigvee\{y \mid \exists x \leq z \,.\, [x \Rightarrow y] \in \mathsf{cmc}\,\rho\} \\
& = & (\bigvee^{\mathrm{p}}(\mathsf{cmc}\,\rho))z.
\end{array}$$

The continuity of meet may be used here, since ρ is a directed representation. For $(**)$ note that if X is critical and $\iota : X \to \rho$ is such that $\pi_1(\iota x) \leq x$ for every $x \in X$ then $\pi_1(\iota X) \preceq X$ and it is critical, by downwards motion of criticality. ∎

3.7.26 Proposition *If f is isolated in the continuous function space between two stably indexed FM-domains B and B' then* $\mathsf{cmc}\,f$ *and* $\mathsf{cmc}^n\,f$ *are isolated, for all n.*

Sufficient conditions for $\mathsf{cmc}^*\,f$ *to be isolated as well in the continuous function space are*

(1) Every non-empty finite bounded set of B' has a maximum element.

(2) B and B' are distributive.

Proof: If f is isolated then it has a finite directed representation ρ, so that by 3.7.25 $\mathsf{cmc}\,f$ has a finite representation $\mathsf{cmc}\,\rho$ and $\mathsf{cmc}\,f$ is isolated as well. $\mathsf{cmc}^n\,f$ is isolated as well, for every n, by a simple induction.

In order to show that $\mathsf{cmc}^*\,f$ is isolated in the continuous function space, we would like to show that iterated application of cmc to a finite representation results in a finite fixpoint, since all iterations are contained in some fixed finite superset; *i.e.*, there must exist some N with $\mathsf{cmc}^*\,f = \mathsf{cmc}^N\,f$. A significant twist here is that cmc is defined for directed representations.

If we assume condition (1) then every finite representation of a function from B to B' is directed. Therefore, if ρ is a finite representation for f then $\mathsf{cmc}^*\,\rho$ is a representation for $\mathsf{cmc}^*\,f$, and $\mathsf{cmc}^*\,\rho$ is finite, since it is contained in the finite set $\mathsf{mc}^*\,\rho = \mathsf{mc}\,\rho$ (meet completion is idempotent). Thus, cmc^* preserves isolatedness.

In general, however, the need to employ a directed representation prevents a simple iteration of the critical meet completion over a representation; rather, the iteration of cmc must be interleaved with iteration of the bounded join completion operation bjc that produces a directed representation (proposition 3.7.24). Thus, $(\mathsf{bjc} \circ \mathsf{cmc})^*\rho$ is a directed representation for $\mathsf{cmc}^*\,f$, and it is contained within the set $(\mathsf{bjc} \circ \mathsf{mc})^*\rho$. But under assumption (2) that the domains are distributive the compound operation

$\mathsf{bjc} \circ \mathsf{mc}$ can be shown idempotent, so that $(\mathsf{bjc} \circ \mathsf{mc})^* \rho = \mathsf{bjc}(\mathsf{mc}\, \rho)$ and it is a finite set for a finite ρ, and serves as the desired finite superset. It follows that $\mathsf{cmc}^* f$ has a finite representation, and it is therefore isolated in the continuous function space.

A more direct proof that (2) is a sufficient condition for $\mathsf{cmc}^* f$ to be isolated in the continuous function space, is obtained via strong algebraicity. With distributive source and target domains, the pointwise-ordered indexable function space has an FM-SFP that is (a restriction of) an SFP on the pointwise-ordered continuous function space, and it follows that the isolated elements of the former are the isolated elements of the latter which are indexable. ∎

We have thus identified sufficient conditions for cmc^* to preserve isolatedness in the continuous function space. This, in turn, is a sufficient condition for the indexable function space to have property (FM).

3.7.27 Proposition *For stably-indexed FM-domains B and B', if $\mathsf{cmc}^* f$ is isolated in the continuous function space $B \to_{\mathrm{c}} B'$ for every isolated continuous function $f : B \to B'$ dominated by some indexable function then*

(1) The isolated elements of the indexable function space $B \to_{\mathrm{x}}^{\mathrm{p}} B'$ are the isolated elements of the continuous function space $B \to_{\mathrm{c}}^{\mathrm{p}} B'$ that are indexable.

(2) The indexable function space has property FM.

Proof: The indexable functions isolated in the continuous function space are easily seen to be isolated in the indexable function space.

For the reverse direction, if g is isolated in the indexable function space then by proposition 3.7.23 $g = \mathsf{cmc}^* f$ for some f isolated in the continuous function space. But if we assume that cmc^* preserves isolatedness in the continuous function space, then g is itself isolated in the continuous function space.

For (2), note that every two indexable functions isolated in the indexable function space are by (1) also isolated in the continuous function space, and their meet in both function spaces coincides and is taken pointwise (propositions 3.7.10 and 2.1.8). By property (FM) for the continuous functions space (proposition 3.7.19) the pointwise meet is isolated in the continuous function space, and by (1) it is also isolated in the indexable function space. ∎

We have thus added to the limited closure result of corollary 3.7.21 another form of limited closure, which is adequate for our needs in chapter 5.

Let a ***tree domain*** be an FM-domain where every non-empty finite bounded set (equivalently, pair) has a maximum element. Tree domains include flat domains, as well as other "tree shaped" domains, such as the lazy natural numbers. Non-trivial products, *e.g.*, $\mathbf{2} \times \mathbf{2}$, are not tree domains.

3.7.28 Corollary *The pointwise-ordered indexable function space $B \to_{\mathrm{x}}^{\mathrm{p}} B'$ between stably-indexed FM-domains with B' a tree domain has property FM, and is an FM-domain.*

As we have previously pointed out, both of these corollaries may be specialized to the stable function space by using the trivial stable index structure.

Chapter 4

Data Sequentiality

As a first example for the application of the framework of indexed computation, we show that first-order PCF-sequential computation may be formalized in this framework. This is done by defining *data index structures*, which may be used to designate *data-sequential* functions — functions indexable with respect to the data index structures on their source and target. Data index structures are stable, and their defining characteristic is the use of the covering relation as the yardstick of atomic increment; a data index in effect determines a flat sub-poset. In section 4.2 we show that data sequentiality coincides with Kahn and Plotkin's definition of sequential functions when restricted to distributive concrete domains.

In section 4.3 we examine the definition of data indices over the product, and note that a data index over a product is induced by a data index in one of the components. The *product assumption* is the assumption that this property is true for index structures in general. Under the product assumption, it is possible to give a general definition of product of indexed domains, extending the poset product.

Section 4.4 is a first examination of sequentiality at higher order. Although the data-sequential function space has some desirable closure properties, we show that the product assumption leads to a failure of application to be indexable, without regard to the index structure or order employed on the function space. This leads us to conclude that the product assumption is adequate only at first order, and must be abandoned, or suitably refined, for a treatment of higher-order sequentiality.

4.1 Data Indices

First-order computation in PCF is carried out by terms whose free variables are of primitive types (or perhaps data types, if we allow product or similar constructors). Intuitively, the computation described by such terms proceeds sequentially and in discrete steps. At each step, the first-order constant being applied and the current state, *i.e.*, the current approximation to the value being computed, determine a component (or, variable) whose value must be known next. The actual computation by a single step, if the value for the chosen component is not already known in the current approximation, consists of looking up that component in the environment, resulting either in an increase of its approximation from $\perp$ to a proper value, or divergence if the environment fails to supply a proper value.

This is under the assumption that primitive types are interpreted by flat domains; *i.e.*, that they are described by some unstructured countable set of (proper) values, with an added bottom value to represent lack of information. Thus, an atomic increment corresponds to an increase in one of the components, and, over the space of tuples, this is a *covering* step.

Data indices formalize these intuitions, extrapolating from products of flat domains to Scott domains.

A ***data index*** of a Scott domain is a non-empty set $r \subseteq \mathbf{K}\, B$ of isolated elements such that

(DX1) *Atomicity*: There exists x covered by all points of r: $x \prec y$ for all $y \in r$.

> Note that if r has two or more elements then x is uniquely determined as $\wedge r$. If r is a singleton then x may fail to be uniquely determined. Nevertheless, we will write $\mathsf{anchor}\, r$ for such an x, and call it the ***anchor*** of r, taking care to deal explicitly with its not being uniquely determined for singletons.

(DX2) *Stability*: Elements of r are pairwise inconsistent: for every $y_1, y_2 \in r$, if $y_1 \Uparrow y_2$ then $y_1 = y_2$.

The extent of a data index is defined to be $\uparrow r$, that is, $r \checkmark x$ iff $x \in \uparrow r$.

The answers of the data index structure are the (singletons of) isolated elements, with $\mathbf{Ans}\, r = \{\{a\} \mid a \in r\}$.

The atomicity of a computation step represented by a data index is embodied by the covering relation — if x is the current approximation to the value being computed and $\mathsf{anchor}\, r \leq x$ for some data index r then r may improve the approximation if $r \not\checkmark x$ (*i.e.*, $x \notin \uparrow r$). The new approximation will then have the form $z = x \vee y$ for some $y \in r$. Over a dI-domain there exists, by proposition 2.1.34, a prime a corresponding to $\mathsf{anchor}\, r \prec y$, so that $z = x \vee a$ and $x \prec z$.

The existence of an anchor covered by all points of r is a *non-local* constraint on the *potential* increments by r; it corresponds to the intuition that a step of the computation takes place over a single "component". Thus, a data index over a domain may be seen as a way of "carving out" a flat sub-domain, that may serve (perhaps only locally) to identify a component of the domain, and represent a computation step that obtains information in that component.

Note that this is a valid definition of a continuous index structure. (IS3) follows from (DX2), and T_0-separation follows from algebraicity.

(DX2) guarantees that there is at most one, unique, increment that may be taken. While the increment corresponding to a covering step $x \prec z$ may be uniquely determined (proposition 2.1.34, applicable over dI-domains), (DX2) further guarantees that there is at most one such z that increases x towards the value being computed, and consequently, at most one of the potential increments of r is applicable. Technically, this is captured by stability of the data index structure.

We may refer to functions that are indexable with respect to data index structures on their source and target as ***data-sequential functions***, with corresponding notions of ***data-sequential opens*** and ***data-critical sets***.

4.1.1 Proposition *Data indices are stable, and data-sequential functions are stable.*

Proof: If $r \checkmark x_1$, $r \checkmark x_2$ and $x_1 \Uparrow x_2$ then, by (DX2), there must be a unique $y \in r$ such that $y \leq x_1, x_2$. Stability of data-sequential functions follows from stability of the data index structures and proposition 3.6.2. ∎

We remark that (DX1) is the essential addition introduced over and above the trivial stable index structure, as defined in example 3.3.2, and thus may be seen as the essential characteristic of data sequentiality.

4.1.1 Examples

Identity functions and constant functions are trivially indexable, hence also data indexable.

Data indices over a flat domain are the non-empty subsets of the proper elements, and the set of all proper elements is a maximal data index (with respect to inclusion of answers of an index, which corresponds here to simple inclusion between indices).

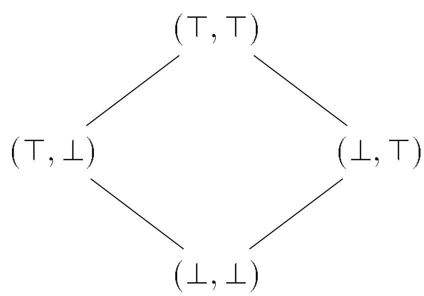

Figure 4.1: The domain $\mathbf{2} \times \mathbf{2}$

The data indices of $\mathbf{2} \times \mathbf{2}$ (see figure 4.1) are $\{(\top, \bot)\}$, $\{(\bot, \top)\}$, and $\{(\top, \top)\}$ (which may always be replaced for separation purposes by one of the first two indices). The set $\{(\bot, \bot)\}$ does not qualify as a data index since it has no anchor — and at any rate it cannot separate between points. The set $X_0 = \{(\top, \bot), (\bot, \top)\}$ conforms to the atomicity requirement (DX1), but fails to satisfy the stability requirement (DX2), so that it is not a data index. X_0 is a data-critical set, since no data index separates it from its meet $(\bot, \bot)$. The continuous function

$$\mathsf{poll} = [(\top, \bot) \Rightarrow \top] \vee^{\mathrm{p}} [(\bot, \top) \Rightarrow \top] : \mathbf{2} \times \mathbf{2} \to \mathbf{2}$$

(a parallel convergence tester) fails to be data sequential, because it fails to preserve the meet of X_0, or alternatively, because $\mathsf{poll}^{-1}\{\top\}$ is not a data-sequential open.

By similar arguments for $\mathbf{Bool} \times \mathbf{Bool}$, the set $\{(\mathsf{T}, \bot), (\bot, \mathsf{T})\}$ is data critical, and the parallel-or function

$$\mathsf{por} = [(\mathsf{T}, \bot) \Rightarrow \mathsf{T}] \vee^{\mathrm{p}} [(\bot, \mathsf{T}) \Rightarrow \mathsf{T}] \vee^{\mathrm{p}} [(\mathsf{F}, \mathsf{F}) \Rightarrow \mathsf{F}] : \mathbf{Bool} \times \mathbf{Bool} \to \mathbf{Bool}$$

is not data sequential. In general, since the critical meet that they fail to preserve is a bounded meet, both poll and por are not stable, so that they will fail to be indexable with respect to any stable index structure.

For examples of data-sequential functions of the same type as por, consider the doubly-strict-, left-strict-, and right-strict-or functions,

$$\begin{array}{rcl}
\mathsf{sor} & = & [(\mathsf{T},\mathsf{T}) \Rightarrow \mathsf{T}] \vee^{\mathrm{p}} [(\mathsf{T},\mathsf{F}) \Rightarrow \mathsf{T}] \vee^{\mathrm{p}} [(\mathsf{F},\mathsf{T}) \Rightarrow \mathsf{T}] \vee^{\mathrm{p}} [(\mathsf{F},\mathsf{F}) \Rightarrow \mathsf{F}] \\
\mathsf{lor} & = & [(\mathsf{T},\bot) \Rightarrow \mathsf{T}] \vee^{\mathrm{p}} [(\mathsf{F},\mathsf{T}) \Rightarrow \mathsf{T}] \vee^{\mathrm{p}} [(\mathsf{F},\mathsf{F}) \Rightarrow \mathsf{F}] \\
\mathsf{ror} & = & [(\bot,\mathsf{T}) \Rightarrow \mathsf{T}] \vee^{\mathrm{p}} [(\mathsf{T},\mathsf{F}) \Rightarrow \mathsf{T}] \vee^{\mathrm{p}} [(\mathsf{F},\mathsf{F}) \Rightarrow \mathsf{F}].
\end{array}$$

We have, for instance,

$$\begin{array}{rcl}
\{(\mathsf{T},\bot),(\mathsf{F},\bot)\} & \in & \mathcal{I}(\mathsf{lor};(\bot,\bot),\{\mathsf{T},\mathsf{F}\}\times\{\mathsf{T},\mathsf{F}\}) \\
\{(\bot,\mathsf{T}),(\bot,\mathsf{F})\} & \in & \mathcal{I}(\mathsf{lor};(\mathsf{F},\bot),\{\mathsf{F}\}\times\{\mathsf{T},\mathsf{F}\}).
\end{array}$$

These examples illustrate that a data index over products of flat domains such as $\mathbf{2}\times\mathbf{2}$ and $\mathbf{Bool}\times\mathbf{Bool}$ is essentially just a choice of a component of the product. This will be formalized in section 4.3.

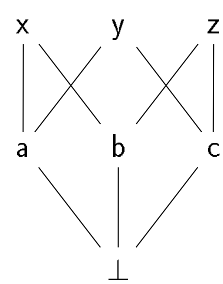

Figure 4.2: The domain $\mathbf{7}$

The data indices over the domain $\mathbf{7}$ (see figure 4.2) are all singletons of proper elements, as well as pairs of elements drawn from $\{\mathsf{x},\mathsf{y},\mathsf{z}\}$. Any subset of the primes $\mathbf{Pr}\,\mathbf{7} = \{\mathsf{a},\mathsf{b},\mathsf{c}\}$ will satisfy the atomicity requirement (DX1), but a set containing two or more primes will fail the stability requirement (DX2). The set $X_1 = \{\mathsf{x},\mathsf{y},\mathsf{z}\}$ is data critical; note that X_1 itself satisfies the stability requirement (DX2), and in particular is not a bounded set, but fails to satisfy atomicity (DX1).

A similar situation may be seen in $\mathbf{Bool}^3$ (see figure 4.3). The set

$$X_2 = \{(\mathsf{T},\mathsf{F},\bot),(\bot,\mathsf{T},\mathsf{F}),(\mathsf{F},\bot,\mathsf{T})\}$$

is data critical, since is has no data index at its meet, which amounts to saying that none of the three flat components of the product is a necessary "bottleneck" for the purpose of determining whether the value being computed is (approximated by) one of the values in X_2. If we now take *Gustave's function* (due to Berry [Ber76]),

$$\mathsf{gf} = [(\mathsf{T},\mathsf{F},\bot) \Rightarrow \mathsf{T}] \vee^{\mathrm{p}} [(\mathsf{T},\mathsf{F},\bot) \Rightarrow \mathsf{T}] \vee^{\mathrm{p}} [(\mathsf{T},\mathsf{F},\bot) \Rightarrow \mathsf{T}] : \mathbf{Bool}^3 \to \mathbf{Bool},$$

it fails to be data sequential, because it does not preserve the meet of X_2. But X_2 is not bounded, so that this poses no problem to the stability of gf; in fact, gf is the primary example of a function that is stable but not sequential (in any reasonable sense of sequential).

$$
\begin{tikzcd}
(\mathsf{T}, \mathsf{F}, \perp) & & & \\
& (\mathsf{T}, \perp, \perp) \arrow[ul] & & \\
(\perp, \mathsf{F}, \perp) \arrow[uu] & & (\perp, \perp, \mathsf{F}) \arrow[dr] & \\
& (\perp, \perp, \perp) \arrow[uu] \arrow[ul] \arrow[ur] \arrow[dl] \arrow[dr] \arrow[dd] & & (\perp, \mathsf{T}, \mathsf{F}) \\
(\perp, \perp, \mathsf{T}) \arrow[dd] & & (\perp, \mathsf{T}, \perp) \arrow[ur] & \\
& (\mathsf{F}, \perp, \perp) \arrow[dl] & & \\
(\mathsf{F}, \perp, \mathsf{T}) & & &
\end{tikzcd}
$$

Figure 4.3: A partial diagram of $\mathbf{Bool}^3$ (3d or aerial view, order indicated by arrows.)

Since gf is pointwise dominated by the constant true function, a data-sequential function, it must preserve criticality of critical sets (proposition 3.6.5). On the other hand, the function $\mathsf{gf} \vee^{\mathrm{p}} [(\mathsf{F}, \mathsf{F}, \mathsf{F}) \Rightarrow \mathsf{F}]$ fails to preserve the criticality of $X_2 \cup \{(\mathsf{F}, \mathsf{F}, \mathsf{F})\}$.

As an example of a data-sequential function of the same type as gf, we point to the conditional

$$\begin{aligned} \mathsf{if} \quad &: \quad \mathbf{Bool}^3 \to \mathbf{Bool} \\ &= \quad [(\mathsf{T}, \mathsf{T}, \bot) \Rightarrow \mathsf{T}] \vee^{\mathrm{p}} \\ & [(\mathsf{T}, \mathsf{F}, \bot) \Rightarrow \mathsf{F}] \vee^{\mathrm{p}} \\ & [(\mathsf{F}, \bot, \mathsf{T}) \Rightarrow \mathsf{T}] \vee^{\mathrm{p}} \\ & [(\mathsf{F}, \bot, \mathsf{F}) \Rightarrow \mathsf{F}]. \end{aligned}$$

4.1.2 An Enabling Relation?

We now address the need to impose an enabling constraint on the computation of indices. This discussion applies to the general definition of index structures, but it can be presented better at this point, using the data index structure as a concrete example.

Note that when describing the intended operational meaning of data indices, we specified that the computation of a data index r allows for an atomic increment of the current approximation x to a new approximation $x \vee y$ for some $y \in r$. This increment is atomic in the sense that $x \prec x \vee y$, under the assumption that $x \geq \mathsf{anchor}\, r$. But if $x \not\geq \mathsf{anchor}\, r$, one would have no guarantee that, in fact, $x \prec x \vee y$. Should this be allowed as a valid atomic increment?

One possible approach is to rule this out; one could add to index structures yet another component, an ***enabling*** relation between indices and points, such that the question represented by an index r may only be posed at points where it is enabled. One would then say that a data index r is enabled at x iff $\mathsf{anchor}\, r \leq x$ (ignore for this discussion the cases where $\mathsf{anchor}\, r$ is not uniquely determined). The same is effectively done in the Kahn-Plotkin definition of sequential functions between concrete domains, presented below.

We choose however not to use an enabling relation (equivalently, to use a trivial enabling relation: every index is enabled at every point), since this reduces considerably the complexity of the definitions and development. The risk we face under this omission is that we will only be able to model a smaller set of behaviors. For instance, with an enabling relation, we can impose a restriction on computation over a disjoint sum, to the effect that, before querying about the value, we must already know in which component of the sum the value lies. Without an enabling relation, we would not be able to impose such a restriction, thus making it possible, in effect, for the increment of the current approximation to be non-atomic, or more than a covering step for data indices. For our present purposes, however, the omission of an enabling seems not to pose any problem. On the contrary, we are about to show an equivalence between Kahn and Plotkin's notion of sequentiality that uses enabling, and data sequentiality, which does not. This is probably attributable to the existence of an explicitly determined $\mathsf{anchor}\, r$ for a data-sequential index r, which may be said to encode the enabling information for r.

4.2 Relationship to Kahn-Plotkin Sequentiality

The data-sequential functions are a conservative extension of Kahn and Plotkin's definition of sequential functions on distributive concrete domains. Although Kahn and Plotkin's definition applies to concrete domains, a class incomparable with dI-domains, most of the work has been carried out over distributive concrete domains, a proper subclass of dI-domains.

This is significant, since the Kahn-Plotkin sequential functions capture first-order computation in PCF, in the sense that the first-order fragment of PCF may be given a fully abstract model using distributive concrete domains and Kahn-Plotkin sequential functions. Therefore, a conservative extension will capture first-order computation in PCF in the same sense.

The main technical differences between this previous work and the formulation of data sequentiality are:

- We do not use an enabling relation (or, use a trivial one).
- Our indices are closed under answer subsets, and index structures contain their prime building blocks, *i.e.*, answer indices. This is a technically convenient decision with respect to representation and formalization of indices, and it seems to be necessary in order to extend the definition beyond CDSs.

  On the downside, the cells of a distributive concrete domain (or, associated concrete data structure) will not form an index structure, since they fail to satisfy these requirements, but make instead an alternative decision, to separate the answers (events) from the indices, and allow only maximal (or canonical) indices.

  Nevertheless, there is a simple relationship between the cells of a CDS and the data index structure: cells are essentially data indices that are maximal under inclusion. As we show, every data index is contained in some maximal data index when working over a distributive concrete domain, making it reasonable to employ maximal indices (represented as cells) in that setting, but this property fails to apply in general.

4.2.1 Kahn-Plotkin Sequentiality

Concrete data structures (CDSs) and concrete domains and sequential functions over them were defined by Kahn and Plotkin [KP78]. This notion of sequentiality, which we will refer to as KP-sequentiality, has been studied by Berry and Curien [BC82, Cur93], and we largely follow their formulation. We give here definitions only to the extent necessary to state and prove the relationship between Kahn-Plotkin sequentiality and data sequentiality; a complete exposition is provided by the above references, in particular [Cur93].

A ***concrete domain*** is a pointed ω-algebraic dcpo satisfying the following axioms, for all isolated x, y, z:

(I) Every isolated element dominates finitely many elements.

(C) If $x \prec y$, $x \prec z$ and $y \Uparrow z$ then $y \vee z$ exists, and $y \prec y \vee z$, $z \prec y \vee z$.

(R) If $(x, y) \succ\!\prec (x, z)$ then $y = z$.

(Where the equivalence $\succ\!\prec$ is the reflexive, transitive and symmetric closure of the covering relation on pairs of isolated elements, defined by $(x, y) \prec (x', y')$ iff $x \prec y$, $x \prec x'$, $x' \prec y'$, $y \prec y'$, and $y \neq x'$.)

(Q) If $x \prec y$, $x \leq z$ and $y \not\uparrow z$ then there exists a unique z' such that $x \prec z' \leq z$ and $y \not\uparrow z'$.

A ***concrete data structure***, or ***CDS***, $(\mathrm{C}, \mathrm{V}, \mathrm{E}, \vdash)$ consists of a countable set C of ***cells***, a countable set V of ***values***, a set $\mathrm{E} \subseteq \mathrm{C} \times \mathrm{V}$ of ***events***, and an ***enabling*** relation $\vdash$ between finite sets of events and cells.

For a set x of events of a CDS, the cells ***filled*** in x are those cells c in the first projection of x, *i.e.*, such that there exists a value v with $(c, v) \in x$. The cells ***enabled*** by x are cells c such that there exists a finite subset $y \subseteq x$ with $y \vdash c$. Such a subset y is called an ***enabling*** of c (in x). The cells ***accessible*** from x are cells that are enabled by x, but are not filled in x.

A CDS is ***well founded*** iff the relation of precedence between its cells is well founded, where we say that a cell c ***precedes*** a cell c' iff c is filled in some set y of events with $y \vdash c'$.

A ***state*** of a well founded CDS is a set x of events that fills every cell with at most one value (this is referred to as *consistency* or *functionality* of x) and such that every cell filled in x is enabled in x (this is referred to as *safety* of x).

A well founded CDS is ***stable*** iff for every state x and cell c enabled in x, c has a unique enabling in x. A CDS is a ***deterministic*** CDS, or DCDS for short, iff it is well founded and stable.

We can now tie CDSs and concrete domains via a representation theorem, due to Kahn and Plotkin [KP78], and generalized and simplified by Winskel's work on *event structures* and *event domains* [Win80, Win87].

4.2.1 Theorem (Representation) *The set of states of a CDS, ordered by inclusion, is a concrete domain. For every concrete domain B, there exists a CDS whose set of states, ordered by inclusion, is isomorphic to B.*

The set of states of a DCDS, ordered by inclusion, is a distributive concrete domain. For every distributive concrete domain B, there exists a DCDS whose set of states, ordered by inclusion, is isomorphic to B.

This allows us to blur the distinction between (distributive) concrete domains and (deterministic) concrete data structures, so that we may regard each element of a concrete domain as a state of a concrete data structure.

From the development cited above one also obtains:

4.2.2 Proposition *A distributive concrete domain is a dI-domain, and it is coherent (i.e., pairwise-consistent subsets have lubs). Its isolated elements are finite states. For states x, y, $x \prec y$ iff $y = x \cup \{(c, v)\}$ for some event (c, v).*

Thus, covering corresponds to filling a single (accessible) cell.

A function $f : B \to B'$ between concrete domains (viewed as sets of states of a CDS, ordered by inclusion) is KP-sequential iff it is continuous and, for every $x \in B$ and cell c' accessible from fx, one of the following holds:

(1) Either c' is not filled in fy for all superstates $y \supseteq x$;

(2) Or there exists some cell c accessible from x that is filled in every superstate $y \supseteq x$ with c' filled in fy.

In case (2), a cell $c \in \mathrm{A}(x)$ as described there is called an ***index of sequentiality*** of f at x for c'.

It is easy to transform the definition to an equivalent one that only needs case (2), by quantifying over c' accessible from fx such that c' is filled in fy for some superstate $y \supseteq x$. This form of the definition is directly related to our definition of indexable functions.

The term *index of sequentiality* motivated our choice of the term *indices*, and thence *indexed computation*: our indices are intended to be objects that may serve as indices of sequentiality in a sense generalizing KP-sequentiality. Similarly, our use of a *filling* relation between indices and points originates in the filling relation between cells and states (and likewise for the *enabling* relation that we have decided to omit).

4.2.2 Data Sequentiality and KP-sequentiality

There is a close correspondence between the cells of DCDSs and data indices, and between data-sequential opens and finite sets of cells. Moreover, data indices in distributive concrete domains are very well behaved, in that the quantification over non-empty finite sets in the basic definitions of indexability may be extended in this setting to all non-empty sets.

4.2.3 Proposition *In a distributive concrete domain B,*

(1) For every data index r and every possible choice of $\mathsf{anchor}\, r$ *there exists a unique cell c accessible from* $\mathsf{anchor}\, r$ *and filled in all elements of r.*

For every cell c accessible from x, there is a unique data index r consisting of prime elements of B, with $\mathsf{anchor}\, r \leq x$*, that is filled in all states above x that fill c. This r is maximal with respect to inclusion among data indices.*

(2) A Scott open set p is indexable iff for every $x \notin p$ with $p \cap \uparrow x$ non-empty there exists a data index $r \in \mathcal{I}(x, p \cap \uparrow x)$.

(3) For every data-sequential open p the set G of cells that are filled in all elements of p is finite and, if $p \neq \emptyset$ and $p \neq B$, G is non-empty.

For every finite set of cells G, the set $\{x \mid \forall c \in G \,.\, \exists v \,.\, (c, v) \in x\}$ of states that fill all cells in C is data-sequential open. This is also the case if we restrict the values filling each of the cells to a subset of the permissible values: $\{x \mid \forall c \in G \,.\, \exists v \in V_c \,.\, (c, v) \in x\}$, for a choice of subsets $V_c \subseteq \mathrm{V}$ for every $c \in C$.

Proof:

(1) If r is a data index then any element x of r covers $\mathsf{anchor}\, r$ by filling exactly one cell, which must already be accessible in $\mathsf{anchor}\, r$. Since elements of r are pairwise inconsistent, this cell must be common to all elements of r, and is uniquely determined.

If c is accessible from x then, by stability of the DCDS, there is a minimal point $y \leq x$ from which c is accessible. Let r be the set of states obtained from y by filling c, so that $y = \mathsf{anchor}\, r$. The elements of r have y as a predecessor by minimality of y, and thus are primes; they are pairwise inconsistent by functionality of states. r is therefore a data index consisting of prime elements. It is maximal and uniquely determined since it must consist of all states that cover y by filling c, and may not contain additional states. (For non-emptiness of r we assume without loss of generality that every cell of the DCDS may be filled in some way.)

(2) If there exists for every $x \notin p$ an index separating x from $p \cap \uparrow x$ then this same index may be used to separate x from any subset of $p \cap \uparrow x$, and indexability of p follows.

For the reverse direction, let p be an indexable open, and $x \notin p$. Assume at first that x is a finite state (*i.e.*, a finite set and an isolated element of the domain). Since $p \cap \uparrow x$ is Scott open, $p \cap \uparrow x = \uparrow X_0$ for some set X_0 of finite states.

For a set $X \subseteq X_0$ of states, let GX be the set of cells filled in all states of X and accessible at x. If we take non-empty finite subsets $X \subseteq X_0$, then by indexability of p there exists a data index $r \in \mathcal{I}(x, X)$; by (1) there exists a cell c corresponding to r which is filled in all states in $\uparrow r$; this implies that GX is non-empty. Moreover, GX is finite, since X_0 and X consist of finite elements that have only finitely many filled cells. Further, it is easy to see that

$$G(X_1 \cup X_2) = GX_1 \cap GX_2$$

for every pair $X_1, X_2 \subseteq X_0$. Therefore the family Γ of GX for all non-empty finite subsets X of X_0 is a family of non-empty finite sets, it is directed under reverse set inclusion, and the limit $GX_0 = \bigcap \Gamma$ is non-empty, that is, there exists a cell c filled in all elements of X_0 (and hence filled in all elements of $p \cap \uparrow x$) and accessible from x, and by (1), there exists a data index that separates x from $p \cap \uparrow x$.

For x not necessarily finite, $p \cap \uparrow x = \uparrow X_0$ for some set X_0, except that now we cannot guarantee that X_0 consists entirely of finite states. If $x < \bigwedge X_0$ then it is easy to find a cell c accessible from x and filled in $\bigwedge X_0$ and all elements of X_0, from which a data index may be derived by (1). Assume therefore that $x = \bigwedge X_0$. Since p is Scott open, there exists a set $X_0' \subseteq p$ of finite states such that $X_0' \preceq X_0$, which implies that all states in X_0' are consistent with x. Let $x' = \bigwedge X_0'$, and clearly $x' \leq x$, so that $x' \notin p$. x' is finite, so that the previous development applies, and there exists a cell c accessible from x' and filled in all elements of $p \cap \uparrow x'$, and in particular in all elements of X_0'. If c is filled in x we derive a contradiction, since it must then be filled with the same value v in all states of X_0, and thus in all states of X_0', so that $x' \cup \{(c, v)\}$ is a lower bound of X_0', contradicting $x' = \bigwedge X_0'$. Therefore c is accessible at x, and by (1) there exists a data index as desired.

(3) Let p be data-sequential open, and G the set of cells filled in all states of p. G is finite since p is defined by its isolated elements, each of which has finitely many filled cells. If p is non-empty and $\bot \notin p$ then there is a data index $r \in \mathcal{I}(\bot, p)$, by (2), which implies that G is non-empty.

Conversely, if G is a finite set of cells then the set p of states that fill all cells in G is Scott open (since G is finite and cells are finitely enabled), and if $x \notin p$ then $p \cap \uparrow x$, if non-empty, has an index separating it from x, obtained from some $c \in G$ that is not filled in x. The same reasoning applies even if only a subset of the possible values are used to fill each $c \in G$.

(Note that G does not determine a unique data-sequential open, although it does determine one that is maximal (with respect to inclusion). Compare, for instance, $\{(\mathsf{T}, \bot), (\mathsf{F}, \bot)\}$ and $\{(\mathsf{T}, \mathsf{T}), (\mathsf{F}, \bot)\}$, both of which have G corresponding to the cell for the left component.) ∎

As a corollary, we obtain the coincidence of KP-sequentiality and data sequentiality over distributive concrete domains.

4.2.4 Corollary *A function $f : B \to B'$ between distributive concrete domains is data-sequential iff it is KP-sequential.*

Proof: Continuity of f is guaranteed by either of the assumptions.

Assume that f is data-sequential, x is a state of B and c' is accessible from fx. By part (3) of the proposition, the set q of states that fill c' is data-sequential open, so that $p = f^{-1}q$ is data-sequential open. Now, $x \notin p$, and if $p \cap \uparrow x$ is non-empty (*i.e.*, there exists some state $y \geq x$ with c' filled in fy) then by part (2) there exists a data index $r \in \mathcal{I}(x, p \cap \uparrow x)$. Therefore, by part (1), there exists some cell accessible from x, and filled in any superstate y of x with c' filled in fy. It follows that f is KP-sequential.

For the converse, assume that f is KP-sequential, and that there exists a data index $r' \in \mathcal{I}(f; x, X)$ for a non-empty finite $X \subseteq \uparrow x$. By part (1) there exists some cell c' accessible from fx and filled in all elements of fX. Since there exists a superset $y \geq x$ such that c' is filled in fy (take any $y \in X$), KP-sequentiality provides for the existence of a cell c accessible from x and filled in all $y \in X$, which, again by part (1), implies the existence of an index separating x and X. Data sequentiality of f follows. ∎

4.2.3 Primal Data Sequentiality

We can obtain a more economical index structure with essentially the same properties by restricting to primal data indices, which are the indices consisting of prime elements. Primal data indices may be turned into an index structure by using the same extents and answers as for data indices (except that the answer indices are now singletons of prime elements). Call this the ***primal data*** index structure. To obtain T_0-separation for this index structure, however, we must restrict ourselves to prime-algebraic domains; all other index properties pose no problem. By definition of prime elements, $\mathsf{anchor}\, r$ must be the predecessor of every element of a primal data index, and it is uniquely determined even for singletons; (DX1) may be rephrased for primal data indices by requiring that all elements of r have a common predecessor, which is then labelled $\mathsf{anchor}\, r$. We obtain definitions of primal data-sequential functions as the functions indexable with respect to these index structures.

Note that the proofs for 4.2.3 and 4.2.4 apply equally well to the primal data index structure, since the index constructed for a cell is a primal data index. Thus they may be taken as showing that data and primal data sequentiality coincide with each other, and with KP-sequentiality, over distributive concrete domains.

There are more data indices than primal data indices, because for every primal data index r and isolated $x \notin \uparrow r$,

$$r \vee x = \{a \vee x \mid a \in r \;\;\&\;\; a \Uparrow x\},$$

if not-empty, is a data index. The converse is true over distributive concrete domains: As a corollary of part (1) of 4.2.3, we have that for every data index r there exists an isolated x and primal data index r_0 with $r = r_0 \vee x$. This fails, however, in general. Consider for instance the domain **7** (diagram 4.2 on page 84): the set $\{\mathsf{x}, \mathsf{z}\}$ is a data index, but doesn't have a corresponding primal data index. Therefore the two notions of data and prime data sequentiality do not quite coincide.

For examples of primal data indices, the primal data indices over $\mathbf{2} \times \mathbf{2}$ are $\{(\top, \bot)\}$ and $\{(\bot, \top)\}$, eliminating the inessential data index $\{(\top, \top)\}$. For **7** we have as primal data indices the singletons of primes: $\{\mathsf{a}\}$, $\{\mathsf{b}\}$, and $\{\mathsf{c}\}$, and we lose the ability, for instance, to separate between b and $\{\mathsf{x}, \mathsf{z}\}$, so that this set is primal data critical.

An advantage of data sequentiality is that it is not limited to prime-algebraic domains, while an advantage of primal data sequentiality is the elimination of superfluous indices. At any rate, over distributive concrete domains, where we intend to interpret data types, the indexable functions resulting from the two coincide. In the sequel we employ data sequentiality in chapter 5, where the underlying domains are not prime-algebraic, and we use primal data sequentiality in chapter 7, where we require the unconditional inducement property possessed by primal data index structures over a product (see proposition 4.3.1 and discussion in section 4.3.2 below).

4.3 Product

We formalize here the informal intuition that data indices over the product correspond to a choice of a component of the product, together with a data index in that component; making such an assumption about general index structures over the product produces a general definition of product of indexable domains.

This is consistent with previous attempts to formalize sequentiality. Definitions using argument positions have usually been applied to functions defined on products of flat domains [Mil77, Vui73]; an argument position in that case is just a choice of a component of the product. Similarly for KP-sequentiality, the cells of the product are taken to be the disjoint (and tagged) union of the cells of the components.

4.3.1 Data Indices over a Product

Thanks to the basic facts concerning the covering relation and the componentwise determination of isolated elements of the product, it is straightforward to show that the category of data-indexed Scott domains and indexable functions between them is cartesian.

4.3.1 Proposition *For Scott domains B_1 and B_2,*

> *(1) For $\ell = 1, 2$, if r is a data index of B_ℓ then $\amalg_\ell\, r = \{\amalg_\ell\, x \mid x \in r\}$ is a data index of $B_1 \times B_2$, and*
>
> $$(*) \qquad r \checkmark_{B_\ell} \pi_\ell\, z \quad \text{iff} \quad \amalg_\ell\, r \checkmark_{B_1 \times B_2} z.$$
>
> *If r consists of primes then so does $\amalg_\ell\, r$.*
>
> *(2) If s is a data index of $B_1 \times B_2$ then there is $\ell \in \{1, 2\}$ such that $\pi_\ell\, s$ is an index of B_ℓ. ℓ is uniquely determined if s is not a singleton.*
>
> *If s is trivial in its $(3 - l_0)$ component, meaning that there exist $\ell_0 \in \{1, 2\}$ and $x_0, y_0 \in B_{\ell_0}$ such that $s \not\checkmark \amalg_{\ell_0} x_0$ and $s \checkmark \amalg_{\ell_0} y_0$, then ℓ is uniquely determined, $\ell = \ell_0$, and $s = \amalg_\ell(\pi_\ell\, s)$.*
>
> *If s consists of primes then so does $\pi_\ell\, s$, ℓ is uniquely determined, and $s = \amalg_\ell(\pi_\ell\, s)$.*

Proof: We prove (1) for the case $\ell = 1$. If r is a data index of B_1 then $\amalg_1\, r = \{(x, \perp) \mid x \in r\}$ is a set of pairwise-inconsistent isolated elements of $B_1 \times B_2$, with $\mathsf{anchor}(\amalg_1\, r) = \amalg_1(\mathsf{anchor}\, r)$, so that it is indeed a data index of $B_1 \times B_2$. The relationship $(*)$ is easy to establish, as is the conservation of primality by injection, see 2.1.32.

(2) Let s be a data index of $B_1 \times B_2$, with $(x, y) = \mathsf{anchor}\, s$, *i.e.*, for every $z \in r$, $(x, y) -\!\!< z$. Now, if there are $x' \geq x$ and $y' \geq y$ such that $(x', y), (x, y') \in s$ then we have a contradiction to pairwise inconsistency of elements of s, since $(x', y) \Uparrow (x, y')$. Therefore, one of the components of all elements of s must be fixed: either $\pi_1\, s = \{x\}$ or $\pi_2\, s = \{y\}$. Assume without loss of generality that $\pi_2\, s = \{y\}$, and we set $\ell = 1$. It is now easy to show that $\pi_1\, s$ is a data index of B with anchor x, as desired. Uniqueness of ℓ for s with two or more elements follows from the fact that exactly one component of s can be fixed. (Whereas possible non-uniqueness for s a singleton corresponds precisely to the possibly non-unique choice of an anchor.)

If s separates $\amalg_{\ell_0} x_0$ and $\amalg_{\ell_0} y_0$ then $\pi_{\ell_0}\, s$ must separate x_0 and y_0 in B_{ℓ_0}, and it must be the case that $\pi_{(3-\ell_0)}\, s = \{\perp\}$. In this case, ℓ is uniquely determined, even for s a singleton, and, under the assumption that $\ell = 1$, it must therefore be the case that $\ell_0 = 1$, $y = \perp$, and $s = \amalg_1(\pi_1\, s)$.

If s consists of primes, then, under the assumption that $\ell = 1$, it must be the case that $y = \perp$, by 2.1.32, so that here again ℓ is uniquely determined, $s = \amalg_1(\pi_1\, s)$, and all elements of $\pi_1\, s$ must also be primes. ∎

By the above, we can define for every data index s over a product $B_1 \times B_2$ the ***main component*** of s, written $\mathsf{main}\, s$, to be ℓ such that $\pi_\ell\, s$ is a data index of B_ℓ. Part (2) above shows that $\mathsf{main}\, s$ is well-defined and unique when s is not a singleton, or when s has a trivial component, or when s is a primal data index — in fact, when s has a uniquely determined anchor. As for anchors, we will use the notation and terminology of the main component for all indices, keeping in mind that in the case of a singleton non-primal data index uniqueness is not guaranteed.

4.3.2 Proposition *The product projections and injections are data sequential, as is the pairing of data-sequential functions.*

The category of Scott domains and data-sequential functions is cartesian, with the concrete cartesian structure.

Proof: The product projections, injections and pairing of indexable functions are already known to be continuous.

If $r \in \mathcal{I}(\pi_1; z, Z)$ then $\iota_1\, r \in \mathcal{I}(z, Z)$, by $(*)$.

If $s \in \mathcal{I}(\iota_1; x, X)$ then $\mathsf{main}\, s = 1$, and $\pi_1\, s \in \mathcal{I}(x, X)$, again by $(*)$ since $s = \iota_1(\pi_1\, s)$.

If $s \in \mathcal{I}(\langle f_1\,, f_2\rangle\,; x, X)$ for data-indexable functions $f_1 : B \to B_1$ and $f_2 : B \to B_2$ then $\pi_\ell\, s \in \mathcal{I}(f_\ell; x, X)$ for $\ell = \mathsf{main}\, s$, and by indexability of f_ℓ there exists $r \in \mathcal{I}(x, X)$. ∎

Note that the same proofs may be used to establish the analogous properties for primal data indices over products, and that the category of dI-domains and primal data-sequential functions is cartesian.

4.3.2 The Product Assumption

The two parts of proposition 4.3.1 may be informally restated as follows:

(1) A computational step over a component of a product may also be seen as (or, may be injected into) a computational step over the product.

Or, equivalently: if you gain information about one of the components, then you gain information about the product.

Or, equivalently: the disjoint sum of indices over components may be seen as a subset of the indices over the product.

(2) A computational step over a product is essentially a computational step of one of the components.

Or, equivalently: if you gain information about the product then you gain information about one (and only one) of the components.

Or, equivalently: the indices over the product may be seen as a subset of the disjoint sum of the indices over the components.

One may now ask if these properties are limited to data indices, or if they can be generally applied to arbitrary index structures.

Principle (1), which we will call the *injective principle* for the product, seems to be a universal property that ought to hold in every model of computation, and we will require any definition of index structures over products of indexed domains to conform to it.

The notion of the injection of an index over one of the components into an index over the product is as described by the equation $(*)$: if r is an index of the ℓ'th component then its injection $\amalg_\ell r$ is an index over the product, and

$$r \checkmark_{B_\ell} \pi_\ell z \qquad \text{iff} \qquad \amalg_\ell r \checkmark_{B_1 \times B_2} z.$$

In contrast, the applicability of principle (2) necessitates, as we shall show, a careful consideration. We will refer to this principle as the *simple product assumption*, or just ***product assumption***.

There are actually two ways to formalize the assumption. The more general one is to say that every index over the product projects down to an index over a component of a product; the proposition shows that this is the case for data indices in general. A stronger requirement would be to ask that every index over the product is in fact induced by (*i.e.*, injected from) an index of a component; this is true only conditionally for data indices in general — only for those which are trivial in one component — but is unconditionally true for primal data indices, which always are trivial in one component. This unconditional inducement principle leads us to employ primal data index structures in chapter 7.

Proposition 4.3.1 thus asserts that the data index structure complies with the injective principle, and satisfies the product assumption.

The conjunction of the two parts is sufficient, in fact, to suggest a general construction of an index structure on the product of posets B_1 and B_2, given index structures $\mathcal{I}_1$ and $\mathcal{I}_2$ on the components: the ***(simple) product index structure*** $\mathcal{I}_\times$ is given by a tagged disjoint sum of the indices over the components,

$$\mathcal{I}_\times = (\{1\} \times \mathcal{I}_1) \cup (\{2\} \times \mathcal{I}_2),$$

with extents determined by

$$(**) \qquad r \checkmark_\ell \pi_\ell z \qquad \text{iff} \qquad (\ell, r) \checkmark_\times z.$$

We define $\mathsf{main}(\ell, r) = \ell$, and $\mathbf{Ans}(\ell, r) = \{\ell\} \times \mathbf{Ans}\, r$. It is easy to check that this defines a valid index structure over the product $B_1 \times B_2$.

This definition now presents us with a second way of constructing an index structure over a product of data indexed domains. But clearly, the representation used for (tagged) disjoint unions is the essential difference[1] between $(*)$ and $(**)$, or, for that matter, between the data index structure over a product of domains and the product index structure over a product of data indexed domains, leading us to assert that the two index structures for the product are isomorphic:

[1]For primal data indices this is in fact the only difference. General data indices encode additional information beyond the component and the index in that component, namely, a fixed element in the other component; but the isomorphism shows that this is an inessential difference.

4.3.3 Proposition *The data index structure $\mathcal{I}_0$ over a product $B_1 \times B_2$ of domains is isomorphic to the product index structure $\mathcal{I}_\times$ obtained for the product when the components are data-indexed, in the sense that the identity function over the product $B_1 \times B_2$ is indexable, both as a function from $(B_1 \times B_2, \mathcal{I}_0)$ to $(B_1 \times B_2, \mathcal{I}_\times)$, and as a function in the reverse direction.*

Proof: For every data index $s \in \mathcal{I}_0(x, X)$ over the product $B_1 \times B_2$, one may obtain an index with the same separation $\iota_\ell(\pi_\ell\, s) \in \mathcal{I}_0(x, X)$, where $\ell = \mathsf{main}\, s$.

To see that this is the case assume that $\iota_\ell(\pi_\ell\, s) \checkmark x$, so that $\pi_\ell\, s \checkmark \pi_\ell\, x$. If s has two or more elements then $\{\pi_{(3-\ell)}\, x\} = \pi_{(3-\ell)}\, X$, and hence $s \checkmark x$, a contradiction. If s is a singleton then $\mathsf{main}\, s$ is in fact not uniquely determined, and we may without loss of generality take $\iota_{(3-\ell)}(\pi_{(3-\ell)}\, s) \in \mathcal{I}_0(x, X)$ instead.

Now, by $(*)$ for $\pi_\ell\, s$, $\pi_\ell\, s \in \mathcal{I}_{B_\ell}(\pi_\ell; x, X)$, so that $(\ell, \pi_\ell\, s) \in \mathcal{I}_\times(x, X)$.

Conversely, if $(\ell, r) \in \mathcal{I}_\times(x, X)$ then $r \in \mathcal{I}_{B_\ell}(\pi_\ell; x, X)$, and $\iota_\ell\, r \in \mathcal{I}_0(x, X)$. ∎

But the construction of the product index structure is not limited to data-indexed domains; it serves to extend a product of the underlying domains to a product of indexed domains.

4.3.4 Proposition *If $\mathcal{I}_\times$ is the product index structure for the indexed dcpos $(B_1, \mathcal{I}_1)$ and $(B_2, \mathcal{I}_2)$, then the product projections $\pi_\ell : (B_1 \times B_2, \mathcal{I}_\times) \to (B_\ell, \mathcal{I}_\ell)$ and product injections $\iota_\ell : (B_\ell, \mathcal{I}_\ell) \to (B_1 \times B_2, \mathcal{I}_\times)$ are indexable, as is the pairing $\langle f_1 , f_2 \rangle : (B, \mathcal{I}) \to (B_1 \times B_2, \mathcal{I}_\times)$ of indexable functions $f_1 : (B, \mathcal{I}) \to (B_1, \mathcal{I}_1)$ and $f_2 : (B, \mathcal{I}) \to (B_2, \mathcal{I}_2)$.*

If $\mathcal{C}$ is a full subcategory of indexable functions between indexed dcpos obtained by restricting the underlying posets of the objects to some class that is closed under cartesian product, then the cartesian product of the underlying posets, equipped with the product index structure, is a categorical product in $\mathcal{C}$.

Proof: This proof repeats the proof of 4.3.2 with very slight adaptations. First, the product projections, injections and pairing of indexable functions are already known to be continuous.

If $r \in \mathcal{I}_1(\pi_1; z, Z)$ then $(1, r) \in \mathcal{I}_\times(z, Z)$, by $(**)$.

If $s \in \mathcal{I}_\times(\iota_1; x, X)$ then $\mathsf{main}\, s = 2$ leads to a contradiction, because then $s = (2, r)$ and $r \in \mathcal{I}_2(\pi_2 \circ \iota_1; x, X)$ with $\pi_2 \circ \iota_1 = \mathsf{K}\bot$. Therefore $\mathsf{main}\, s = 1$, $s = (1, r)$, and $r \in \mathcal{I}_1(x, X)$.

If $s \in \mathcal{I}_\times(\langle f_1 , f_2 \rangle\, ; x, X)$ for indexable functions $f_1 : B \to B_1$ and $f_2 : B \to B_2$ then $s = (\ell, r)$ for some $\ell \in \{1, 2\}$, and since $f_\ell = \pi_\ell \circ \langle f_1 , f_2 \rangle$, $r \in \mathcal{I}_\ell(f_\ell; x, X)$. By indexability of f_ℓ there exists $r \in \mathcal{I}(x, X)$. ∎

Criticality in the simple product index structure has a nice characterization in terms of componentwise criticality. The above proof, for instance, has a very simple reformulation using componentwise criticality and the characterization of indexable functions as functions preserving criticality and critical meets.

4.3.5 Proposition *If $\mathcal{I}_\times$ is the product index structure for $(B_1, \mathcal{I}_1)$ and $(B_2, \mathcal{I}_2)$ then $Z \subseteq (B_1 \times B_2, \mathcal{I}_\times)$ is critical iff both $\pi_1 Z$ and $\pi_2 Z$ are critical in their respective components.*

Proof: Since meet is taken componentwise, $\wedge Z = (\wedge \pi_1 Z, \wedge \pi_2 Z)$.

An index $r \in \mathcal{I}(\pi_\ell; \wedge Z, Z)$, with $\ell \in \{1, 2\}$, induces an index $(\ell, r) \in \mathcal{I}(\wedge Z, Z)$.

Conversely, an index $s \in \mathcal{I}(\wedge Z, Z)$ is induced by $r \in \mathcal{I}(\pi_\ell; \wedge Z, Z)$, with $s = (\ell, r)$ for some $\ell \in \{1, 2\}$. ∎

4.4 Towards Sequentiality at Higher Order

We turn now to higher-order sequentiality. By our previous development (corollaries 3.7.6 and 3.7.28) dI-domains are closed under the stably-ordered data-sequential function space and FM-domains are closed under the pointwise-ordered data-sequential function space (into a flat domain). This constitutes progress over the state of affairs for KP-sequentiality, since, for instance, (distributive) concrete domains are not closed under the stably-ordered stable function space.

Thanks to closure under function space, we are now in a position to ask if the application operation $\mathsf{app} : (B \rightarrow_{\mathrm{x}} B') \times B \rightarrow B'$ is data-sequential, *i.e.*, is it indexable when the data-sequential function space is equipped with the data index structure? Or, for that matter, we can ask if there is any index structure on the data-sequential function space that will make the application indexable? (And we may pose these questions twice, for both the pointwise order and the stable order on the function space.)

The answer, *given the product assumption*, is negative, independently of the index structure employed on the function space, and of the order used (stable or pointwise).

We also show that the data index structure is not adequate for use over the function space. From this we conclude that it is necessary to parameterize the index structure associated with a domain, a need anticipated by the definition of indexed domains, and that a different index structure should be used at higher order. Chapter 5 resumes the search for such an index structure.

4.4.1 Application and the Product Assumption

The following example demonstrates that application fails to be indexable under the product assumption.

Let $\mathsf{gf}_1, \mathsf{gf}_2, \mathsf{gf}_3 : \mathbf{Bool}^3 \rightarrow \mathbf{Bool}$ be the functions

$$\begin{aligned} \mathsf{gf}_1 &= [(\mathsf{T}, \mathsf{F}, \perp) \Rightarrow \mathsf{T}] \\ \mathsf{gf}_2 &= [(\perp, \mathsf{T}, \mathsf{F}) \Rightarrow \mathsf{T}] \\ \mathsf{gf}_3 &= [(\mathsf{F}, \perp, \mathsf{T}) \Rightarrow \mathsf{T}], \end{aligned}$$

easily seen to be data-sequential. Let their pairwise pointwise joins be $\mathsf{gf}_{1,2} = \mathsf{gf}_1 \vee^{\mathrm{p}} \mathsf{gf}_2$, $\mathsf{gf}_{1,3} = \mathsf{gf}_1 \vee^{\mathrm{p}} \mathsf{gf}_3$, and $\mathsf{gf}_{2,3} = \mathsf{gf}_2 \vee^{\mathrm{p}} \mathsf{gf}_3$; these functions are data-sequential as well, while the pointwise join of all three, $\mathsf{gf} = \mathsf{gf}_1 \vee^{\mathrm{p}} \mathsf{gf}_2 \vee^{\mathrm{p}} \mathsf{gf}_3$ is Gustave's function, which is stable but not data-sequential. The diagram 4.4 presents these functions, with the stable order given

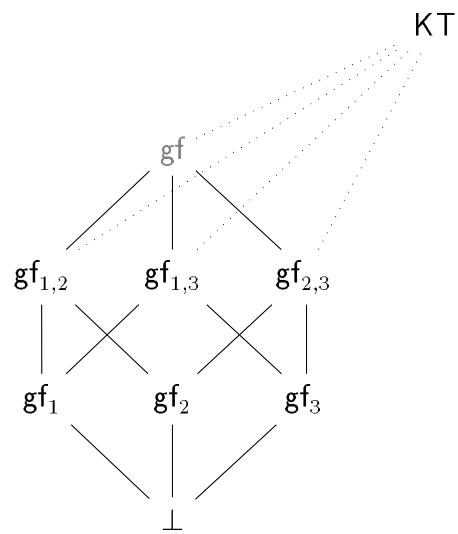

Figure 4.4: A partial diagram of $\mathbf{Bool}^3 \to \mathbf{Bool}$

by solid lines, pointwise order by solid and dotted lines, and the non-sequential function gf in gray.

In fact, the set

$$\{\mathsf{gf}_1 \vee^{\mathrm{p}} [(\mathsf{F},\mathsf{F},\mathsf{F}) \Rightarrow \mathsf{F}], \mathsf{gf}_2 \vee^{\mathrm{p}} [(\mathsf{F},\mathsf{F},\mathsf{F}) \Rightarrow \mathsf{F}], \mathsf{gf}_3 \vee^{\mathrm{p}} [(\mathsf{F},\mathsf{F},\mathsf{F}) \Rightarrow \mathsf{F}]\}$$

is an example of a pairwise consistent set of data-sequential functions that does not possess a data-sequential join (under either the pointwise or stable orders — the two coincide here). This example was given by Berry and Curien to illustrate that concrete domains — which are coherent, *i.e.*, have lubs for pairwise-consistent sets — are not closed under the stably-ordered KP-sequential function space [BC82]. The reason we add $[(\mathsf{F},\mathsf{F},\mathsf{F}) \Rightarrow \mathsf{F}]$ as a disjunct is to rule out as a potential join the constant true function KT, which does dominate gf in the pointwise order.

Now turn to the application function $\mathsf{app} : (\mathbf{Bool}^3 \to \mathbf{Bool}) \times \mathbf{Bool}^3 \to \mathbf{Bool}$. Let $z = (\mathsf{gf}_1, (\perp,\perp,\perp))$ and

$$Z = \{(\mathsf{gf}_1, (\mathsf{T},\mathsf{F},\perp)), (\mathsf{gf}_{1,2}, (\perp,\mathsf{T},\mathsf{F})), (\mathsf{gf}_{1,3}, (\mathsf{F},\perp,\mathsf{T}))\}\,.$$

Importantly, $Z \subseteq \uparrow z$ and $z = \wedge Z$, and this is true both for the stable and pointwise orders on the function space. Now, $\mathsf{app}\, z = \perp$ and $\mathsf{app}\, Z = \{\mathsf{T}\}$, so that there exists an index $r' \in \mathcal{I}(\mathsf{app}; z, Z)$ separating the images of z and Z under app.

Does there exist an index that separates z and Z? Under the product assumption, *i.e.*, using the product index structure over $(\mathbf{Bool}^3 \to \mathbf{Bool}) \times \mathbf{Bool}^3$, the answer is negative, because, if s is such an index, then $s = (1, r)$ or $s = (2, (\ell, r))$ for $\ell \in \{1,2,3\}$. If $\mathsf{main}\, s = 1$, for instance, r would need to separate $\pi_1\, z$ from $\pi_1\, Z$; but this is impossible, since $\pi_1\, z \in \pi_1\, Z$. The other three possibilities are ruled out by the same reasoning, and app thus fails to be indexable.

The failure of app to be indexable may also be demonstrated using criticality. $z = \wedge Z$, and Z is critical, because its projections $\pi_1 Z$, and $\pi_\ell(\pi_2 Z)$ for $\ell \in \{1, 2, 3\}$ are all critical, since they each have a minimal element. But $\mathsf{app}\, z < \wedge(\mathsf{app}\, Z)$, and app fails to be indexable because it does not preserve this critical meet.

This example relies on the definitions of indexability and data sequentiality and on the product assumption. But the only use made of data sequentiality is to justify the existence of an index with extent $\{\mathsf{T}, \mathsf{F}\}$ over **Bool**. We do not rely in any way on any specific index structure over the function space, and the result is valid whether the stable or the pointwise order is used on the data-sequential function space.

We therefore attribute the failure of indexability of app in the indexed framework entirely to the product assumption, appropriately labelled at this point as the "simple" product assumption to underscore our newly found insight. We claim that the simple product assumption is adequate for formalizing first-order computation, but it is not adequate for a satisfactory treatment of higher-order PCF-sequentiality: in higher-order PCF-sequential computation it must be possible to increment the information about a product (say, $(B \to B') \times B$) in a way that increments the information about both components simultaneously.

4.4.2 Data Sequentiality at Higher Order?

We have ruled out the product assumption for higher-order computation. There still remains a question whether the data indices may be used over function spaces. That data sequentiality complies with the product assumption is a first negative indication, and in fact we proceed to give more evidence that it is not adequate for modelling higher-order PCF-sequential computation.

The qualification here with respect to PCF is important, since, as we have already indicated, there are different notions of sequential computation. Notably, data sequentiality, as well as the product assumption, is appropriate for higher-order computation over Berry and Curien's sequential algorithms between DCDSs. Essentially, this is because the enabling component of a DCDS is used to constrain the construction of a sequential algorithm so that applying it yields an increase in the information which is a covering step; and a covering step in a component is a covering step in the product, in agreement with the product assumption. This, we believe, explains the success of KP-sequentiality in capturing computation in this framework.

This description does not apply, however, to computation over arrow types in PCF, where no such discipline is imposed. As we have seen, application is incompatible with the product assumption, and we show now that application and related functionals may result in an increase in the information that is more than a covering step, thereby violating the central aspect of data sequentiality.

4.4.1 Example Consider for instance $\mathsf{app} : (\mathbf{2} \to \mathbf{2}) \times \mathbf{2} \to \mathbf{2}$. If $\mathsf{app}(f, x) = \top$, there are two possibilities: either $(f, x) \geq ([\bot \Rightarrow \top], \bot)$, or $(f, x) \geq ([\top \Rightarrow \top], \top)$. Under the pointwise order, both possibilities are not covering steps of $(\bot, \bot)$, while under the stable order the latter is not a covering step. •

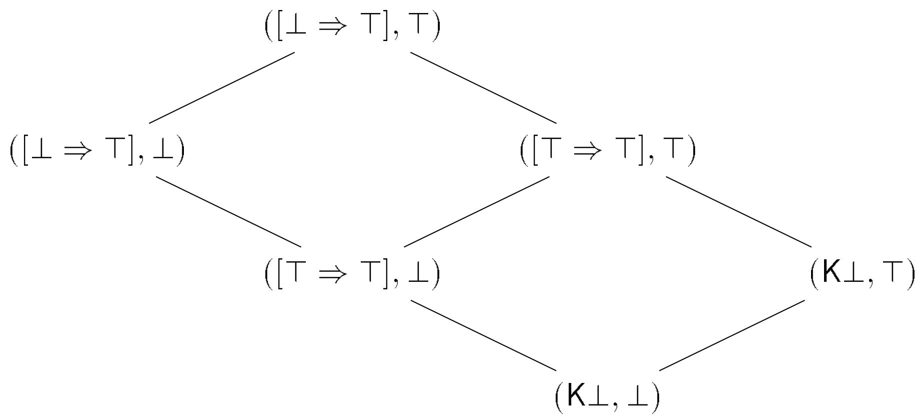

Figure 4.5: The domain $(\mathbf{2} \to^{\mathrm{p}} \mathbf{2}) \times \mathbf{2}$, function space ordered pointwise

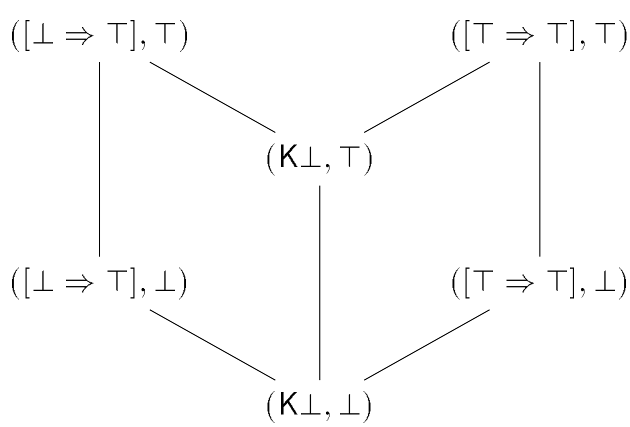

Figure 4.6: The domain $(\mathbf{2} \to^{\mathrm{s}} \mathbf{2}) \times \mathbf{2}$, function space ordered stably

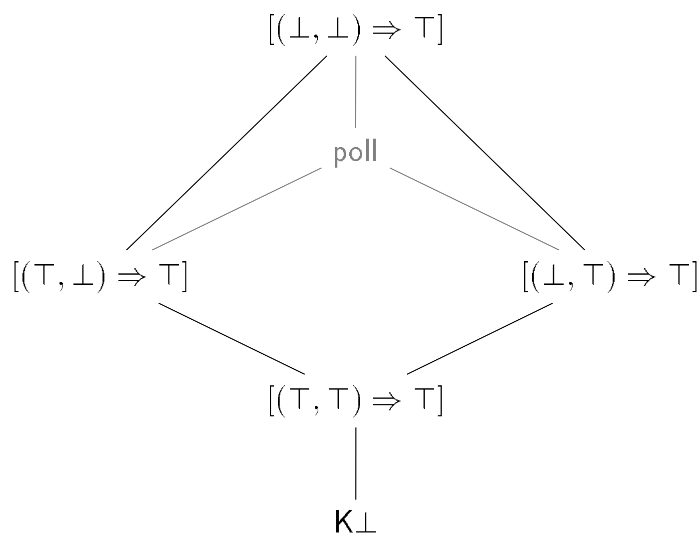

Figure 4.7: The domain $\mathbf{2} \times \mathbf{2} \to^{\mathrm{p}} \mathbf{2}$, ordered pointwise

4.4.2 Example For another example, this time over a function space in isolation, consider $\mathsf{app} : (\mathbf{2} \times \mathbf{2} \to \mathbf{2}) \times (\mathbf{2} \times \mathbf{2}) \to \mathbf{2}$, with its right argument fixed to $\bot = (\bot, \bot)$, that is, look at $\phi = \mathsf{app} \circ \underline{\nu}_1 : (\mathbf{2} \times \mathbf{2} \to \mathbf{2}) \to \mathbf{2}$. Now, if $\phi f = \top$, and we know nothing about f — our current approximation is $\mathsf{K}\bot$ — then we can deduce that $f(\bot, \bot) = \top$, and $f \geq^{\mathrm{p}} [(\bot, \bot) \Rightarrow \top]$ (an equality since there is no larger function). But this is not a covering step when the pointwise order is employed on the function space. •

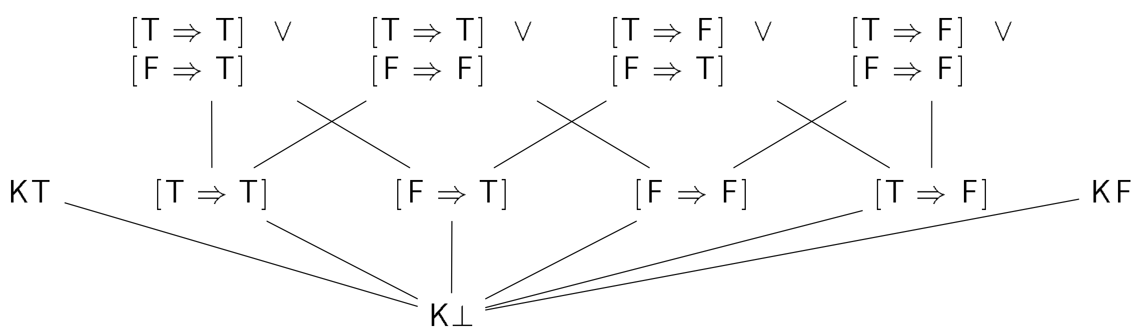

Figure 4.8: The domain $\mathbf{Bool} \to^{\mathrm{s}} \mathbf{Bool}$, ordered stably

4.4.3 Example To obtain a similar example for a function space in isolation, under the stable order, requires a more involved construction. Let

$$\begin{aligned} \phi \quad &: \quad (\mathbf{Bool} \to \mathbf{Bool}) \to \mathbf{Bool} \\ &= \quad \lambda f \in \mathbf{Bool} \to \mathbf{Bool} \,.\, f(f\,\mathsf{T}) \\ &= \quad \mathsf{app} \circ \langle \mathsf{id} \,, \mathsf{app} \circ \langle \mathsf{id} \,, \mathsf{KT} \rangle \rangle \,. \end{aligned}$$

Then, if $\phi f = \mathsf{T}$ there are two possibilities: $f \geq^{\mathrm{p}} [\mathsf{T} \Rightarrow \mathsf{T}]$ (this includes $f = \mathsf{KT}$), or $f \geq^{\mathrm{p}} [\mathsf{T} \Rightarrow \mathsf{F}] \vee [\mathsf{F} \Rightarrow \mathsf{T}]$ (an equality). The latter is not a covering step of $\mathsf{K}\bot$ in the stable order.

The PCF term defining this ϕ employs *imbrication* [BCL85, p. 129], that is, the input, represented here by f, is itself applied to some term that also refers to f, so that, as the example illustrates, we obtain a "compound" increment. •

Chapter 5

Constant-applicative Sequentiality

This chapter presents a first attempt to construct a model of higher-order PCF-sequential computation using the framework of indexed computation. The previous chapter provides us with a formalization at first order, using data sequentiality and the product assumption, but as we have shown, these are not adequate for a general treatment of higher-order PCF-sequentiality.

In order to advance our understanding of computation at higher order, this chapter makes a compromise: we will consider higher-order computation, and attempt to formalize it as an index structure over the function space; but we will retain the product assumption, with the consequence that any model constructed will not be a fully abstract model for PCF. Instead, we will make a simplifying assumption concerning computation over a function space, and a corresponding restriction on the language under consideration. The end result of this chapter is a restricted notion of *constant-applicative* (or *ca*) sequential computation over function spaces, that, combined with data sequentiality at first order and the product assumption, yields a fully abstract model for ca-PCF, a sub-language of PCF obtained by only allowing variables of arrow types to be applied to closed terms.

Of course, we will not obtain a CCC, and application will be excluded from the constant-applicative model. This is definitely not meant as a comment on the general suitability of CCCs to provide models for languages with higher-order features. Rather, since the ca-model is fully abstract for ca-PCF, this merely indicates that a CCC is not needed to model the restricted higher-order features of ca-PCF. This is just another example of the need to choose the right tools for the task at hand, or, in the semantics arena, choosing the right model and semantic treatment for the language at hand and the desired goals.

This chapter is partly a digression from the main line of development, and its definitions and results will not be used in the sequel. But from the methodological and historical points of view, it serves as a gentle, and perhaps essential, introduction to the formalization of higher-order PCF-sequential computation. The general discussion here concerning the definition of indices at higher order will remain valid, and our further attempt in chapter 7 to formalize computation at higher order will start by attempting to refine ca-sequentiality. In particular, the proof of definability of isolated elements given here for ca-PCF ought to serve as a model for such a proof in the general case.

5.1 Higher-order Computation

The first observation to make here is that the sole primitive available for computation over function spaces is application. (We ignore the fixpoint primitive, which intuitively should not be necessary for definability of isolated elements.) Thus, we are seeking an *applicative index structure* that will formalize the computational behavior of application.

Atomicity of sequential computation at first order is captured by the covering relation, as formalized by data sequentiality. We have shown that this is inappropriate at higher order, and are now looking for a substitute notion of atomicity. A first attempt is to say that an increment to an approximation f is a step function $[x \Rightarrow y]$ with $fx < y$, so that we go to an approximation $f \vee^{\mathrm{p}} [x \Rightarrow y]$. Intuitively, this corresponds to application to x, with observed result y. This is even a covering step in the stable order if $fx = \bot$, but not in the pointwise order. But as the concluding example of chapter 3 shows, this is not sufficient, since imbrication presents some kind of compound increment of possibly more than one step function. Therefore, instead of formalizing atomicity by identifying the class of possible increments by application, we will simply encode the act of application itself. To do this, we now need to ask what is the information that needs to be present in such an encoding.

Of course, we have implicitly assumed here that we may only learn about the function space by observing its extensional, applicative, behavior. Making other assumptions here, for instance, that we may peek inside the black box corresponding to the term or computational agent, is feasible, but would lead to an intensional treatment.

Prompters and Residual Indices

An essential aspect that we need to capture is that all the potential increments resulting from application have in common the *argument* to the function being applied, modulo dependence of this argument on the input. This is a global constraint on the potential increments, similar to the existence of an anchor for a data index. We will refer to this argument as a *prompter*, since it may be seen as prompting the relevant function space for more information.

Once we apply an element of a function space to a prompter we obtain a result in the target domain. It is only by observing this value — that is, by observing its increase from the current approximation fx, if the current approximation in the function space is f and (the current approximation for) the prompter is x — that we may deduce an increase over the function space. Thus, we also need to specify, as part of an index that describes application, what the *residual index* is, over the result of the application.

Note the syntactic parallels. If we are writing a PCF term to compute over a function space, we will have at our disposal some free variable `f` of arrow type $\tau \rightarrow \tau'$ that represents that function space, and we will ultimately obtain an atomic increase in information by applying `f` to some term M, which plays the role of a *(syntactic) prompter*. Pursuing the analogy between indices and contexts , the residual index corresponds to a context with hole of type τ', into which the application is placed. Thus, a prompter M of type τ and a context $P[-]$ with hole of type τ' determine a context $P[(-M)]$ with hole of type $\tau \rightarrow \tau'$, which ought to correspond to the applicative indices that we are trying to formalize.

Dependent Prompters

Let us look closer at prompters. On the syntactic side, they are terms of type appropriate as an argument to the arrow type being subjected to attention, and, importantly, they may include any of the free variables provided by the context.

A simple example is the following functional:

$$\begin{aligned}\phi_0 \quad = \quad & \lambda f : \mathbf{Bool} \times \mathbf{Bool} \to \mathbf{Bool}\ .\\ & \mathsf{if}\ f(\mathsf{T},\bot)\&f(\mathsf{F},\mathsf{T})\&\neg f(\mathsf{F},\mathsf{F})\ \mathsf{then}\ \mathsf{T}\ \mathsf{else}\ \bot,\end{aligned}$$

where & is the left-strict-and function, written in infix notation, and $\neg$ is negation[1] . When ϕ_0 computes over the function space $\mathbf{Bool} \times \mathbf{Bool} \to \mathbf{Bool}$, it may be seen as successively increasing its information about its input. If the result of the computation is T (as may be determined by a residual index over the output domain $\mathbf{Bool}$) then it may be deduced that the computation followed the sequence of approximations

$$\bot, \quad [(\mathsf{T},\bot) \Rightarrow \mathsf{T}], \quad \begin{matrix}[(\mathsf{T},\bot) \Rightarrow \mathsf{T}] & \vee,\\ [(\mathsf{F},\mathsf{T}) \Rightarrow \mathsf{T}] & \end{matrix} \quad \begin{matrix}[(\mathsf{T},\bot) \Rightarrow \mathsf{T}] & \vee.\\ [(\mathsf{F},\mathsf{T}) \Rightarrow \mathsf{T}] & \vee\\ [(\mathsf{F},\mathsf{F}) \Rightarrow \mathsf{F}] & \end{matrix}$$

Each step of the computation corresponds to an increment of the information about the input in $\mathbf{Bool} \times \mathbf{Bool} \to \mathbf{Bool}$, by prompting it with a prompter in $\mathbf{Bool} \times \mathbf{Bool}$. Divergence of any step would imply divergence of the entire computation. The prompters used here are *constant*: they do not depend on the input, and, as terms, they are closed — contain no free variables.

Dependent prompters, in contrast to constant prompters, are those prompters that do depend on the input. Consider the following examples:

$$\begin{aligned}\phi_1 \quad &= \quad \lambda f : B \to B'\ .\ \lambda x : B\ .\ fx,\\ \phi_2 \quad &= \quad \lambda f : \mathbf{Bool} \times \mathbf{Bool} \to \mathbf{Bool}\ .\\ & \qquad \mathsf{if}\ f(f(\mathsf{T},\bot),f(\bot,\mathsf{T}))\&f(\mathsf{T},\mathsf{F})\&f(\mathsf{F},\mathsf{T})\&\neg f(\mathsf{F},\mathsf{F})\ \mathsf{then}\ \mathsf{T}\ \mathsf{else}\ \bot,\\ \phi_2' \quad &= \quad \lambda f : \mathbf{2} \times \mathbf{2} \to \mathbf{2}\ .\\ & \qquad f(f(\top,\bot),f(\bot,\top)).\end{aligned}$$

ϕ_1 is the application function for the function space $B \to B'$; it may be said to compute over the function space, prompting it with the prompter x which depends on the input. ϕ_2 is the least functional that maps left-strict-or and right-strict-or to true, and ϕ_2' is a simplified version of ϕ_2. These functionals are definable in PCF, using *imbrication*, which, in this context, means that the prompter is dependent on the function space over which we are computing; we will call such a prompter a *self-dependent prompter*.

At this point, and for the rest of this chapter, we will not attempt to treat dependent prompters; we will restrict our attention exclusively to constant prompters, leading to the

[1] We should really be talking of computation as described by a term in the language, rather than a function, but the correspondence of ϕ_0 to a PCF term is clear (up to currying), and we will take the liberty of not making a sharp distinction here. Going from terms to functions will not lose the uniqueness of the computation strategy of the term if the primitives used have a unique computation strategy — *e.g.*, the conditional and left-strict-and, as in ϕ_0, but not doubly-strict-and.

notion of *constant-applicative sequentiality*. The examples ϕ_1 and ϕ_2, ϕ_2' are thus ruled out of consideration. We will resume the search for a general notion of *applicative sequentiality* in chapter 7.

5.2 The Constant-applicative Model

Following our decision to restrict attention to constant-applicative sequentiality, we proceed now to define the constant-applicative model, or the ca-model, for the constant-applicative sub-language of PCF, ca-PCF. This section presents the type interpretation, and the next section will give a semantics for ca-PCF in the model, and show full abstraction.

5.2.1 Constant-applicative Indices

We define the ***constant-applicative*** or ***ca***-index structure over the indexable function space between indexed Scott domains $(B, \mathcal{I})$ and $(B', \mathcal{I}')$ to be

$$\mathcal{I}_{\mathrm{ca}} = \{[x \Rightarrow^{\mathrm{ca}} r'] \mid x \in \mathbf{K}\, B \ \ \& \ \ r' \in \mathcal{I}'\}\,.$$

$[x \Rightarrow^{\mathrm{ca}} r']$ is the ***ca-index*** corresponding to the constant prompter $x \in \mathbf{K}\, B$ and the residual index $r' \in \mathcal{I}'$. We set

$$[x \Rightarrow^{\mathrm{ca}} r'] \checkmark_{\mathrm{ca}} f \qquad \text{iff} \qquad r' \checkmark' fx,$$

and

$$\mathbf{Ans}[x \Rightarrow^{\mathrm{ca}} r'] = \{[x \Rightarrow^{\mathrm{ca}} s'] \mid s' \in \mathbf{Ans}\, r'\}\,.$$

Thus, $[x \Rightarrow^{\mathrm{ca}} r']$ is an answer index iff r' is an answer index.

5.2.1 Proposition *The ca-index structure over the pointwise-ordered indexable function space is a well defined index structure, and it is stable if $\mathcal{I}'$ is stable.*

Proof: All required properties of $\mathcal{I}_{\mathrm{ca}}$ follow from the corresponding properties for $\mathcal{I}'$. Some partial details:

T_0-separation: If $f \neq g$ then there exists x such that $fx \neq gx$, and by continuity x may be assumed isolated. By T_0-separation of $\mathcal{I}'$, there exists r' that separates fx and gx, so that $[x \Rightarrow^{\mathrm{ca}} r']$ separates f and g.

Countability of $\mathbf{Ans}\, \mathcal{I}_{\mathrm{ca}}$ follows from countability of $\mathbf{K}\, B$ and $\mathbf{Ans}\, \mathcal{I}'$.

Continuity: If $[x \Rightarrow^{\mathrm{ca}} r'] \checkmark_{\mathrm{ca}} (\bigvee^{\mathrm{p}} F)$ for some pointwise-directed set F then $r' \checkmark' (\bigvee^{\mathrm{p}} F)x = \bigvee(Fx)$, so that by continuity of $\mathcal{I}'$, $r' \checkmark' f_0 x$ for some $f_0 \in F$, and $[x \Rightarrow^{\mathrm{ca}} r'] \checkmark_{\mathrm{ca}} f_0$.

Stability follows similarly if $\mathcal{I}'$ is stable, recalling that the meet in the pointwise ordered indexable function space is taken pointwise (proposition 3.7.10). ∎

5.2.2 The Constant-applicative Category

Let CA be the category of stably indexed FM-domains and indexable functions. We will use as a distinguished product the poset product, equipped with the simple product index structure. By 4.3.4 this gives CA a cartesian structure.

For the *constant-applicative* exponentiation $B \to_{\mathrm{ca}} B'$ of two objects, we will take the indexable function space between the two, ordered pointwise, with the ca-index structure. The exponentiation will only be taken for B' a tree domain, since that is the situation where we are guaranteed that the exponentiation will be an FM-domain, by 3.7.28. Even when this exponentiation object is an object of the category, it fails to qualify as a categorical exponentiation, since the use of the simple product is sufficient to rule out indexability of application. Thus, this category is not a CCC, but it is self-enriched to a limited extent, in the sense that certain homsets may be seen as objects of the category.

We have chosen FM-domains as underlying our objects since we have been able to prove a limited form of closure under the pointwise-ordered indexable function space (proposition 3.7.28). A word is in order concerning the choice of the pointwise order. The stable order over function spaces was introduced in order to salvage stability of application; but since we are at this point looking for a restricted model that does not include application, we see no need for the stable order. Technically we rely below on the meet in pointwise ordered indexable function space being determined pointwise (proposition 3.7.10); this property is not true for the stably ordered indexable function space, so that the proofs below would not apply directly, were we to employ the stable order.

We now establish a few facts about the ca-index structure and the category. It is important to note that this is all stated under the specific definitions of product and exponentiation adopted in CA, and, in particular, the codomain of the exponentiation is assumed to be a tree domain. To emphasize this, we will refer to the property of being a function of CA as ***ca-indexability***.

5.2.2 Proposition *A non-empty finite set $F \subseteq B \to_{\mathrm{ca}} B'$ is critical iff for all $x \in \mathbf{K}\,B$, $Fx = \{fx \mid f \in F\}$ is critical.*

Proof: Note that by definition $[x \Rightarrow^{\mathrm{ca}} r'] \in \mathcal{I}_{\mathrm{ca}}(\wedge^{\mathrm{p}} F, F)$ iff $r' \in \mathcal{I}'((\wedge^{\mathrm{p}} F)x, Fx)$, which is in turn equivalent to $r' \in \mathcal{I}'(\wedge(Fx), Fx)$, since $(\wedge^{\mathrm{p}} F)x = \wedge(Fx)$. ∎

Since application $\mathsf{app} = \mathsf{uncurry}\,\mathsf{id}$ is not ca-indexable, uncurrying fails to preserve ca-indexability. Currying does preserve ca-indexability.

5.2.3 Proposition *Currying preserves ca-indexability, i.e., if $f : B_1 \times B_2 \to B'$ is a ca-indexable function for B' flat then $\mathsf{curry}\, f : B_1 \to (B_2 \to_{\mathrm{ca}} B')$ is ca-indexable.*

Proof: First we need to show that $\mathsf{curry}\, f$ is well defined, that is, that for every $x \in B_1$, $\mathsf{curry}\, fx$ is ca-indexable. Continuity has already been shown. If Y is a critical set of B_2 then $\{x\} \times Y$ is critical, by 4.3.5, so that

$$\mathsf{curry}\, fxY = f(\{x\} \times Y)$$

is critical, by ca-indexability of f. Moreover, the critical meet is preserved:

$$\begin{array}{rcll} \wedge(\mathsf{curry}\, fxY) & = & \wedge(f(\{x\} \times Y)) & \\ & = & f(x, \wedge Y) & (f \text{ ca-indexable}) \\ & = & \mathsf{curry}\, fx(\wedge Y). & \end{array}$$

Therefore $\mathsf{curry}\, fx$ is ca-indexable.

We now show that $\mathsf{curry}\, f$ is ca-indexable. Let X be a critical set of B_1, so that for every $y \in B_2$, $X \times \{y\}$ is critical, and

$$\mathsf{curry}\, fXy = f(X \times \{y\})$$

is critical, by ca-indexability of f. Since this is so for every y, it follows by 5.2.2 that $\mathsf{curry}\, fX$ is critical. Moreover, for every y we have

$$\begin{array}{rcll} (\wedge(\mathsf{curry}\, fX))y & = & \wedge(\mathsf{curry}\, fXy) & \text{(meet taken pointwise)} \\ & = & \wedge f(X \times \{y\}) & (f \text{ ca-indexable}) \\ & = & f(\wedge X, y) & \\ & = & \mathsf{curry}\, f(\wedge X)y. & \end{array}$$

This means that $\wedge(\mathsf{curry}\, fX) = \mathsf{curry}\, f(\wedge X)$, so that $\mathsf{curry}\, f$ preserves critical meets as well as criticality, and is ca-indexable. ∎

While application in general is not ca-indexable, application of a fixed ca-indexable function $f \in B \to B'$, *i.e.*, the functional $\lambda z \in B \;.\; fz$, coincides with f and is therefore ca-indexable. More importantly, application to a fixed argument is ca-indexable.

5.2.4 Proposition *For every $z \in B$, the function $\lambda f \in B \to_{\mathrm{ca}} B' \;.\; fz$ is ca-indexable.*

Proof: If F is a critical set of $B \to_{\mathrm{ca}} B'$ then Fz is critical, and $\wedge(Fz) = (\wedge F)z$ because meet is taken pointwise. ∎

5.2.3 Full Uncurrying

The meaning $[\![\tau \to \tau']\!]$ of an arrow type in the model will essentially be taken to be the ca-indexable function space between $[\![\tau]\!]$ and $[\![\tau']\!]$. A further refinement is still needed before we are able to define a type interpretation. Type interpretations are usually defined in CCCs, where there is an isomorphism

$$[\![\tau_1]\!] \to ([\![\tau_2]\!] \to [\![\tau']\!]) \cong ([\![\tau_1]\!] \times [\![\tau_2]\!]) \to [\![\tau']\!]$$

via currying and uncurrying. In that case, it doesn't really matter which of the two is taken to define $[\![\tau_1 \to (\tau_2 \to \tau')]\!]$, and there is a natural preference for the lefthand side, since then $[\![\tau \to \tau']\!]$ can be defined in terms of its components to be $[\![\tau]\!] \to [\![\tau']\!]$.

In CA, however, there is no such isomorphism, since uncurrying does not preserve ca-indexability. In particular, any function whose source is a flat domain is ca-indexable, because there is always one and only one possible way to increase the information about the input.

For example, the curried version of the parallel-or, $\mathsf{curry\,por} : \mathbf{Bool} \to (\mathbf{Bool} \to_{\mathrm{ca}} \mathbf{Bool})$ is trivially indexed. Following the discussion in section 3.4.1, we deduce from this that sequentiality of a function must be judged by looking at the function in its *fully uncurried* form, *i.e.*, such that $\mathsf{uncurry}$ has been repeatedly applied as much as possible. We will rely on the type structure to guide this.

Note that there are two possibilities now. We may either take $[\![\tau \to \tau']\!]_{\mathrm{ca}}$ to be a subset of $[\![\tau]\!]_{\mathrm{ca}} \to_{\mathrm{ca}} [\![\tau']\!]_{\mathrm{ca}}$ consisting only of functions that remain ca-indexable under repeated uncurrying, or we could take it to be $[\![\tau_1 \times \cdots \times \tau_n \to \rho]\!]_{\mathrm{ca}}$, where $\tau_1 \times \cdots \times \tau_n \to \rho$ is the fully uncurried form of $\tau \to \tau'$ (which must be of the form $\tau_1 \to \cdots \to \tau_n \to \rho$). We adopt the latter approach here. The first option is missing type information that would indicate how far to go for full uncurrying; we adopt it, however, for the fully sequential type interpretation in chapter 7.

Thus, we will regard as ca-sequential only those functions whose fully uncurried form is ca-indexable, and to build a model including only such functions we will interpret arrow types in their fully uncurried form.

Luckily, our limited result for closure of FM-domains under the ca-sequential function space (proposition 3.7.28) will apply, now that the target of all functions will be flat.

5.2.4 The Ca-sequential Type Interpretation

We now define the ca-sequential type interpretation $[\![-]\!]_{\mathrm{ca}}$, mapping each type τ to an indexed FM-domain $[\![\tau]\!]_{\mathrm{ca}}$. The definition is parameterized by a type interpretation $[\![-]\!]_0$ of ground types as flat domains.

- For a ground type ρ, $[\![\rho]\!]_{\mathrm{ca}}$ is the flat domain $[\![\rho]\!]_0$, equipped with the data index structure.

- Each arrow type τ can be written uniquely in the form $\tau_1 \to \cdots \to \tau_n \to \rho$, where $n \geq 1$ and ρ is ground. We define $[\![\tau]\!]_{\mathrm{ca}}$ to be the ca-exponentiation $[\![\tau_1]\!]_{\mathrm{ca}} \times \cdots \times [\![\tau_n]\!]_{\mathrm{ca}} \to_{\mathrm{ca}} [\![\rho]\!]_{\mathrm{ca}}$.

We assume that we have at least ground types `Bool` and `Nat`, corresponding to the flat domains of truth values and natural numbers respectively.

We may refer to a function $f \in [\![\tau]\!]_{\mathrm{ca}}$ in this model as ***ca-sequential***. But note that ca-sequentiality of a function is relative to the type in whose interpretation it resides, which determines the extent of uncurrying.

5.3 A Sub-language of PCF

We now introduce ca-PCF, a sub-language of PCF that incorporates the simplifying assumption that functions may only be prompted by constant prompters, and give it a model in CA. The presentation largely parallels that of PCF and, with additional complications arising from the full uncurrying that is involved in the interpretation.

5.3.1 The ca-PCF Typing System and Semantics

Raw (untyped) terms are built from a given set of constants, identifiers, application and abstraction in the usual way, as in PCF. We define axioms and inference rules for judgements of the form $\Gamma \vdash^{\mathrm{ca}} M : \tau$, to be read as: the term M has (ca-PCF) type τ in type context Γ.

The essential restriction imposed by our typing system is that a variable of functional type may only be applied to closed terms. This captures the simplifying assumption that prompters cannot depend on the input. For convenience we also require that a variable of functional type be applied successively to as many arguments as needed to obtain a result of ground type. This restriction is not important, but removing it would necessitate an additional syntactic category for terms of arrow type that may only be applied to closed terms.

The terms of ca-PCF are those terms M for which a judgement $\Gamma \vdash^{\mathrm{ca}} M : \tau$ is derivable. We use L to range over terms, and K to range over closed terms. A term K is closed iff it has no free identifiers; equivalently, if $\vdash^{\mathrm{ca}} K : \tau$ is derivable for some τ.

We define a semantic function $[\![-]\!]_{\mathrm{ca}}$ for judgements $\Gamma \vdash^{\mathrm{ca}} M : \tau$ by induction on the derivation of the judgement. Throughout we assume that Γ has form $v_1 : \gamma_1, \ldots, v_m : \gamma_m$ and that τ is written in the form $\tau_1 \to \cdots \to \tau_n \to \rho$, where ρ is ground. The meaning of $\Gamma \vdash^{\mathrm{ca}} M : \tau$ will be

$$\begin{aligned} [\![\Gamma \vdash^{\mathrm{ca}} M : \tau]\!]_{\mathrm{ca}} &\in [\![\gamma_1 \to \cdots \to \gamma_m \to \tau]\!]_{\mathrm{ca}} \\ &= [\![\gamma_1]\!]_{\mathrm{ca}} \times \cdots \times [\![\gamma_m]\!]_{\mathrm{ca}} \times [\![\tau_1]\!]_{\mathrm{ca}} \times \cdots \times [\![\tau_n]\!]_{\mathrm{ca}} \to_{\mathrm{ca}} [\![\rho]\!]_{\mathrm{ca}}. \end{aligned}$$

Note that the environment is "blended into" the semantic domains; this is necessary, since all functions in the model, including the meanings of terms, are to be fully uncurried.

We assume a semantic function $[\![-]\!]_0$ for constants such that $[\![c]\!]_0 \in [\![\tau]\!]_0$ for each constant c of type τ. We assume the standard collections of ground types and PCF constants, except for the recursion constants $\mathtt{Y}^\tau$, and we assume that the constants are given the standard interpretation, modulo their being fully uncurried.

We use the following notations for associative isomorphisms and projections over finite products,

$$\begin{aligned} \mathsf{assoc}_{B_1,\ldots,B_i;B_{i+1},\ldots,B_n} &: (B_1 \times \cdots \times B_i \times B_{i+1} \times \cdots \times B_n) \\ &\quad \cong (B_1 \times \cdots \times B_i) \times (B_{i+1} \times \cdots \times B_n) \\ \mathsf{assoc}_{\Gamma,\tau_1\to\cdots\to\tau_n\to\rho} &= \mathsf{assoc}_{[\![\gamma_1]\!]_{\mathrm{ca}},\ldots,[\![\gamma_m]\!]_{\mathrm{ca}};[\![\tau_1]\!]_{\mathrm{ca}},\ldots,[\![\tau_n]\!]_{\mathrm{ca}}}, \\ \pi_{i;B_1,\ldots,B_n} &: B_1 \times \cdots \times B_n \to B_i \\ \pi_{i;\Gamma} &= \pi_{i;[\![\gamma_1]\!]_{\mathrm{ca}},\ldots,[\![\gamma_m]\!]_{\mathrm{ca}}}. \end{aligned}$$

These generalized associative isomorphisms and projections are ca-sequential, following the general results about indexed domain product.

The following are the axioms and inference rules for the typing system, together with the definition of the semantic function $[\![-]\!]_{\mathrm{ca}}$.

- Constants:

$$\frac{}{\Gamma \vdash^{\mathrm{ca}} c : \tau}\ \mathsf{const}$$

$$\begin{aligned}
[\![\Gamma \vdash^{\mathrm{ca}} c : \tau]\!]_{\mathrm{ca}} &= \lambda(x_1 \in [\![\gamma_1]\!]_{\mathrm{ca}}, \ldots x_m \in [\![\gamma_m]\!]_{\mathrm{ca}}, y_1 \in [\![\tau_1]\!]_{\mathrm{ca}}, \ldots, y_n \in [\![\tau_n]\!]_{\mathrm{ca}}) \,.\\
&\qquad [\![c]\!]_0(y_1, \ldots, y_n)\\
&= [\![c]\!]_0 \circ \pi_2 \circ \mathsf{assoc}_{\Gamma,\tau}
\end{aligned}$$

for every constant c of type τ.

- Variables:

$$\frac{\vdash^{\mathrm{ca}} K_1 : \tau_1 \quad \ldots \quad \vdash^{\mathrm{ca}} K_n : \tau_n}{\Gamma \vdash^{\mathrm{ca}} vK_1 \ldots K_n : \rho}\,\mathsf{ca\text{-}var}$$

$$\begin{aligned}
[\![\Gamma \vdash^{\mathrm{ca}} vK_1 \ldots K_n : \rho]\!]_{\mathrm{ca}} &= \lambda(x_1 \in [\![\gamma_1]\!]_{\mathrm{ca}}, \ldots, x_m \in [\![\gamma_m]\!]_{\mathrm{ca}}) \,.\\
&\qquad x_i([\![\vdash^{\mathrm{ca}} K_1 : \tau_1]\!]_{\mathrm{ca}}, \ldots, [\![\vdash^{\mathrm{ca}} K_n : \tau_n]\!]_{\mathrm{ca}})\\
&= (\lambda f \,.\, f([\![\vdash^{\mathrm{ca}} K_1 : \tau_1]\!]_{\mathrm{ca}}, \ldots, [\![\vdash^{\mathrm{ca}} K_n : \tau_n]\!]_{\mathrm{ca}})) \circ \pi_{i;\Gamma}
\end{aligned}$$

provided i is the rightmost position in Γ of an occurrence of v, and $\tau = \gamma_i$.

- Application:

$$\frac{\Gamma \vdash^{\mathrm{ca}} L : \tau_0 \to \tau \quad \Gamma \vdash^{\mathrm{ca}} L_0 : \tau_0}{\Gamma \vdash^{\mathrm{ca}} LL_0 : \tau}\,\mathsf{app}$$

$$\begin{aligned}
[\![\Gamma \vdash^{\mathrm{ca}} LL_0 : \tau]\!]_{\mathrm{ca}} &= \lambda(x_1 \in [\![\gamma_1]\!]_{\mathrm{ca}}, \ldots, x_m \in [\![\gamma_m]\!]_{\mathrm{ca}}, y_1 \in [\![\tau_1]\!]_{\mathrm{ca}}, \ldots, y_n \in [\![\tau_n]\!]_{\mathrm{ca}}) \,.\\
&\qquad [\![\Gamma \vdash^{\mathrm{ca}} L : \tau_0 \to \tau]\!]_{\mathrm{ca}}(x_1, \ldots, x_m, f(x_1, \ldots, x_m), y_1, \ldots, y_n)
\end{aligned}$$

where $\tau_0 = \tau_0^1 \to \cdots \to \tau_0^{n_0} \to \rho_0$ and

$$\begin{aligned}
f &= \lambda(x_1 \in [\![\gamma_1]\!]_{\mathrm{ca}}, \ldots, x_m \in [\![\gamma_m]\!]_{\mathrm{ca}}) \,.\\
&\qquad \lambda(z_1 \in [\![\tau_0^1]\!]_{\mathrm{ca}}, \ldots, z_{n_0} \in [\![\tau_0^{n_0}]\!]_{\mathrm{ca}}) \,.\, [\![\Gamma \vdash^{\mathrm{ca}} L_0 : \tau_0]\!]_{\mathrm{ca}}(x_1, \ldots, x_m, z_1, \ldots, z_{n_0}).
\end{aligned}$$

- Abstraction:

$$\frac{\Gamma, v : \tau \vdash^{\mathrm{ca}} L : \tau'}{\Gamma \vdash^{\mathrm{ca}} (\lambda v : \tau \,.\, L) : \tau \to \tau'}\,\mathsf{abs}$$

$$[\![\Gamma \vdash^{\mathrm{ca}} (\lambda v : \tau \,.\, L) : \tau \to \tau']\!]_{\mathrm{ca}} = [\![\Gamma, v : \tau \vdash^{\mathrm{ca}} L : \tau']\!]_{\mathrm{ca}}$$

5.3.1 Proposition *Every term has a unique ca-PCF type per typing context in which it is typable: if $\Gamma \vdash^{\mathrm{ca}} L : \tau$ and $\Gamma \vdash^{\mathrm{ca}} L : \tau'$ are both derivable then $\tau = \tau'$.*

The function $[\![-]\!]_{\mathrm{ca}}$ is well-defined, and for every derivable judgement $\Gamma \vdash^{\mathrm{ca}} L : \tau$,

$$[\![\Gamma \vdash^{\mathrm{ca}} L : \tau]\!]_{\mathrm{ca}} \in [\![\gamma_1 \to \cdots \to \gamma_m \to \tau]\!]_{\mathrm{ca}},$$

where $\Gamma = v_1 : \gamma_1, \ldots, v_m : \gamma_m$.

Proof: An induction on the structure of a term L shows that L has a unique typing derivation (per type context in which it is typable). Note that if a derivation may end with the variable rule ca-var then it may not end with rule app, so that no ambiguity may arise.

In order to show that the semantic function assigns to terms meanings in the ca-sequential model we rely on the previous development. In general, we use the closure of ca-sequential functions under composition, as well as ca-sequentiality of projections and associative and commutative isomorphisms of the product. In addition, for const we use the assumption that $[\![c]\!]_0 \in [\![\tau]\!]_{\mathrm{ca}}$. For app we rely on preservation of ca-sequentiality by currying, proposition 5.2.3, and ca-sequentiality of mediating morphisms. The abs case is trivial. The interesting case is rule ca-var, where we rely on ca-sequentiality of application to a fixed argument, proposition 5.2.4. ∎

5.3.2 Definability of Isolated Elements

The link between the syntactic restrictions of ca-PCF and the semantic assumptions of the sequential model is formulated as a full abstraction result.

We use notations such as Def_x, Test_x and $\mathrm{Sel}_{r;\eta}$ below to stand for representatives of classes of terms with certain properties. It is important to note that the representatives are *not* determined uniquely, but this notation is convenient for, on the one hand, showing that the class is not empty, or, on the other hand, using an element of the class in some other construction.

We write $(\rho' \to)^k \rho'$ to abbreviate

$$(\rho' \to)^k \rho' = \underbrace{\rho' \to (\cdots \to (\rho'}_{k \text{ times}} \to \rho')).$$

We say that a model $[\![-]\!]$ of a class of types (that includes all ground types) and the ca-terms of those types in the category CA is ***articulate*** iff for every type τ in the class and every ground type ρ,

(1) For every isolated $x \in [\![\tau]\!]$ there exists a closed term Def_x that defines x in the sense that $[\![\vdash^{\mathrm{ca}} \mathrm{Def}_x : \tau]\!] = x$.

(2) For every isolated $x \in [\![\tau]\!]$ and every ground type ρ' there exists a closed term $\mathrm{Test}_x^{\tau,\rho'} : \tau \to (\rho' \to \rho')$ with

$$[\![\vdash^{\mathrm{ca}} \mathrm{Test}_x^{\tau,\rho'} : \tau \to (\rho' \to \rho')]\!] = \lambda(z \in [\![\tau]\!], y \in [\![\rho']\!]) \,.\, \begin{cases} y & \text{if } x \leq z \\ \perp & \text{otherwise.} \end{cases}$$

(3) For every finite $k > 0$, every index r over $[\![\tau]\!]$ with exactly k answers, and every enumeration $\eta : \{1, \ldots, k\} \to \mathbf{Ans}\, r$ of the answers of r, there is a closed term $\mathrm{Sel}_{r;\eta}^{\tau,\rho'}$

such that

$$[\![\vdash^{\mathrm{ca}} \mathrm{Sel}^{\tau,\rho'}_{r;\eta} : \tau \to (\rho' \to)^k \rho']\!] = \lambda(z \in [\![\tau]\!], y_1 \in [\![\rho']\!], \ldots, y_k \in [\![\rho']\!]) \;.\; \begin{cases} y_1 & \text{if } \eta 1 \checkmark z \\ \vdots & \\ y_i & \text{if } \eta i \checkmark z \\ \vdots & \\ y_k & \text{if } \eta k \checkmark z \\ \bot & \text{otherwise.} \end{cases}$$

We call a term $\mathrm{Sel}^{\tau;\rho'}_{r;\eta}$ a ***selector*** for r and $\mathrm{Test}^{\tau;\rho'}_{x}$ a ***tester*** for x. We will drop the type superscripts from these notations. To see that the intended meaning of a selector term is a well defined function, we rely on disjointness of extents of answers of an index, (IS3). Notationally, we may spell out the enumeration of a selector by giving a list of the answers of r, as in $\mathrm{Sel}_{s_1,\ldots,s_k}$, or, if $r = [x \Rightarrow^{\mathrm{ca}} r']$ is a ca-index, by giving an enumeration of the answers of the residual index r', as in $\mathrm{Sel}_{[x \Rightarrow^{\mathrm{ca}} s'_1,\ldots,s'_k]}$. Also, the answers of data indices are singletons, and we will omit the set brackets if listing the elements explicitly.

5.3.2 Proposition *The standard interpretation of booleans and naturals is articulate.*

Proof: We have assumed constants defining all elements of **Bool** and **Nat**.

We can choose for $\mathrm{Sel}_{\mathsf{T},\mathsf{F}} : \mathtt{Bool} \to \rho' \to \rho' \to \rho'$ the constant $\mathtt{if}^{\rho'}$. For other selectors over `Bool` use the obvious variations.

For $\mathrm{Sel}_{x_1,\ldots,x_k} : \mathtt{Nat} \to (\rho' \to)^k \rho'$ take

$$\begin{aligned} \mathrm{Sel}_{x_1,\ldots,x_k} \;=\;& \lambda \mathtt{z} : \mathtt{Nat} \,.\, \lambda \mathtt{y}_1 : \rho' \,.\, \ldots \lambda \mathtt{y}_k : \rho' \,. \\ & \quad \mathtt{if}\ \mathtt{z} = 0\ \mathtt{then}\ M_0\ \mathtt{else} \\ & \quad\quad \mathtt{if}\ \mathtt{z} = 1\ \mathtt{then}\ M_1\ \mathtt{else} \\ & \quad\quad\quad \ldots \\ & \quad\quad\quad\quad \mathtt{if}\ \mathtt{z} = k'\ \mathtt{then}\ M_{k'}\ \mathtt{else} \\ & \quad\quad\quad\quad\quad \Omega \end{aligned}$$

where $k' = \mathsf{max}\,\{x_1, \ldots, x_k\}$, and for $0 \le j \le k'$, $\mathtt{z} = j$ is short for $\mathtt{(=0)((-1)}^j\mathtt{z)}$, and $M_j = \mathtt{y}_i$ if there exists i such that $j = x_i$, and $M_j = \Omega$ otherwise. It is important that the testing of `z` against the values $0, \ldots, k'$ be carried out in increasing order.

For Test_x we can simply take Sel_x, since $\{x\}$ is a data index. ∎

5.3.3 Proposition *If* $[\![-]\!]_0$ *is an articulate model of ground types and it interprets the ground types as data indexed flat domains then* $[\![-]\!]_{\mathrm{ca}}$ *is an articulate model of ca-PCF.*

Proof: By type induction on τ.

- If τ is a ground type then $[\![-]\!]_{\mathrm{ca}}$ coincides with $[\![-]\!]_0$, which is assumed articulate.

- If τ is not ground, assume that $\tau = \tau_1 \to \ldots \to \tau_n \to \rho$, and let f be an isolated sequential function in $[\![\tau]\!]_{\mathrm{ca}}$. Since f is isolated, it is the lub of a finite set of step functions. Choose a minimal set F of step functions such that $f = \bigvee F$, say $F = \{[\bar{x}_i \Rightarrow y_i] \mid i \leq \ell\}$, where each $\bar{x}_i = (x_i^1, \ldots, x_i^n)$. By minimality of ℓ, each of the y_i's is a proper value. Continue now by a secondary induction on ℓ.

  - If $\ell = 0$ then $f = \bot$, so we can let $\mathrm{Def}_\bot = \Omega^\tau = \lambda \mathtt{v}_1 : \tau_1 \ldots . \lambda \mathtt{v}_n : \tau_n \,.\, \Omega^\rho$.
    Testing is trivial in this case, and we set $\mathrm{Test}_\bot = \lambda \mathtt{f} : \tau \,.\, \lambda \mathtt{v} : \rho' \,.\, \mathtt{v}$.

  - If $\ell = 1$ then $f = [(x_1^1, \ldots, x_1^n) \Rightarrow y_1]$. By the induction hypothesis for smaller types there are closed terms Def_{y_1} and testers $\mathrm{Test}_{x_1^j}$ for each j. We can take

    $$\mathrm{Def}_f = \lambda \mathtt{v}_1 : \tau_1 \ldots . \lambda \mathtt{v}_n : \tau_n \,.\, \mathrm{Test}_{x_1^1} \mathtt{v}_1 (\ldots (\mathrm{Test}_{x_1^n} \mathtt{v}_n (\mathrm{Def}_{y_1})) \ldots).$$

    The testing problem for $\ell = 1$ is solved by reducing it to a testing problem over $[\![\rho]\!]_{\mathrm{ca}}$, using the induction hypothesis for existence of $\mathrm{Def}_{x_1^i}$:

    $$\mathrm{Test}_f = \lambda \mathtt{f} : \tau \,.\, \mathtt{v} : \rho' \,.\, \mathrm{Test}_y (\mathtt{f} \mathrm{Def}_{x_1^1} \ldots \mathrm{Def}_{x_1^n}) \mathtt{v}.$$

  - If $\ell > 1$, let $X = \{\bar{x}_i \mid i \leq \ell\}$. Since $[\![\rho]\!]_{\mathrm{ca}}$ is a flat domain and $\ell > 1$ is minimal, it must be the case that $f(\wedge X) = \bot$ and $f(\wedge X) \notin \uparrow fX$. Thus, there is an index $r' \in \mathcal{I}(f; \wedge X, X)$. By ca-sequentiality of f, there must therefore exist an index $r_0 \in \mathcal{I}(\wedge X, X)$ over the product $[\![\tau_1]\!]_{\mathrm{ca}} \times \cdots \times [\![\tau_n]\!]_{\mathrm{ca}}$, derived from an index in one of the components; assume without loss of generality that it is derived from an index in the m'th component, so that there is an $r \in \mathcal{I}(\wedge(\pi_m X), \pi_m X)$. This implies that $\pi_m X$ has no least element, and it has at least two elements, and as many as ℓ. Let $Y = \pi_m X$.

    If we take r minimal with respect to number of answers, it will have at most ℓ answers, one per element of Y, since we can discard answers that contain no elements of Y. More importantly, by stability of the indices, r will have at least 2 answers. To see this, note that if the extent of an answer data index contains Y then it must contain $\wedge Y$, since it is a single lobe that, being stable, is closed under meet (proposition 2.2.9). Similarly, if $r = [x \Rightarrow^{\mathrm{ca}} r']$ is an answer ca-index whose extent contains Y then the extent of r' contains Yx, and by the above also $(\wedge Y)x = \wedge(Yx)$, so that $\wedge Y$ fills r.

    Let k be the number of answers of r, $2 \leq k \leq \ell$, and let $s_1, \ldots, s_k$ be an enumeration of these answers. By induction hypothesis, a selector for r is definable, and this now lets us split F into corresponding collections of step functions, each with fewer than ℓ elements. More formally, for $j \leq k$, let $f_j = \bigvee F_j$, where $F_j = \{[\bar{x}_i \Rightarrow y_i] \mid i \leq \ell \;\&\; s_j \checkmark x_i^m\}$. Since each f_j is the lub of fewer than ℓ step functions, it is definable, by the secondary induction hypothesis. We are now able to define f:

    $$\mathrm{Def}_f = \lambda \mathtt{v}_1 : \tau_1 \ldots . \lambda \mathtt{v}_n : \tau_n \,.\, \mathrm{Sel}_{s_1, \ldots s_k} \mathtt{v}_m (\mathrm{Def}_{f_1} \mathtt{v}_1 \ldots \mathtt{v}_n) \ldots (\mathrm{Def}_{f_k} \mathtt{v}_1 \ldots \mathtt{v}_n).$$

The tester in this case may be obtained by the conjunction of the separate testers for f_j or, as we define it here, the testers for the individual step functions, as defined in the case $\ell = 1$ above:

$$\mathrm{Test}_f = \lambda \mathtt{f} : \tau \,.\, \mathtt{v} : \rho' \,.\, \mathrm{Test}_{[\bar{x}_1 \Rightarrow y_1]}\mathtt{f}(\ldots(\mathrm{Test}_{[\bar{x}_\ell \Rightarrow y_\ell]}\mathtt{f}\mathtt{v})).$$

We now show definability of $\mathrm{Sel}_{[\bar{x} \Rightarrow^{\mathrm{ca}} s'_1, \ldots, s'_k]}$ in the functional case. This is done by transforming the selection problem in the function space into a selection problem in the ground case, which has already been solved. Note that $\bar{x} \in [\![\tau_1]\!]_{\mathrm{ca}} \times \cdots \times [\![\tau_n]\!]_{\mathrm{ca}}$, so we can set $\bar{x} = (x_1, \ldots, x_n)$, and by induction hypothesis each of the isolated elements x_i is definable by some Def_{x_i}. We thus obtain:

$$\begin{aligned} \mathrm{Sel}_{[\bar{x} \Rightarrow^{\mathrm{ca}} s'_1, \ldots, s'_k]} \;=\; & \lambda \mathtt{f} : \tau \,.\, \lambda \mathtt{v}_1 : \rho' \,.\, \ldots \lambda \mathtt{v}_k : \rho' \,. \\ & \mathrm{Sel}_{s'_1, \ldots, s'_k}(\mathtt{f}\,\mathrm{Def}_{x_1} \ldots \mathrm{Def}_{x_n})\mathtt{v}_1 \ldots \mathtt{v}_k. \end{aligned}$$

Note that `f` is applied to closed arguments, so this is a valid term. ∎

5.3.4 Corollary *The ca-model with the standard interpretation of ground types is articulate, and the isolated elements of the interpretations of all types are definable.*

The essential difference between this definability proof and Plotkin's proof for the parallel extension of PCF [Plo77, lemma 4.5] is in the synthesis of the defining term for arrow type with $l > 1$, in the above terminology. Plotkin's proof uses the parallel conditional facility to combine a defining term for the lub of l step functions with an additional step function to obtain a defining term for the lub of $l + 1$ step functions; we rely instead on the existence of an index that partitions the set of step functions into smaller sets.

Full abstraction — both inequational and equational — follows by standard arguments from the definability of isolated elements.

5.3.5 Proposition *The semantics $[\![-]\!]_{\mathrm{ca}}$ is inequationally fully abstract with respect to itself as a notion of program behavior. That is, for all pairs of terms L and L', it is the case that*

$$[\![\Gamma \vdash^{\mathrm{ca}} L : \tau]\!]_{\mathrm{ca}} \leq [\![\Gamma \vdash^{\mathrm{ca}} L' : \tau]\!]_{\mathrm{ca}}$$

for every Γ and τ for which the judgements are derivable iff for every program context $P[-]$ for which both L and L' are compatible (with type ρ),

$$[\![\vdash^{\mathrm{ca}} P[L] : \rho]\!]_{\mathrm{ca}} \leq [\![\vdash^{\mathrm{ca}} P[L'] : \rho]\!]_{\mathrm{ca}}.$$

To link up this result with the standard notion of behavior for PCF programs, we verify that $[\![-]\!]_{\mathrm{ca}}$ agrees with the usual operational semantics for PCF, as presented in [Plo77].

5.3.6 Proposition *The program behaviors induced by the semantics $[\![-]\!]_{\mathrm{ca}}$ and the operational semantics coincide. That is, for every closed term P of ground type ρ, $[\![\vdash^{\mathrm{ca}} P : \rho]\!]_{\mathrm{ca}} = x \neq \bot$ iff P evaluates to (the constant denoting) x, and $[\![\vdash^{\mathrm{ca}} P : \rho]\!]_{\mathrm{ca}} = \bot$ iff the evaluation of P diverges.*

Proof: Details omitted. ∎

In summary, the semantics $[\![-]\!]_{\mathrm{ca}}$ is fully abstract for ca-PCF with respect to the usual notion of program behavior.

5.3.7 Remark The sub-language ca-PCF is, of course, less expressive then PCF. It cannot define application, for instance, or functions that are only definable using imbrication such as $[\mathsf{lor} \Rightarrow \mathsf{T}] \vee [\mathsf{ror} \Rightarrow \mathsf{T}]$. Nevertheless, we feel justified in asserting that it is not a completely trivial language. It could be interesting to think of ways to assess its expressiveness other than these coarse qualitative extremes, although it is not clear what can be expected.

A serious deficiency of ca-PCF is the absence of the fixpoint operator. The fixpoint functional itself is not ca-sequential, but instead of adding it to the language, one may consider adding a μ-binding construct, with essentially the same effect, such that $\mu \mathtt{f} : \tau \,.\, M\mathtt{f}$ is equivalent to $\mathsf{fix}\, M$. However, in order to obtain the desired expressiveness, we would need a way of circumventing the restriction on term formation. If we differentiate between the syntactic classes of variables, bound by λ, and μ-identifiers, bound by μ, we could allow μ-identifiers to be applied to terms that are not necessarily closed. This approach might be workable, but it is somewhat ad-hoc, and we do not study it further. •

5.3.3 Beyond Flat Domains

Our proof of definability, following Plotkin's proof for the parallel extension of PCF with respect to the continuous functions model, assumes that ground types are interpreted as flat domains. If we look at the use made of this fact in the proof, we find that it was used to guarantee the existence of an index separating a non-empty finite set of proper elements from $\perp$.

This suggests a way to discard the flatness assumption; a ground type could be interpreted by a much more complex domain, as long as an index over it were supplied as part of the computational process, serving as r' in the proof. In essence, the computation would be driven by queries over its output. This is very close to the model of computation for Berry and Curien's language CDS0 [BC85].

Under such an operational discipline, it should be possible to extend together the language and the model to include non-flat ground types. To obtain closure under function space, we should restrict ourselves at this point to tree domains. An example of a candidate for such an extension is the domain of lazy naturals (figure 2.2 on page 14), a tree domain.

Chapter 6

Bi-ordered Structures

The use of the stable order over the stable function space leads to a cartesian closed category and a stable functions model of PCF, but it has its drawbacks. Chief among them is the introduction of "freak" functionals that are present in the stable functions model, but are not even monotone with respect to the pointwise order on the function space. An example is the functional

$$[[\bot \Rightarrow \top] \Rightarrow \mathsf{T}] \vee^{\mathrm{p}} [[\top \Rightarrow \top] \Rightarrow \mathsf{F}] : (\mathbf{2} \to \mathbf{2}) \to \mathbf{Bool}.$$

The stable functions model is therefore not a true restriction of the continuous functions model.

This has led Berry to study bi-ordered structures — sets equipped with *two* partial orders. The function space may thus carry with it both the pointwise and the stable orders, which may both serve to restrict functionals as appropriate. The example above, for instance, would be eliminated. He was successful in obtaining a CCC of stable functions between bidomains, and a corresponding model.

Berry's work on stable functions on bidomains is reported in French in his thesis [Ber79], and, to my knowledge, there is only one account available in English [Ber78], and it is quite preliminary and lacking in detail. For this reason, the first two sections of this chapter are devoted to a detailed presentation of Berry's development, following closely Berry's original work. The few differences or new concepts introduced are pointed out, but in all other respects the credit for this work is Berry's (but of course I am responsible for any errors that may be present here).

Following an exposition of bidomains and stable functions between them, the third section of this chapter extends the framework of indexed computation to bidomains. We define indexed bidomains, and indexable functions between them. The next chapter will define specific index structures over bidomains.

Finally, the last section of this chapter points out how bidomains may be seen to arise from a restriction of the primitives available for computation — a lack of expressiveness of the index structures, if you will.

6.1 Bidcpos

A ***biposet*** $C = (A, \leq^\bullet, \leq^\star)$ is a set A together with two partial orders $\leq^\bullet$ and $\leq^\star$ on A. We write $C^\bullet$ and $C^\star$ for the posets $(A, \leq^\bullet)$ and $(A, \leq^\star)$, respectively. We may also use the symbols $\bullet$ and $\star$ to indicate which of the orders is being referred to. Thus, for instance, x and y are $\star$-consistent, or $x \Uparrow^\star y$, iff x and y have an upper bound in $C^\star$, which may be referred to as an $\star$-upper bound.

A ***bidcpo*** is a biposet $C = (A, \leq^\bullet, \leq^\star)$ such that

(BD1) $C^\bullet = (A, \leq^\bullet)$ is a dcpo with continuous meet $\wedge^\bullet : C^\bullet \times C^\bullet \to C^\bullet$.

(BD2) $C^\star = (A, \leq^\star)$ is a pointed dcpo, and the identity function $\mathsf{id} : C^\star \to C^\bullet$ is continuous, *i.e.*,

(BD2.1) $x \leq^\star y$ implies $x \leq^\bullet y$ for every $x, y \in C$.

(BD2.2) $\bigvee^\star X = \bigvee^\bullet X$ for every $\star$-directed $X \subseteq C$.

(BD3) $\wedge^\bullet$ is $\star$-monotone: if $x \leq^\star x'$ and $y \leq^\star y'$ then $x \wedge^\bullet y \leq^\star x' \wedge^\bullet y'$.

(BD4) For every $\bullet$-directed X and X', if for every $x \in X$ and $x' \in X'$ there exist $y \in X$ and $y' \in X'$ such that $x \leq^\bullet y$, $x' \leq^\bullet y'$ and $y \leq^\star y'$, then $\bigvee^\bullet X \leq^\star \bigvee^\bullet X'$.

A bidcpo is ***distributive***, or a ***d-bidcpo*** for short, if in addition

(BDd) $C^\star$ is bounded-complete and distributive, and the identity function $\mathsf{id} : C^\star \to C^\bullet$ preserves consistent joins, *i.e.*, $x \vee^\star y = x \vee^\bullet y$ for every $x, y \in C$ such that $x \Uparrow^\star y$.

The orders $\leq^\bullet$ and $\leq^\star$ may be thought of abstractly, but we may also refer to $\leq^\bullet$ as the *approximation* order (or *extensional* or *observable* order), and to $\leq^\star$ as the *information* (or *intensional* or *intrinsic*) order. This reflects the view that $\leq^\star$ is some intrinsic information order on the underlying set, and $\leq^\bullet$ reflects the (limited) ability of the computation to approximate that order, using only the (limited) set of primitives at its disposal[1] . We will further justify this view in section 6.4.

Bidcpos may be portrayed by Hasse diagrams with solid lines representing the information order $\leq^\star$ and both solid and dotted lines representing the approximation order $\leq^\bullet$.

6.1.1 Example Figure 6.1 is a diagram of $(\mathbf{2} \to \mathbf{2}, \leq^{\mathrm{p}}, \leq^{\mathrm{s}})$, the space of monotone (and continuous and stable) functions from $\mathbf{2}$ to $\mathbf{2}$, bi-ordered by the pointwise order (dotted and solid lines) and the stable order (solid lines). $\bullet$

Property (BD4) is equivalent to a simpler property when $C^\bullet$ is ω-algebraic. A biposet $C = (A, \leq^\bullet, \leq^\star)$ with $C^\bullet$ a dcpo has property (BD4$'$) iff

(BD4$'$) The relation $\leq^\star$ is $\leq^\bullet$-co-inductive. That is, if $x_1, x_2, \ldots, x_n, \ldots$ and $y_1, y_2, \ldots, y_n, \ldots$ are $\bullet$-increasing chains such that $x_i \leq^\star y_i$ for all i then $\bigvee^\bullet x_i \leq^\star \bigvee^\bullet y_i$.

[1] The choice of the symbols is for their mnemonic value when considering the orders to be employed over the bi-ordered function space: $\leq^\bullet$ will be taken to be the pointwise order, hence a point $\bullet$, and $\leq^\star$ will be taken to be the *sta*ble order, hence a *sta*r $\star$.

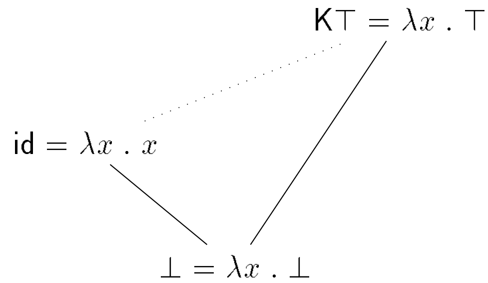

Figure 6.1: The function space $\mathbf{2} \to \mathbf{2}$, bi-ordered pointwise and stably

6.1.2 Proposition *Let $C = (A, \leq^\bullet, \leq^\star)$ be a biposet where $C^\bullet$ is a dcpo. If C has property (BD4) then it has property (BD4$'$). If, moreover, $C^\bullet$ is ω-algebraic and C has property (BD4$'$) then it has property (BD4).*

Proof: If C has (BD4) and $X = x_1, x_2, \ldots$ and $Y = y_1, y_2, \ldots$ are $\bullet$-increasing chains such that $x_i \leq^\star y_i$ for all i then X and Y are directed sets related as required by (BD4), because for every x_i and y_j, $x_i \leq^\bullet x_k$, $y_j \leq^\bullet y_k$, and $x_k \leq^\star y_k$, where $k = \mathsf{max}\{i, j\}$. Thus, by (BD4), $\bigvee^\bullet X \leq^\star \bigvee^\bullet Y$, so that (BD4$'$) holds.

Conversely, assume that (BD4$'$) holds for C, and $C^\bullet$ is ω-algebraic with some enumeration $z_1, z_2, \ldots$ of its isolated elements. Let X and Y be directed sets related as required by (BD4).

Let X' be an $\bullet$-increasing sequence formed by an enumeration of a subset of X, and that is co-final with X: $\bigvee^\bullet X' = \bigvee^\bullet X$. X' may be formed by taking x'_1 to be some arbitrary element of x, and for every i, taking x'_{i+1} to be an element of X that dominates x_i and is an upper bound of $\{z_j \mid z_j \leq^\bullet \bigvee^\bullet X \;\;\&\;\; j \leq i\}$. Y' may be formed analogously for Y.

Now form X'' and Y'' as co-final sub-sequences of X and Y respectively such that $x''_i \leq^\star y''_i$ for every i. By assumption, there exist $x''_1 \in X$ and y''_1 such that $x'_1 \leq^\bullet x''_1$, $y'_1 \leq^\bullet y''_1$, and $x''_1 \leq^\star y''_1$. Now, for every i, it is possible by assumption to choose x''_{i+1} and y''_{i+1} such that $x''_i, x'_i \leq^\bullet x''_{i+1}$ (passing first through an upper bound in X of x''_i and x'_i), $y''_i, y'_i \leq^\bullet y''_{i+1}$, and $x''_{i+1} \leq^\star y''_{i+1}$. By (BD4$'$) it now follows that $\bigvee^\bullet X = \bigvee^\bullet X'' \leq^\star \bigvee^\bullet Y'' = \bigvee^\bullet Y$. ∎

A bidcpo may trivially be obtained by replicating the order of a suitable dcpo. A bidcpo is ***trivial*** if its two orders coincide.

6.1.3 Proposition *If $(A, \leq)$ is a pointed dcpo with continuous meet then $(A, \leq, \leq)$ is a bidcpo. If, moreover, $(A, \leq)$ is bounded-complete and distributive than $(A, \leq, \leq)$ is a d-bidcpo.*

Proof: Immediate from definitions. ∎

Consistent meets in $C^\star$ exist and are obtained as the meets in $C^\bullet$.

6.1.4 Proposition *If $C = (A, \leq^\bullet, \leq^\star)$ is a bidcpo and $x, y \in C$ are $\star$-consistent then they have a $\star$-meet, which is equal to their $\bullet$-meet: $x \wedge^\bullet y = x \wedge^\star y$.*

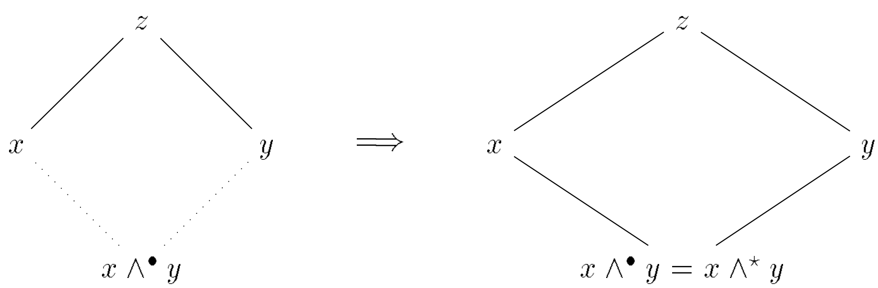

Figure 6.2: $\star$-consistent $\star$-meets are obtained as $\bullet$-meets.

Figure 6.2 is a graphical representation of the proposition.

Proof: Let $x \Uparrow^\star y$, with z a $\star$-upper bound, which is also a $\bullet$-upper bound (BD2.1). By $\star$-monotonicity of $\wedge^\bullet$ (BD3),

$$\begin{array}{lcrcl} x \wedge^\bullet y & \leq^\star & x \wedge^\bullet z & = & x, \\ x \wedge^\bullet y & \leq^\star & z \wedge^\bullet y & = & y, \end{array}$$

and $x \wedge^\bullet y$ is a $\star$-lower bound of x and y. If w is another $\star$-lower bound of x and y then, again by (BD3),

$$w = w \wedge^\bullet w \leq^\star x \wedge^\bullet y.$$

Thus $x \wedge^\bullet y$ is in fact the $\star$-meet of x and y. ■

The next proposition summarizes some further properties of bidcpos:

6.1.5 Proposition *For a bidcpo C,*

(1) $C^\bullet$ is pointed, and $\bot_{C^\bullet} = \bot_{C^\star}$.

(2) If $x \leq^\bullet y$ and $x \Uparrow^\star y$ then $x \leq^\star y$.

(3) For every $z \in C$, $\leq^\bullet$ and $\leq^\star$ coincide when restricted to $\downarrow^\star z$.

(4) $\mathbf{K}\, C^\bullet \subseteq \mathbf{K}\, C^\star$.

Proof:

(1) $C^\star$ has a least element $\bot_{C^\star}$ by (BD2). By (BD2.1) this is also a least element of $C^\bullet$.

(2) This is a special case of 6.1.4; if $x \leq^\bullet y$ and $x \Uparrow^\star y$ then $x = x \wedge^\bullet y = x \wedge^\star y$, so that $x \leq^\star y$.

Figure 6.3 depicts this, a special case of figure 6.2.

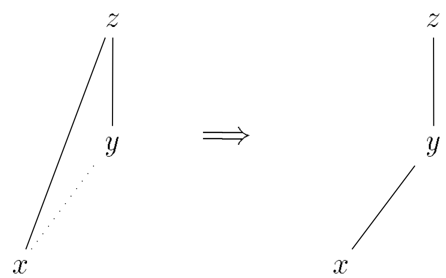

Figure 6.3: Proposition 6.1.5 (2)

(3) For every x and y $\star$-dominated by z, $x \leq^\bullet y$ iff $x \leq^\star y$ by the conjunction of (2) and (BD2.1).

(4) Assume that x is isolated in $C^\bullet$, and $x \leq^\star \bigvee^\star Y$ for some $\star$-directed Y. By (BD2.1) and (BD2.2), Y is $\bullet$-directed and $x \leq^\bullet \bigvee^\star Y = \bigvee^\bullet Y$, and by $\bullet$-isolatedness of x there exists $y \in Y$ with $x \leq^\bullet y$. But $\bigvee^\star Y$ is a $\star$-upper bound on x and y, so that $x \leq^\star y$, as desired to show that x is $\star$-isolated. ∎

The restrictions imposed on biposets by the bidcpo and d-bidcpo axioms are quite substantial. The two orders of a bidcpo coincide completely when restricted to $\downarrow^\star x$, for every point x. If $C^\star$ has a maximum element then $C^\star$ and $C^\bullet$ coincide completely. The least elements of $C^\bullet$ and $C^\star$ coincide, and we may unambiguously write $\bot_C$ or $\bot$ for $\bot_{C^\bullet} = \bot_{C^\star}$, and call this the least or bottom element of the bidcpo.

The product of biposets is taken to be cartesian product of the underlying sets, bi-ordered by the componentwise approximation order and the componentwise information order.

6.1.6 Proposition *Bidcpos and d-bidcpos are closed under products.*

Proof: Follows from development for the uni-ordered case, and the fact that orders and limits are determined componentwise. Details omitted. ∎

6.1.1 Function Spaces over Bidcpos

We consider now spaces of functions over bidcpos, concentrating on stable functions. Again, the development follows closely Berry's lead, except for the new concept of *admissibility* that is defined here to improve the packaging of the presentation, so that it may be employed elsewhere.

Stable Functions on Bidcpos

Properties like monotonicity and continuity of functions may be qualified with $\bullet$ or $\star$ to indicate the orders used to determine the property. For instance, $f : C \to C'$, fundamentally a function between the underlying sets of the bidcpos, is said to be $\bullet$-continuous iff f is a continuous function between $C^\bullet$ and $C'^\bullet$.

6.1.7 Proposition *If f is a function between bidcpos C and C' that is $\bullet$-continuous and $\star$-monotone then f is $\star$-continuous.*

Proof: Let $X \subseteq C$ be $\star$-directed. By $\star$-monotonicity of f, fX is also $\star$-directed, and by (BD2.1) X and fX are also $\bullet$-directed. Therefore,

$$f(\bigvee^{\star} X) = f(\bigvee^{\bullet} X) = \bigvee^{\bullet}(fX) = \bigvee^{\star}(fX).$$

by $\bullet$-continuity of f and a double application of (BD2.2). ∎

6.1.8 Corollary *The $\bullet$-meet $\wedge^{\bullet}$ of a bidcpo is $\star$-continuous.*

Proof: Follows from (BD3) and 6.1.7. ∎

A function $f : C \to C'$ between bidcpos is ***bicontinuous*** iff it is both $\bullet$-continuous and $\star$-continuous. f is ***stable*** iff it is $\bullet$-continuous and $\star$-stable. According to our conventions, f is $\star$-stable if it is $\star$-continuous and

$$\forall x, y \in C \,.\, x \Uparrow^{\star} y \implies f(x \wedge^{\star} y) = fx \wedge^{\star} fy.$$

If f is already known or assumed to be $\bullet$-continuous then the requirement for $\star$-continuity of f may be weakened to $\star$-monotonicity of f, by 6.1.7. If f is $\star$-monotone then the $\star$-meets taken here coincide with the $\bullet$-meets, by 6.1.4, so that the preservation of consistent meets may be equivalently phrased as

$$\forall x, y \in C \,.\, x \Uparrow^{\star} y \implies f(x \wedge^{\bullet} y) = fx \wedge^{\bullet} fy.$$

It is important to keep in mind the distinction between stability of functions between posets and stability of functions between biposets. Following Berry, we rely on the context or on explicit qualification to designate the intended notion of stability.

From this point we restrict our attention to d-bidcpos. Let BST be the category of d-bidcpos and stable functions. The identity on a bidcpo is obviously stable, and stable functions are closed under composition. Note that a proof of this is representative of the way in which many of the proofs in this development rely on the previous development for the uni-ordered case: Continuous and stable functions between dcpos (with consistent meets) are closed under composition, so that $\bullet$-continuous and $\star$-stable functions between bidcpos are closed under composition, and thus stable functions between bidcpos are closed under composition. Similar considerations show stability of product morphisms — projections, and the pairing of stable functions, taking the bidcpo product described above. This proves the following:

6.1.9 Proposition *The category* BST *of d-bidcpos and stable functions is cartesian.*

Ordering Functions between Bidcpos

When talking of, *e.g.*, the stable ordering on functions $f, g : C \to C'$ between bidcpos it is necessary to indicate which of the orders is employed on C and C'. To do that, we introduce the following notation:

- f is ***•-pointwise*** dominated by g, written as $f \leq^{\bullet\mathrm{p}} g$, iff for every $x \in C$, $fx \leq^{\bullet}_{C'} gx$. This is the pointwise order on the function space $C \to C'^{\bullet}$.

- f is ***⋆-pointwise*** dominated by g, written as $f \leq^{\star\mathrm{p}} g$, iff for every $x \in C$, $fx \leq^{\star}_{C'} gx$. This is the pointwise order on the function space $C \to C'^{\star}$.

  By (BD2.1) for C', $f \leq^{\star\mathrm{p}} g$ implies that $f \leq^{\bullet\mathrm{p}} g$.

- f is ***⋆-stably*** dominated by g, written as $f \leq^{\star\mathrm{s}} g$, iff $f \leq^{\star\mathrm{p}} g$ and for $x \Uparrow^{\star} y$, $fx \wedge^{\star} gy = fy \wedge^{\star} gx$, or, equivalently, $f \leq^{\star\mathrm{p}} g$ and for $x \leq^{\star} y$, $fx = fy \wedge^{\star} gx$. This is the stable order on the function space $C^{\star} \to C'^{\star}$.

  Again, since the •-meet and the ⋆-meet coincide on ⋆-bounded elements, this definition can be equivalently stated with the •-meet substituting for the ⋆-meet. This is in fact the more common form in the proofs below.

Exponentiation and Admissibility

The ***stable exponentiation*** $C \to_{\mathrm{bs}} C'$ of d-bidcpos C and C', denoted $(C \to_{\mathrm{bs}} C', \leq^{\bullet\mathrm{p}}, \leq^{\star\mathrm{s}})$, is defined to be the space of stable functions from C to C', bi-ordered by $\leq^{\bullet\mathrm{p}}$ and $\leq^{\star\mathrm{s}}$. In other words, $\leq^{\bullet}$ and $\leq^{\star}$ over the stable function space will be taken to be $\leq^{\bullet\mathrm{p}}$ and $\leq^{\star\mathrm{s}}$, respectively. As usual, we use here $C \to_{\mathrm{bs}} C'$ both as a biposet and its underlying set.

We now need to show that the stable exponentiation is in fact a d-bidcpo. We package Berry's proof in two separate parts, bridged by a new definition of an *admissible* subset of the stable function space, in preparation for the consideration of more restricted notions of functions.

First, we look at some limits of interest in the stable exponentiation, and show that they exist and are taken •-pointwise.

6.1.10 Proposition *For d-bidcpos C and C', the stable exponentiation $C \to_{\mathrm{bs}} C'$ has •-meets and •-directed lubs, both determined •-pointwise, and ⋆-consistent ⋆-joins, also determined •-pointwise.*

Proof: Let F be a •-pointwise directed subset of the stable function space between bidcpos C and C'. Let f_0 be the •-pointwise lub of F, defined for every $x \in C$ by $f_0 x = \bigvee^{\bullet} \{fx \mid f \in F\}$. f_0 is •-continuous and ⋆-stable by 2.1.7. f_0 is thus stable, and it is clearly the lub of F in $C \to_{\mathrm{bs}} C'$.

- Let f_1 and f_2 be two stable functions, and let f_0 be their •-pointwise meet, *i.e.*, $f_0 x = (f_1 x) \wedge^{\bullet} (f_2 x)$ for $x \in C$. f_0 is •-continuous and ⋆-stable by 2.1.8, so that it is stable, and it is clearly the •-meet of f_1 and f_2 in $C \to_{\mathrm{bs}} C'$.

- Let $f_1 \Uparrow^{\star\mathrm{s}} f_2$ be two stable functions. By 2.1.12, this implies that $f_1 \Uparrow^{\star\mathrm{p}} f_2$, *i.e.*, for every $x \in C$, $f_1 x \Uparrow^\star f_2 x$, and, moreover, for every $x \Uparrow^\star y$, $f_1 x \wedge^\star f_2 y = f_1 y \wedge^\star f_2 x$. This further implies that for $x \Uparrow^\star y$, using stability of f_1,

$$\begin{array}{rcl} f_1 x \wedge^\star f_2 y = f_1 y \wedge^\star f_2 x & = & f_1 x \wedge^\star f_2 y \wedge^\star f_1 y \wedge^\star f_2 x \\ & \leq^\star & f_1 x \wedge^\star f_1 y \\ & = & f_1 (x \wedge^\star y). \end{array} \tag{6.1}$$

Let f_0 be the $\star$-pointwise join of f_1 and f_2, *i.e.*, for every $x \in C$, $f_0 x = f_1 x \vee^\star f_2 x$. f_0 is well defined and $\star$-continuous by 2.1.7. By (BDd) for C', f_0 is equal to the $\bullet$-pointwise join of f_1 and f_2, and it is $\bullet$-continuous, again by 2.1.7. To prove stability of f_0, let $x \Uparrow^\star y$, and then

$$\begin{array}{rcll} f_0 x \wedge^\star f_0 y & = & (f_1 x \vee^\star f_2 x) \wedge^\star (f_1 y \vee^\star f_2 y) & \text{(Distributivity of } C'^\star) \\ & = & (f_1 x \wedge^\star f_1 y) \vee^\star (f_1 x \wedge^\star f_2 y) \vee^\star & \text{(6.1 and stability of } f_1, f_2) \\ & & \quad (f_2 x \wedge^\star_1 y) \vee^\star (f_2 x \wedge^\star f_2 y) & \\ & = & f_1 (x \wedge^\star y) \vee^\star f_2 (x \wedge^\star y) & \\ & = & f_0 (x \wedge^\bullet y). & \end{array}$$

f_0 $\star$-pointwise dominates f_1 and f_2. To show that it $\star$-stably dominates them, let $x \leq^\star y$, and

$$\begin{array}{rcll} f_1 y \wedge^\star f_0 x & = & f_1 y \wedge^\star (f_1 x \vee^\star f_2 x) & \\ & = & (f_1 y \wedge^\star f_1 x) \vee^\star (f_1 y \wedge^\star f_2 x) & (6.1) \\ & = & f_1 (x \wedge^\star y) & \\ & = & f_1 x. & \end{array}$$

Hence $f_1 \leq^{\star\mathrm{s}} f_0$, and $f_2 \leq^{\star\mathrm{s}} f_0$ is shown similarly. To show that f_0 is indeed the $\star$-stable join of f_1 and f_2, assume that f is a $\star$-stable upper-bound of f_1 and f_2. Then f $\star$-pointwise dominates f_0 by definition, and, if $x \leq^\star y$ then

$$\begin{array}{rcl} f_0 y \wedge^\star f x & = & (f_1 y \vee^\star f_2 y) \wedge^\star f x \\ & = & (f_1 y \wedge^\star f x) \vee^\star (f_2 y \wedge^\star f x) \\ & = & f_1 x \vee^\star f_2 x \\ & = & f_0 x, \end{array}$$

so that f_0 is indeed the $\star$-stable join, as desired, as well as the $\bullet$-pointwise join. ∎

The proposition turns out to characterize properties of subsets of the stable function space that are sufficient to establish that the subset in question is a d-bidcpo when bi-ordered $\bullet$-pointwise and $\star$-stably. We encapsulate this in the definition of admissibility: A subset $F \subseteq C \to_{\mathrm{bs}} C'$ of the stable function space between d-bidcpos is ***admissible*** iff it includes the constant bottom function and it is closed under $\bullet$-meets, $\bullet$-directed lubs, and $\star$-consistent $\star$-joins in the stable function space. By the proposition above, the stable function space itself is admissible and, moreover, $\bullet$-meets, $\bullet$-directed lubs and $\star$-consistent $\star$-joins are all determined $\bullet$-pointwise in an admissible subset of the stable function space.

We may now generalize the stable exponentiation. If $\mathcal{C}$ is a subcategory of Bst, *i.e.*, with (some sub-class of) d-bidcpos as objects, and with a subset of the stable functions between d-bidcpos for morphisms, we let the ***$\mathcal{C}$-exponentiation*** $C \to_{\mathcal{C}} C'$ of objects C and C' be $(C \to_{\mathcal{C}} C', \leq^{\bullet \mathrm{p}}, \leq^{\star \mathrm{s}})$, the set of $\mathcal{C}$-functions from C to C', bi-ordered by $\leq^{\bullet \mathrm{p}}$ and $\leq^{\star \mathrm{s}}$. The stable exponentiation, then, is just the Bst-exponentiation. We say that $\mathcal{C}$ is an ***admissible*** subcategory of Bst iff for every two $\mathcal{C}$-objects C and C', $C \to_{\mathcal{C}} C'$ is an admissible subset of $C \to_{\mathrm{bs}} C'$; that is, it contains the constant bottom function, and $(C \to_{\mathcal{C}} C', \leq^{\bullet \mathrm{p}})$ has meets and directed lubs, both determined $\bullet$-pointwise, and $(C \to_{\mathcal{C}} C', \leq^{\star \mathrm{s}})$ has consistent joins, determined $\bullet$-pointwise.

The significance of this definition is captured by the following proposition:

6.1.11 Proposition *If $\mathcal{C}$ is an admissible subcategory of* Bst *then the $\mathcal{C}$-exponentiation of every two $\mathcal{C}$-objects C and C' is a d-bidcpo.*

Proof: Let C and C' be $\mathcal{C}$-objects, so they are d-bidcpos. Let $C \to_{\mathcal{C}} C'$ be the set of $\mathcal{C}$-functions between them, which are all stable.

(BD1) By assumption, $(C \to_{\mathcal{C}} C', \leq^{\bullet \mathrm{p}})$ is a dcpo with meets. Continuity of the $\bullet$-pointwise meet for the exponentiation is directly inherited from continuity of the $\bullet$-meet for C': if $F_1, F_2 \subseteq C \to_{\mathcal{C}} C'$ are $\bullet$-pointwise directed sets then

$$\begin{aligned}
(\bigvee^{\bullet} F_1 \wedge^{\bullet} \bigvee^{\bullet} F_2)x &= (\bigvee^{\bullet} \{fx \mid f \in F_1\}) \wedge^{\bullet} (\bigvee^{\bullet} \{fx \mid f \in F_2\}) \\
&\qquad \text{(;Meets and directed lubs taken $\bullet$-pointwise)} \\
&= \bigvee^{\bullet} \{f_1 x \wedge^{\bullet} f_2 x \mid f_1 \in F_1 \;\;\&\;\; f_2 \in F_2\} \\
&\qquad \text{(;Continuity of $\wedge^{\bullet}_{C'}$)} \\
&= (\bigvee^{\bullet} \{f_1 \wedge^{\bullet} f_2 \mid f_1 \in F_1 \;\;\&\;\; f_2 \in F_2\})x \\
&\qquad \text{(;Meets and directed lubs taken $\bullet$-pointwise).}
\end{aligned}$$

This establishes (BD1) for $(C \to_{\mathcal{C}} C', \leq^{\bullet \mathrm{p}}, \leq^{\star \mathrm{s}})$.

(BD2) The constant bottom function $\mathsf{K}\bot_{C'}$ is the least element of $(C \to_{\mathcal{C}} C', \leq^{\star \mathrm{s}})$. Clearly, if $f \leq^{\star \mathrm{s}} g$ then $f \leq^{\bullet \mathrm{p}} g$, so that (BD2.1) holds. If $F \subseteq C \to_{\mathcal{C}} C'$ is a $\star$-stably directed set then it is also $\bullet$-pointwise directed, so that it has $f_0 = \bigvee^{\bullet \mathrm{p}} F$ as its $\bullet$-pointwise lub. We show that f_0 is also the $\star$-stable lub of F.

For every $f \in F$, $f \leq^{\star \mathrm{p}} f_0$, since

$$f_0 x = \bigvee^{\bullet} \{fx \mid f \in F\} = \bigvee^{\star} \{fx \mid f \in F\}$$

by (BD2.2) for C'. This also shows that f_0 is the $\star$-pointwise lub of F.

If $x \leq^{\star} y$ and g is $\star$-stably consistent with every $f' \in F$, then, using directedness of F,

$$\begin{aligned}
gx \wedge^{\bullet} f_0 y &= gx \wedge^{\bullet} \bigvee^{\star} \{f'y \mid f' \in F\} \\
&= \bigvee^{\star} \{gx \wedge^{\bullet} f'y \mid f' \in F\} && \text{(Continuity of $\wedge^{\bullet}_{C'}$)} \\
&= \bigvee^{\star} \{gy \wedge^{\bullet} f'x \mid f' \in F\} && (g \Uparrow^{\star \mathrm{s}} f' \;\;\&\;\; x \Uparrow^{\star} y) \\
&= gy \wedge^{\bullet} \bigvee^{\star} \{f'x \mid f' \in F\} && \text{(Continuity of $\wedge^{\bullet}_{C'}$)} \\
&= gy \wedge^{\bullet} f_0 x.
\end{aligned}$$

Thus, if $g \Uparrow^{\star \mathrm{p}} f_0$ and $g \Uparrow^{\star \mathrm{s}} F$ then $g \Uparrow^{\star \mathrm{s}} f_0$; this applies to every $f \in F$, by $\star$-stable directedness, and to every $\star$-stable upper-bound of F. Together with f_0's being the $\star$-pointwise lub of F, this allows us to conclude that f_0 is also the $\star$-stable lub of F.

(BD3) Let $f_1 \leq^\star f_1'$ and $f_2 \leq^\star f_2'$ be functions in $C \to_{\mathcal{C}} C'$. For every $x \in C$,

$$(f_1 \wedge^\bullet f_2)x = f_1 x \wedge^\bullet f_2 x \leq^\star f_1' x \wedge^\bullet f_2' x = (f_1' \wedge^\bullet f_2')x,$$

using $\star$-monotonicity of $\wedge^\bullet_{C'}$ (BD3), so that $f_1 \wedge^\bullet f_2 \leq^{\star \mathrm{p}} f_1' \wedge^\bullet f_2'$. If $x \Uparrow^\star y$ then, using stability of the functions and definition of the $\star$-stable order,

$$\begin{aligned}
(f_1 \wedge^\bullet f_2)x \wedge^\bullet (f_1' \wedge^\bullet f_2')y &= f_1 x \wedge^\bullet f_2 x \wedge^\bullet f_1' y \wedge^\bullet f_2' y \\
&= f_1 y \wedge^\bullet f_2 y \wedge^\bullet f_1' x \wedge^\bullet f_2' x \\
&= (f_1 \wedge^\bullet f_2)y \wedge^\bullet (f_1' \wedge^\bullet f_2')x,
\end{aligned}$$

and $f_1 \wedge^\bullet f_2 \leq^{\star \mathrm{s}} f_1' \wedge^\bullet f_2'$, so that $\wedge^\bullet$ is $\star$-monotone.

(BD4) Let $F, F' \subseteq C \to_{\mathcal{C}} C'$ be $\bullet$-pointwise directed sets as assumed for (BD4), with lubs $f_0 = \bigvee^{\bullet \mathrm{p}} F$ and $f_0' = \bigvee^{\bullet \mathrm{p}} F'$. For every $x \in C$, $f_0 x \leq^\star f_0' x$, by applying (BD4) for C', hence $f_0 \leq^{\star \mathrm{p}} f_0'$. Let $x \Uparrow^\star y$ and

$$\begin{aligned}
Z_1 &= \{fx \wedge^\bullet f'y \mid f \in F \;\;\&\;\; f' \in F'\} \\
Z_2 &= \{fy \wedge^\bullet f'x \mid f \in F \;\;\&\;\; f' \in F'\}\,.
\end{aligned}$$

By assumption, for every $f \in F$ and $f' \in F'$ there exist $f_1 \in F$ and $f_1' \in F'$ such that $f \leq^{\bullet \mathrm{p}} f_1$, $f' \leq^{\bullet \mathrm{p}} f_1'$ and $f_1 \leq^{\star \mathrm{s}} f_1'$. Thus,

$$fx \wedge^\bullet f'y \leq^\bullet f_1 x \wedge^\bullet f_1' y = f_1 y \wedge^\bullet f_1' x \in Z_2,$$

and $\bigvee^\bullet Z_1 \leq^\bullet \bigvee^\bullet Z_2$. The reverse inequality can be shown symmetrically, and, by $\bullet$-continuity of $\wedge^\bullet$,

$$f_0 x \wedge^\bullet f_0' y = \bigvee\nolimits^\bullet Z_1 = \bigvee\nolimits^\bullet Z_2 = f_0 y \wedge^\bullet f_0' x.$$

Therefore $f_0 \leq^{\star \mathrm{s}} f_0'$, and (BD4) holds.

- The above is sufficient to establish that $C \to_{\mathcal{C}} C'$ is a bidcpo. Note that, by 6.1.4, $(C \to_{\mathcal{C}} C', \leq^{\star \mathrm{s}})$ has consistent meets, which coincide with the $\bullet$-meets, that is, are determined pointwise.

(BDd) Finally, observe that $(C \to_{\mathcal{C}} C', \leq^{\star \mathrm{s}})$ is already assumed to have consistent joins and thus is bounded-complete, it is distributive since $C'^\star$ is distributive and consistent meets and consistent joins are determined pointwise, and, moreover, an $\star$-stable consistent join in $C \to_{\mathcal{C}} C'$ is also the $\bullet$-pointwise join. ∎

6.1.12 Corollary *The stable exponentiation of two d-bidcpos is a d-bidcpo.*

6.1.13 Proposition BST *is a CCC.*

Proof: Stability of application $\mathsf{app} : (C \to_{\mathrm{bs}} C') \times C \to C'$ follows from its continuity as a function $\mathsf{app} : (C^\bullet \to_{\mathrm{c}} C'^\bullet, \leq^{\mathrm{p}}) \times C^\bullet \to C'^\bullet$ and its stability as a function $\mathsf{app} : (C^\star \to_{\mathrm{s}} C'^\star, \leq^{\mathrm{s}}) \times C^\star \to C'^\star$. Note that these three versions of application have differing domains of definition, but the bidcpo version is clearly a restriction of the other two, and its $\bullet$-continuity and $\star$-stability may be inferred. ∎

Admissible Maps

We will be interested in maps that preserve the admissible structure of function spaces, but can give the definition over d-bidcpos in general: $f : C \to C'$ is an ***admissible map*** iff f is stable and preserves $\bullet$-meets and $\star$-consistent $\star$-joins.

Note that $\bullet$-directed lubs that are also of interest for admissibility will be preserved by stable functions. Also, there is some redundancy in the definition in that preservation of $\bullet$-meets implies preservation of bounded $\star$-consistent $\star$-meets, so that stability may be relaxed: a function is admissible iff it is $\bullet$-continuous, $\star$-monotone and preserves $\bullet$-meets and $\star$-consistent $\star$-joins.

6.1.14 Proposition *Admissible maps are closed under composition and function product.*

The product operation $(\times)$ *on functions, pre- and post-composition* $(\ \circ f)$ *and* $(f \circ\ )$ *operations for every stable function* f*, and the uncurrying operation are all admissible when restricted to admissible subsets of their source*[2]*.*

Proof: Closure under composition is immediate. Closure under function product follows from use of componentwise orders on the product.

Admissibility of the function-product operation, pre- and post-composition with fixed stable functions and the uncurrying operation, when restricted to admissible function spaces, are shown using similar arguments. We show here that the product and uncurrying operations preserve $\star$-consistent $\star$-joins, and the rest is very similar.

To see that the product map

$$(\times) : (C_1 \to C_1') \times (C_2 \to C_2') \to (C_1 \times C_2 \to C_1' \times C_2'),$$

restricted to admissible subsets of the function spaces, preserves $\star$-consistent $\star$-joins, taken $\bullet$-pointwise over admissible function spaces and componentwise over products, let $f_1 \Uparrow^{\star s} g_1$ and $f_2 \Uparrow^{\star s} g_2$. Then, for every $x \in C_1$ and $y \in C_2$,

$$\begin{aligned} ((f_1 \vee^{\bullet p} g_1) \times (f_2 \vee^{\bullet p} g_2))(x, y) &= (f_1 x \vee^{\bullet} g_1 x, f_2 y \vee^{\bullet} g_2 y) \\ &= (f_1 x, f_2 y) \vee^{\bullet} (g_1 x, g_2 y) \\ &= ((f_1 \times f_2) \vee^{\bullet p} (g_1 \times g_2))(x, y). \end{aligned}$$

To see that the uncurrying operation

$$\mathsf{uncurry} : (C_1 \to (C_2 \to C_3)) \to (C_1 \times C_2 \to C_3),$$

[2] *Note that we must take* $\mathsf{uncurry} : (C_1 \to (C_2 \to C_3)) \to (C_1 \times C_2 \to C_3)$ *as restricted to an admissible subset of* $C_1 \to F$*, where* F *is an admissible subset of* $C_2 \to C_3$*. We allow ourselves to regard* $f : C_1 \to (C_2 \to C_3)$ *as a function* $f : C_1 \to F$ *when* $fC_1 \subseteq F$*. The product map must be restricted to admissible subsets in each of the components of its source.*

For the target of uncurry *or any of the other operations we may always take the full stable function space between* $C_1 \times C_2$ *and* C_3*, since uncurrying preserves stability and the stable function space is admissible.*

Of course, admissibility of a function with respect to stable function spaces implies its admissibility with respect to any admissible subset of the stable functions, as long as the image of its restricted source is contained within its restricted target.

restricted to an admissible subset of $C_1 \to F$ where F is an admissible subset of $C_2 \to C_3$, preserves $\star$-consistent $\star$-joins, let $f \Uparrow^{\star s} g$. Then for every $(x, y) \in C_1 \times C_2$,

$$\begin{aligned}
\mathsf{uncurry}(f \vee^{\bullet p} g)(x, y) &= (f \vee^{\bullet p} g)xy \\
&= (fx \vee^{\bullet p} gx)y \\
&= (fxy) \vee^{\bullet} (gxy) \\
&= \mathsf{uncurry}\, f(x, y) \vee^{\bullet} \mathsf{uncurry}\, g(x, y) \\
&= (\mathsf{uncurry}\, f \vee^{\bullet p} \mathsf{uncurry}\, g)(x, y).
\end{aligned}$$

This relies, of course, on the $\bullet$-pointwise definition of $\star$-consistent $\star$-joins for admissible function spaces, as well as preservation of $\star$-consistency by application. ∎

The composition and application operations are not admissible in general.

6.2 Bidomains

The next step in the development is a restriction of the objects under consideration by imposing algebraicity on the d-bidcpos. This leads to bidomains, defined by Berry to be d-bidcpos C that are strongly algebraic via a stable SFP Ψ for $C^\star$ that is also a continuous SFP for $C^\bullet$.

A ***stable projection*** for a d-bidcpo C is a stable function $\psi : C \to C$ that is idempotent ($\psi \circ \psi = \psi$), and $\star$-stably deflationary ($\psi \leq^{\star s} \mathsf{id}$). ψ is a ***finite*** stable projection iff ψC is a finite set. A ***stable SFP*** Ψ for a d-bidcpo C is a countable sequence $\psi_1, \psi_2, \ldots$ of finite stable projections that is increasing and has the identity as its lub in the $\star$-stable order, *i.e.*, $\psi_i \leq^{\star s} \psi_{i+1}$ and $\mathsf{id} = \bigvee^{\star s} \{\psi_i \mid i \in \mathbb{N}\}$.

6.2.1 Proposition *ψ is a stable projection on C iff it is a stable projection on $C^\star$ and a (continuous) projection on $C^\bullet$. Ψ is a stable SFP on C iff it is a stable SFP on $C^\star$ and a (continuous) SFP on $C^\bullet$.*

Proof: Immediate from definitions. ∎

A ***bidomain*** is a d-bidcpo that has a stable SFP. We may also say that such a d-bidcpo is ***strongly algebraic***.

Here are some properties of bidomains, completing the bidcpo properties in 6.1.5:

6.2.2 Proposition *For a bidomain C,*

(1) $C^\bullet$ is a Scott domain.

(2) $C^\star$ is a dI-domain.

(3) $\mathbf{K}\, C^\bullet = \mathbf{K}\, C^\star$.

(4) For every $X \subseteq C$, if $\Uparrow^\star X$ then $\bigvee^\star X = \bigvee^\bullet X$. In other words, the identity $\mathsf{id} : C^\star \to C^\bullet$ preserves all existing lubs.

(5) Property (BD4′) may be substituted for property (BD4) in the definition of a bidomain.

Proof: Let C be a bidomain with stable SFP Ψ.

(1) Corollary of 2.1.23, since Ψ is a continuous SFP for $C^\bullet$.

(2) Corollary of 2.1.27, since Ψ is a stable SFP for $C^\star$.

(3) The inclusion $\mathbf{K}\,C^\bullet \subseteq \mathbf{K}\,C^\star$ is already given for a general bidcpo by 6.1.5. For the equality, observe that by 2.1.22 $\mathbf{K}\,C^\bullet$ and $\mathbf{K}\,C^\star$ are both uniquely determined by Ψ, a continuous SFP for both $C^\bullet$ and $C^\star$, and we have

$$\mathbf{K}\,C^\bullet = \mathbf{K}\,C^\star = \bigcup\{\psi_i C \mid i \in \mathbb{N}\}\,.$$

(4) For bidcpos in general, if $X \subseteq C$ has lub $\bigvee^\star X$ in $C^\star$ then it is certainly $\bullet$-consistent, so that it has a $\bullet$-lub, and $\bigvee^\bullet X \leq^\bullet \bigvee^\star X$. For the reverse inequality we use the above properties of bidomains. If $y \in \mathbf{K}\downarrow^\star(\bigvee^\star X)$ then there exists some finite subset $X' \subseteq X$ such that $y \leq^\star \vee^\star X'$ (because by taking joins of a consistent set of a Scott domain one obtains a directed set with the same lub). By (BDd), $\vee^\star X' = \vee^\bullet X'$, and $y \leq^\bullet \vee^\bullet X' \leq^\bullet \bigvee^\bullet X$. Therefore $\bigvee^\star X \leq^\bullet \bigvee^\bullet X$, and the desired equality follows.

(5) Corollary of 6.1.2. ∎

By this proposition, we may unambiguously talk of isolated elements of a bidomain, and write $\mathbf{K}\,C$ for $\mathbf{K}\,C^\bullet = \mathbf{K}\,C^\star$.

Note that the essential thing about the definitions of strong algebraicity for bidomains is that stable projections over bidcpos is are simultaneously $\bullet$-continuous and $\star$-stable, and, moreover, a stable SFP for a bidcpo C is simultaneously a stable SFP for $C^\star$ and a (continuous) SFP for $C^\bullet$. It is not sufficient for $C^\star$ to have a stable SFP Ψ_1 and for $C^\bullet$ to have a different (continuous) SFP Ψ_2. In fact, Berry has shown [Ber79] examples of bidcpos that satisfy all properties listed in 6.1.5 and 6.2.2, but fail to have a stable SFP as required. This added level of complexity helps to explain why, in contrast to dI-domains and Scott domains that have a direct characterization without recourse to strong algebraicity (2.1.27 and 2.1.23), no such direct characterization is currently known for bidomains.

A bidomain is ***trivial*** if its two orders coincide. Trivial bidomains coincide with dI-domains.

6.2.3 Proposition *$(A, \leq)$ is a dI-domain iff $(A, \leq, \leq)$ is a bidomain.*

Proof: Corollary of 2.1.27 and 6.1.3, and, for the reverse direction, corollary of 6.2.2. ∎

6.2.4 Proposition *Bidomains are closed under product and stable exponentiation. The category of bidomains and stable functions is a CCC.*

Proof: We already know that the underlying biposet of product or stable exponentiation of bidomains is a d-bidcpo (6.1.6 and 6.1.12), and strong algebraicity follows immediately from the development for continuous and stable strong algebraicity in the uni-ordered case (section 2.1.5).

Stability of the product morphisms and application, already established for BST, remains valid. ∎

6.3 Indexed Bidomains

In this section we examine the extension of the framework of indexed computation to bi-ordered structures. Technically, this entails equipping biposets with index structures, representing the collection of primitives available for the computation, and designating the class of well behaved functions between indexed biposets.

6.3.1 Indexability for Bidcpos

The order of a poset matters in establishing monotonicity, continuity and stability of an index structure over the poset. T_0-separation is independent of the order, and its conjunction with monotonicity implies order separation for the chosen order.

In defining indexable functions between dcpos, the order plays a further role beyond establishing continuity of the functions: This is in the quantification over non-empty finite $X \subseteq \uparrow x$ that we sometimes hide as an implicit assumption of the notation $r \in \mathcal{I}(x, X)$.

When we move to bidcpos there are, at least formally, two candidates for these roles — $\leq^\bullet$ and $\leq^\star$. We will use $\leq^\star$, which we view as the intrinsic information ordering on the underlying set. Technically, this choice will afford us an easy generalization of some of the results for the uni-ordered case. We say more on this in section 6.4.

Thus, we define an ***index structure*** for a bidcpo C to be an index structure for $C^\star$, or, alternatively put, a $\star$-monotone index structure over the underlying set. An index structure over C is said to be ***continuous*** or ***stable*** iff it is continuous or stable as an index structure over $C^\star$.

A function $f : C \to C'$ between indexed bidcpos is ***indexable*** iff it is $\bullet$-continuous and $\star$-indexable. In other words, if it is $\bullet$-continuous, $\star$-continuous, and for every x, non-empty finite $X \subseteq \uparrow^\star x$ and $r' \in \mathcal{I}'(f; x, X)$ there exists $r \in \mathcal{I}(x, X)$. Note the use of the information order in the definition. The requirement of $\star$-continuity may be weakened to $\star$-monotonicity under assumption of $\bullet$-continuity.

6.3.1 Proposition *An indexable function between stably indexed bidcpos is stable.*

Proof: Follows immediately from 3.6.2. ∎

We now use the previous development for admissibility to show that the bi-ordered indexable function space is a d-bidcpo.

6.3.2 Proposition *The indexable functions between stably indexed d-bidcpos form an admissible subset of the stable function space.*

The category of stably indexed d-bidcpos and indexable functions is (equivalent to) an admissible subcategory of BST.

Proof: Since we assume stable index structures, the indexable functions are stable. By proposition 6.1.10, the $\bullet$-pointwise meet, $\bullet$-pointwise directed lub and $\star$-stably consistent join operations on indexable functions produce stable functions. We conclude the proof by recalling that these operations also preserve $\bullet$-indexability: The $\bullet$-pointwise lub of an $\bullet$-pointwise directed set of $\star$-indexable functions is $\star$-indexable by 3.7.2. The $\bullet$-pointwise meet of $\star$-indexable functions is $\star$-indexable by 3.7.10. The $\bullet$-pointwise join of $\star$-stably consistent $\star$-indexable functions is $\star$-indexable by 3.7.7. ∎

6.3.3 Corollary *The indexable function space between stably indexed d-bidcpos, ordered $\bullet$-pointwise and $\star$-stably, is a d-bidcpo.*

Proof: Follows from 6.1.11 and 6.3.2. ∎

Another useful fact that generalizes smoothly from the uni-ordered case is down-closure of indexable functions in the stable order, corresponding to the $\star$-stable order in the bi-ordered setting.

6.3.4 Proposition *For continuous functions $f, g : C \to C'$, if $f \leq^{\star s} g$ and g is indexable then f is indexable as well.*

Proof: Follows directly from 3.7.5. ∎

6.3.5 Corollary *If C is an indexed bidomain then it has a stable SFP consisting of indexable finite projections.*

Proof: As a matter of fact, every stable SFP Ψ over C consists of indexable functions, since they are all $\star$-stably dominated by the identity, which is indexable. ∎

We delay the treatment of further aspects of indexable functions and bi-ordered structures — such as product — to the next chapter.

6.4 Conformality of Index Structures

At this point, we would like to give an alternative motivation for the use of bi-ordered structures, emerging from the question of conformality of index structures. This also motivates the choice of the information order for assessing indexability.

Suppose we are given a set A with some partial order $\leq^\star$. An index structure $\mathcal{I}$ over A defines another ordering over A,

$$x \leq^\bullet y \quad \text{iff} \quad (\checkmark\ x) \subseteq (\checkmark\ y).$$

(That is, $x \leq^\bullet y$ iff every index r that fills x also fills y.)

This defines a partial order: Reflexivity and transitivity of $\leq^\bullet$ follow from the same properties of inclusion, and anti-symmetry follows from T_0-separation of the index structure.

We call $\leq^\star$ the ***information order***, since it is assumed to convey some intrinsic ordering of A by information content. We call $\leq^\bullet$ the ***approximation order***, since it orders A according to observable approximability using $\mathcal{I}$. We may now talk about the relationship between the two orders.

Recall that we have defined a family O of subsets of a poset B to be *conformal* when it completely defines the partial order, *i.e.*, $x \leq y$ iff $\{p \in O \mid x \in p\} \subseteq \{p \in O \mid y \in p\}$. Thus, by definition, an index structure will always be conformal with respect to the approximation order that it induces. Therefore, it is only interesting to consider separation properties of an index structure with respect to the intrinsic information order.

When the index structure is *conformal* (with respect to the information order) the two orders coincide. One may then talk of a single order $\leq = \leq^\star = \leq^\bullet$ on A, uniquely determined by an index structure over A, that relates elements by both information content and their approximability by the index structure.

The index structures that we have defined so far are conformal:

6.4.1 Proposition *Data index structures and constant-applicative index structures (over domains with conformal index structures) are conformal.*

Proof: Easy to check, details omitted. ∎

The more general case, however, is that of *non-conformal* index structures, for which $\leq^\bullet$ may differ from $\leq^\star$. As an example consider the function space $\mathbf{2} \to \mathbf{2}$ with the stable order as the intrinsic information order (see figure 6.1), and take an index structure with extents $\{\mathsf{K}\top\}$ and $\{\mathsf{id}, \mathsf{K}\top\}$. We have that $\mathsf{id} \leq^\bullet \mathsf{K}\top$ in the approximation order induced by the index structure, even though the two functions are not related by the stable order; the index structure thus fails to be conformal.

Thus, there are two ways to view the construction of an indexed bidcpo. One may start with a bidcpo $C = (A, \leq^\bullet, \leq^\star)$ and equip it with an index structure for $C^\star$. If this index structure induces $\leq^\bullet$, then an equivalent view is that one starts out with a poset $C^\star = (A, \leq^\star)$, which is equipped with an index structure and a biposet is obtained when one considers the approximation order induced on A by the index structure.

Of course, to obtain an indexed bidcpo by adding an index structure to $(A, \leq^\star)$ still necessitates that the bidcpo axioms be satisfied for $(A, \leq^\bullet, \leq^\star)$; it is not clear at this point to what extent this can be guaranteed by imposing further requirements on the index structure. We do not devote additional attention to this question, except for pointing out that monotonicity of an index structure over $(A, \leq^\star)$, essentially a guarantee that computation must respect the information ordering, guarantees that the two orders are related as required by axiom (BD2.1):

6.4.2 Proposition *If $\mathcal{I}$ is a monotone index structure for $(A, \leq^\star)$ then*

$$x \leq^\star y \implies x \leq^\bullet y.$$

Proof: Monotonicity is equivalent to

$$x \leq^{\star} y \implies (\checkmark\ x) \subseteq (\checkmark\ y),$$

and the right-hand side is equivalent to $x \leq^{\bullet} y$. ∎

Chapter 7

Applicative Sequentiality

We have described first-order sequentiality within the framework of indexed computation, using the notion of *data index structures*. Our first attempt to extend this to higher-order sequentiality was based on *constant-applicative index structures* for function spaces, together with the *simple product assumption*. This chapter proposes a more sophisticated approach for the treatment of higher-order sequentiality.

Applicative indices will generalize the constant-applicative indices over the function space. As underlying objects we employ *bidomains*, following Berry's observation that the unique fully abstract extensional model of PCF is a model of bidomains [Ber79]. We argue that every object should be equipped with an *applicative structure*, that is, a way of being seen, and indexed, as a function space. Thus the objects employed here are bidomains with applicative structure, which we call *spaces*. The product of two spaces is a space, hence it may essentially be indexed as a function space; this replaces the simple product assumption.

The morphisms of interest between spaces are the fully indexable functions — functions whose fully uncurried form is indexable with respect to the applicative index structure. Of particular technical interest is the development related to full indexability of the product morphisms. We are not able to establish that fully indexable functions are closed under composition, so that we do not obtain a category of fully indexable functions, nor do we obtain a fully indexable model of PCF. We discuss these shortcomings and possible remedies in the concluding section of this chapter.

7.1 From Constant-applicative to Applicative Indices

This section is a mostly informal presentation of the rationale leading to the definitions employed in this chapter. Our aim is to obtain a satisfactory notion of sequential functions between bidomains by designating suitable index structures for the product of bidomains and for the sequential function space between bidomains. We sketch and iteratively refine here some candidate definitions. The definitions themselves appear in the next section, except for the definition of a hierarchical presentation of an index structure that may be safely given here.

7.1.1 Selecting an Underlying Category

We aim to take indexed bidomains as objects under consideration in this chapter. It turns out that we will need to add yet some more structure, and we will eventually define *applicative spaces*, or just *spaces*, that are bidomains with additional *applicative structure*.

Although we do not explicitly explore this avenue, it should be evident that the treatment given here for bidomains should also apply if uni-ordered structures are employed instead (with the stable order on the function space, and to a certain extent if the pointwise order is employed). We prefer to work directly with bidomains for the reasons given in chapter 6: one needs to restrict functions by recourse to both the stable and pointwise orders on a function space, and, as Berry has shown [Ber79], the fully abstract model of PCF is a model of bidomains and (a subset of the) stable functions on bidomains. We will therefore adopt as an underlying category the category of bidomains and stable functions, and we will aim to continue to restrict it, so as to obtain a model with a better fit for PCF. It turns out that this is not an optimal substrate on which to build, and we point out possible improvements and alternatives in the concluding remarks.

We assume that data types — types with no arrows — are to be interpreted over some cartesian subcategory of the underlying category that is *a priori* equipped with index structures. We call this subcategory the *data category*, and call its objects, which are objects of the underlying category equipped with an index structure, *data domains*. As morphisms we take the indexable functions between the data domains. The choice of a data category thus fixes a notion of first-order sequentiality. For our purposes, it seems natural to choose for this role the category of data-indexed dI-domains, trivially seen as indexed bidomains. Following the results of chapter 4, any construction based on this foundation will coincide with PCF-sequentiality at first order.

Even though we do not emphasize it, and concentrate solely on the specific choices we have made, there seems to be some generality in the choice of a data category, subject to certain assumptions that we will point out in the course of the discussion. The main assumption that we will find necessary, and that seems to capture the essence of what it means to be a domain of data, is that the index structure on the product of data domains is determined by the simple product assumption.

We will use the term *data domain* to refer to an object of the data category, and *data space* to refer to a space that is, essentially, adequate for interpreting a data type. The two notions are distinct, but interchangeable for all but the most formal uses, the difference between the two being the difference between a dI-domain and the same dI-domain regarded as a data-indexed trivial bidomain with a trivial applicative structure.

7.1.2 Functionalizing the Prompter

An obvious built-in deficiency of the constant-applicative index structure over a function space $C \to C'$ — assume some generic function space at this point — is the constancy of the prompters that they employ; in other words, the fact that in the ca-index $[x \Rightarrow^{\mathrm{ca}} r]$, with $r \in \mathcal{I}_{C'}$ representing a residual computation over the target, the prompter $x \in C$ is constant and does not depend on the computational context.

This can be remedied by turning the prompter from a constant element $x \in C$ into a *prompter function* $h : C_0 \to C$ into C. The source C_0 of the prompter function would represent the context of the computation, or, the universe over which computation is taking place, so that h would be able to encapsulate the desired dependence; if $z \in C_0$ represents the current information available overall, then $hz \in C$ is the value that is to be used to prompt for further information in the function space $C \to C'$.

Intuitively, the definition is intended to reflect the fact that information about a function space is gained by applying it to some function of the computational context. Operationally, one may think of C_0 as representing (the product of the domains for) all of the free variables at the point where we wish to prompt a variable of arrow type.

7.1.3 Applicative Indices for Function Spaces

We start by considering the relatively simple case of a computation over the function space under consideration, so that $C_0 = C \to C'$. On the syntactic or operational level, one may think of this as having only one free variable of arrow type, as in example 4.4.3.

In this case, h is a function from $C \to C'$ into C, and the generalization is from a constant-applicative index $[x \Rightarrow^{\mathrm{ca}} r]$ with

$$[x \Rightarrow^{\mathrm{ca}} r] \checkmark_{C \to C'} f \qquad \text{iff} \qquad r \checkmark_{C'} fx$$

to an *applicative index* $[h \Rightarrow^{\mathrm{a}} r]$ with

$$(*) \qquad [h \Rightarrow^{\mathrm{a}} r] \checkmark_{C \to C'} f \qquad \text{iff} \qquad r \checkmark_{C'} f(hf).$$

As for constant-applicative indices, we set

$$\mathbf{Ans}[h \Rightarrow^{\mathrm{a}} r] = \{[h \Rightarrow^{\mathrm{a}} s] \mid s \in \mathbf{Ans}\, r\}\,.$$

The constant-applicative indices are clearly a special case of the applicative indices, by setting $h = \mathsf{K}x$, a constant prompter function.

Of course, the prompter function h of an applicative index should be drawn from some restricted set of functions. It ought to be definable in the language — at least if its definability is required for definability of isolated elements — so as to assist in a proof of definability along the lines of the proof for the constant-applicative case (proposition 5.3.3) where the defining term for a constant prompter x would be generalized to a defining term for the prompter function h, applied to all the variables at that point — which constitute the context.

Note that a data space B may be given applicative indices by regarding it as the trivial function space $\mathbf{1} \to B$. This just says that in order to obtain information about a data space there is no need to apply it. The only applicable prompter function is $\mathsf{K}\bot : B \to \mathbf{1}$, and $(*)$ specializes to:

$$[\mathsf{K}\bot \Rightarrow^{\mathrm{a}} r] \checkmark_{\mathbf{1} \to B} \mathsf{K}x \qquad \text{iff} \qquad r \checkmark_B x.$$

That is, the applicative index structure over a data space viewed as a function space coincides with the original index structure over the data space.

But before we can formalize applicative indices properly, we need to address the circularity in the emerging definition: observe that indexable functions over the function space, such as $h : (C \to C') \to C$, may only be identified once we have associated an index structure with the function space; and we are now proposing to construct an index structure that relies on indexable functions of this type. We break the circularity by employing a hierarchical presentation of the applicative index structure.

7.1.4 Hierarchical Presentation of Index Structures

7.1.1 Definition [Hierarchical Presentation] An index structure $\mathcal{I}' = (I', \checkmark', \mathbf{Ans}')$ over a poset C is said to ***extend*** another index structure $\mathcal{I} = (I, \checkmark, \mathbf{Ans})$ over C iff

(HP1) The set of indices of $\mathcal{I}$ is a subset of the indices of $\mathcal{I}'$: $I \subseteq I'$.

(HP2) $\mathcal{I}'$ assigns the same extents and answers as $\mathcal{I}$ to every $r \in \mathcal{I}$: $(r \checkmark) = (r \checkmark')$ and $\mathbf{Ans}\, r = \mathbf{Ans}'\, r$ for every $r \in \mathcal{I}$.

It follows that every $r \in \mathcal{I}$ is an answer iff it is an answer in $\mathcal{I}'$: $\mathbf{Ans}\,\mathcal{I} = \mathcal{I} \cap \mathbf{Ans}\,\mathcal{I}'$.

(HP3) $\mathcal{I}$ is closed under taking answers in $\mathcal{I}'$: if $\mathbf{Ans}'\, r \subseteq \mathcal{I}$ then $r \in \mathcal{I}$, for every $r \in \mathcal{I}'$.

The relation of extension is clearly a partial order: reflexivity, transitivity and anti-symmetry essentially follow from the same properties of inclusion on the sets of indices, as used in (HP1).

A ***hierarchical presentation*** of an index structure $\mathcal{I}$ is an ω-chain of index structures $\{\mathcal{I}^\alpha \mid \alpha \in \mathbb{N}\}$ that is increasing and has lub $\mathcal{I}$ in the extension ordering. The lub is obtained by union; that is, by taking the collection of all indices occuring in the chain and preserving their extents and answers. Note that $\mathcal{I}$ is stable iff every one of its layers $\mathcal{I}^\alpha$ for $\alpha \in \mathbb{N}$ is stable.

We refer collectively to this hierarchy of index structures and to additional hierarchies that it directly induces as the ***indexable hierarchies***. Typographically, we reserve α and its variants to indicate a level in the indexable hierarchies, ranging over the naturals.

If $\mathcal{I}$ is given a fixed hierarchical presentation $\{\mathcal{I}^\alpha \mid \alpha \in \mathbb{N}\}$ then $r \in \mathcal{I}^\alpha$ is said to be an index at level α, or just an ***α-index***. The ***level*** $\mathsf{level}\, r$ of an index $r \in \mathcal{I}$ is defined to be the least α such that $r \in \mathcal{I}^\alpha$.

A layering of index structures also induces a layering of indexable functions. A function $f : (C, \mathcal{I}) \to (C', \mathcal{I}')$ is said to be ***α-indexable*** with respect to a hierarchical presentation $\{\mathcal{I}^\alpha \mid \alpha \in \mathbb{N}\}$ of $\mathcal{I}$ iff it is indexable as a function $f : (C, \mathcal{I}^\alpha) \to (C', \mathcal{I}')$; that is, indexability of f may be established using solely α-indices over the its domain of definition. The level $\mathsf{level}\, f$ of such an indexable function is the least α such that f is α-indexable. •

To break the circularity of the proposed definition of the applicative indices we will define the applicative indices by a hierarchical presentation. At level 0 we take no indices, and at level $\alpha + 1$ indices of the form $[h \Rightarrow^{\mathrm{a}} r]$ as above, where h is α-indexable, *i.e.*, its indexability may be established using indices that have been defined previously. Note that the indexable functions at level 0 are the constant functions. This in turn means that at level 1 of the applicative index hierarchy we obtain exactly the constant-applicative indices.

7.1.2 Remark The level of a function in the indexable hierarchy may be seen as a measure of the complexity of a proof of its being indexable. This is a constructive measure: the identity function, for instance, is trivially seen to be indexable, but it is maximal in the indexable hierarchy, since it should reply to a query by an α-index over its output by using the same α-index to query its input. Hence, showing it indexable should entail using all indices over the input.

The level in the indexable hierarchy can therefore be said to take into account the cost of constructing the indices used in the proof. Of course, since we aim for a coincidence of indexability and definability, this should be closely related to a constructive proof of definability, namely, constructing a term. •

7.1.5 Applicative Indices for Simple Products

We now have a blueprint for the definition of an index structure over a function space $C \to C'$, obtained by considering the case where there is no further context to the computation, $C_0 = C \to C'$. This will let us employ the indexable function space between indexed bidomains as a candidate for the exponentiation in the category of indexable functions on the indexed bidomains.

Consider now the general case: suppose that we are interested in knowing more about a function space $C \to C'$ — we call this the *main component* — but that there are additional components to the computational context. A naive formal rewriting of $(*)$ gives

$$(**) \qquad [h \Rightarrow^{\mathrm{a}} r] \checkmark_{C \to C'} f \quad \text{iff} \quad r \checkmark_{C'} f(hz),$$

for $z \in C_0$ and $h : C_0 \to C$ at level α, yielding $[h \Rightarrow^{\mathrm{a}} r]$ at level $\alpha + 1$. But what is the relationship between f and z? Presumably, if C_0 is meant to represent the context of the computation, then the function space $C \to C'$ ought to be a component of the overall context C_0; in other words, there must be some projection $\pi : C_0 \to (C \to C')$. f, then, must surely be the π-th component of z, *i.e.*, $f = \pi\, z$; and the right hand side of $(**)$ should read $r \checkmark_{C'} \pi\, z(hz)$. But now we notice that this is not just a criterion for obtaining information $f = \pi\, z$ about the main component; rather, it is a criterion for obtaining information z about the entire context C_0. Thus, we really end up defining an index structure over C_0 itself, relative to some choice of a main component designated by π:

$$(***) \qquad [h \Rightarrow^{\mathrm{a}}_{\pi} r] \checkmark_{C_0} z \quad \text{iff} \quad r \checkmark_{C'} \pi\, z(hz).$$

This reflects precisely the inadequacy of the simple product assumption for higher-order computation: even if we aim to gain information about a single component of the computational context, the prompter employed may be a function of additional components of the context, in which case the computation may also provide information about these additional components (see, for instance, example 4.4.1).

As an example, let us specialize $(***)$ for the case $C_0 = (C \to C') \times C_2$, where we choose $C \to C'$ as the main component, *i.e.*, $\pi = \pi_1$. we have:

$$[h \Rightarrow^{\mathrm{a}}_{1} r] \checkmark (f, y) \quad \text{iff} \quad r \checkmark_{C'} f(h(f, y)),$$

for $h : (C \to C') \times C_2 \to C$ and $r \in \mathcal{I}_{C'}$. (We indicate the choice of main component for the index by a subscript of 1, rather than π_1.)

This is a first indication that an index structure for the product may be defined in terms of applicative indices, at least in the case where the main component is taken to be a function space. Note that for a main component $C_1 = C \to C'$ we use the fact that C_1 is a function space to recover the *argument* domain $\mathsf{arg}C_1 = C$ and *result* domain $\mathsf{res}C_1 = C'$ of C_1, which are necessary in order to be able to talk about a prompter function from C_0 into $\mathsf{arg}C_1$, and a residual index $r \in \mathcal{I}_{\mathsf{res}C_1}$.

Let us consider now the operational analogies. The context object C_0 corresponds to the product of the types of the free variables, and the choice of a main component corresponds to the choice of a free variable $\mathtt{f} : \tau \to \tau'$ of arrow type, to be applied to some prompter expression that may incorporate all free variables, and corresponds to the prompter function. The residual index in the result domain of the main component corresponds to the syntactic context into which the application is placed.

Since we may consider data spaces as trivial function spaces, we may extend our treatment to cases where the main component is chosen to be a data domain B, designated by a projection $\pi : C_0 \to B$, by treating it as if it were the function space $\mathbf{1} \to B$. Again, a prompter function $h : C_0 \to \mathbf{1}$ may only be $\mathsf{K}\bot$, and $(*{*}*)$ effectively reduces to

$$[\mathsf{K}\bot \Rightarrow^{\mathrm{a}}_{\pi} r] \checkmark z \qquad \text{iff} \qquad r \checkmark_{C'} (\lambda x \in \mathbf{1} \,.\, \pi\, z)(\mathsf{K}\bot z) = \pi\, z$$

so that $[\mathsf{K}\bot \Rightarrow^{\mathrm{a}}_{\pi} r]$ is essentially equivalent to r: more precisely, it is equivalent to the injection of the index r of B into an index of C_0 . But this is a revival of the simple product assumption: applicative indices over the product with a main component that is a data space are essentially the indices of that component. If both components of the product are data objects then in fact the applicative indices over the product as outlined here will yield precisely the simple product index structure, and will coincide with the index structure assumed for the data category. This is entirely consistent with our contention that the simple product assumption is appropriate for data spaces.

7.1.6 Introducing the Applicative Structure

We have shown how applicative indices may be defined for (binary) products of function spaces (and data spaces), with close analogies to the construction of terms. This may be applied equally well to the construction of applicative indices for direct n-ary products or repeated products where the main component of the index is taken to be a function space (or a data space). Note, incidentally, that the applicative indices for a function space coincide with the applicative indices derived for the function space when considered as a unary product.

However, if we consider n-ary products via repeated binary products, or the associativity of products, we must be able to deal with main components that are not function spaces.

Consider the products $C_a = (C_1 \to C_1') \times ((C_2 \to C_2') \times C_3)$ and $C_b = ((C_1 \to C_1') \times (C_2 \to C_2')) \times C_3$. We would expect the indices defined for them to be essentially identical. Let us choose then $C_1 \to C_1'$ as the main component of C_a, and we can easily talk of applicative indices $[h \Rightarrow^{\mathrm{a}}_1 r]$ with $h : C_a \to C_1$, $r \in \mathcal{I}_{C_1'}$, and

$$[h \Rightarrow^{\mathrm{a}}_1 r] \checkmark_{C_a} (f, (g, z)) \qquad \text{iff} \qquad r \checkmark_{C_1'} f(h(f, (g, z))).$$

How would an analogous index be obtained for C_b? $C_1 \to C_1'$ is not directly a component of C_b; formally speaking, the best one can do instead is choose $(C_1 \to C_1') \times (C_2 \to C_2')$ as a main component, that is, choose to advance the information by application of a product. This makes no sense, unless we have a way of viewing the product $(C_1 \to C_1') \times (C_2 \to C_2')$ as a function space. And a more or less natural way to do so is via the product operation,

$$(\times) : (C_1 \to C_1') \times (C_2 \to C_2') \to (C_1 \times C_2 \to C_1' \times C_2'),$$

which may be viewed as embedding a product of function spaces into a function space on the products. Using this view of the product $C_\times = (C_1 \to C_1') \times (C_2 \to C_2')$, we would assign it an argument domain $\mathsf{arg}C_\times = C_1 \times C_2$ and a result domain $\mathsf{res}C_\times = C_1' \times C_2'$ so as to be able to employ it, at least formally, as a main component for the construction of an applicative index over C_b: $[h' \Rightarrow_1^{\mathrm{a}} r']$, with $h' : C_b \to \mathsf{arg}C_\times$, $r' \in \mathcal{I}_{\mathsf{res}C_\times}$, and

$$\begin{array}{lll} [h' \Rightarrow_1^{\mathrm{a}} r'] \checkmark_{C_b} ((f,g),z) & \text{iff} & r' \checkmark_{C_1' \times C_2'} (f \times g)(h'((f,g),z)) \\ & & \quad = (f(\pi_1 \circ h'((f,g),z)), g(\pi_2 \circ h'((f,g),z))) \end{array} .$$

Note the use of the embedding $(\times)$ to turn the pair (f,g) of functions into a function $f \times g$ over pairs.

In general, we will say that an applicative structure for C is an assignment to C of an argument domain $\mathsf{arg}C$, a result domain $\mathsf{res}C$, and an *applicative embedding* $\iota_C : C \to (\mathsf{arg}C \to \mathsf{res}C)$ which makes it possible to view every element $x \in C$ as a function $\iota x : \mathsf{arg}C \to \mathsf{res}C$.

If we now assume that r' is an index over the product $C_1' \times C_2'$ that is induced by the index r of C_1',

$$r' \checkmark_{C_1' \times C_2'} (x,y) \quad \text{iff} \quad r \checkmark_{C_1'} x,$$

then we can set $h' = \underline{\iota}_1 \circ h$ and we can establish one half of the desired correspondence, since

$$[h \Rightarrow_1^{\mathrm{a}} r] \checkmark_{C_a} (f,(g,z)) \quad \text{iff} \quad r \checkmark_{C_1'} f(h(f,(g,z))) \quad \text{iff} \quad [h' \Rightarrow_1^{\mathrm{a}} r'] \checkmark_{C_b} ((f,g),z).$$

7.1.7 Refining the Applicative Structure: Spaces

For the first half of the desired correspondence between applicative indices of C_a and applicative indices of C_b we assumed that an index $r \in \mathcal{I}_{C_1'}$ in a component of the product induces an index in the product $r' \in \mathcal{I}_{C_1' \times C_2'}$, which seems like a reasonable assumption to make — information about a component is also information about the whole.

The other direction is more problematic. Given $[h' \Rightarrow_1^{\mathrm{a}} r'] \in \mathcal{I}_{C_b}$, it is possible to set $h = \pi_1 \circ h'$, and *if* r' is induced by an index $r \in \mathcal{I}_{C_1'}$, then $[h \Rightarrow_1^{\mathrm{a}} r] \in \mathcal{I}_{C_a}$ is the desired corresponding index. If r' is induced by an index $r_2 \in \mathcal{I}_{C_2'}$ then we would expect to obtain an index of C_a with a different choice of main component. The problem, though, is that these two possibilities are not exhaustive, precisely because the simple product assumption is not generally valid any more.

The simple product assumption is valid if C_1' and C_2' happen to be data spaces. But if they are not, we can now resort to the abstraction afforded us by the use of an applicative structure; we can try to choose for $\mathsf{res}C_\times$ a data space, different from $C_1' \times C_2'$ if need be. This will restore the validity of the simple product assumption where it counts, and complete the desired correspondence between the applicative indices of C_a and C_b.

Thus, not only will we want the applicative structure to make it possible to view $C_\times$ as a function space; we will want to be able to view it as a function space whose result domain is a data space. To demonstrate the feasibility of obtaining such applicative structures, we first show that analogous argument and result type operations may be defined for the types of a simple type system (extended with product types):

$$\begin{array}{rclrcl} \mathsf{arg}\rho & = & \mathbf{1} & \mathsf{res}\rho & = & \rho \\ \mathsf{arg}(\tau_1 \times \tau_2) & = & \mathsf{arg}\tau_1 \times \mathsf{arg}\tau_2 & \mathsf{res}(\tau_1 \times \tau_2) & = & \mathsf{res}\tau_1 \times \mathsf{res}\tau_2 \\ \mathsf{arg}(\tau \to \tau') & = & \tau \times \mathsf{arg}\tau' & \mathsf{res}(\tau \to \tau') & = & \mathsf{res}\tau'. \end{array}$$

Note that the result type $\mathsf{res}\tau$ of every type τ is a data type, with no arrow constructors. Of course, in order to specify a data space that may be considered a result domain of a product $C_\times$ we may require access to similar result domains for the components of the product. Therefore, just as in the case of types, we must be able to talk about the inductive construction of objects.

We start working now with *spaces*; a space E is a bidomain (or, an object of the underlying category) equipped with an applicative structure consisting of an argument domain $\mathsf{arg}E$ which is itself a space, a result domain $\mathsf{res}E$ that is a data space, and an applicative embedding ι_E that maps E into the function space $\mathsf{arg}E \to \mathsf{res}E$.

The definition of argument and result types as above is also used in this chapter to define argument and result spaces for the product and exponentiation of spaces. A necessary refinement of this definition is presented and motivated in the concluding section of this chapter.

We must now introduce a hierarchy of spaces, to break the circularity of defining a space in terms of its argument domain which is itself a space. We associate with every space an order, and set $\mathsf{ord}\,E = 1 + \mathsf{ord}(\mathsf{arg}E)$. We may call this the *order hierarchy*, and reserve β and its variants to indicate level in this hierarchy, ranging over the naturals. At order 0 we take data spaces with a trivial applicative structure. Of course, by equipping objects with an applicative structure and imposing on them a countable hierarchy we force them to be finitely generated, in analogy with types. We return to this point of view in remark 7.2.2 below.

7.1.8 Full Uncurrying

Requiring that the result domains be *data* spaces when assigning an applicative structure $\iota_E : E \to (\mathsf{arg}E \to \mathsf{res}E)$ to a space E ties in nicely with our earlier discussion regarding the need to assess indexability of functions in their fully uncurried form (sections 3.4.1 and 5.2.3). For constructing the constant-applicative model we needed to rely on the type structure to obtain functions in their fully uncurried form. This is not necessary any more, thanks to the presence of applicative structure in spaces.

If $f : E \to E'$ is a function between spaces and $\iota_{E'} : E' \to (\mathsf{arg}E \to \mathsf{res}E')$ is the applicative embedding of E' then $\iota_{E'} \circ f : E \to (\mathsf{arg}E' \to \mathsf{res}E')$ is a form of f that produces a function representation of f's result, and $\hat{f} = \mathsf{uncurry}(\iota_{E'} \circ f) : E \times \mathsf{arg}E' \to \mathsf{res}E'$ may be seen as a fully uncurried version of f, since it is a function into a data space. We will call fully indexable the functions that are indexable in their fully uncurried form $\hat{f}$ with respect to the applicative index structures. These are the functions that we propose to employ as morphisms between spaces.

Following a decision to employ fully indexable functions, we need to go back and revise some of our earlier sketches. Most importantly, the prompter function h would need to be fully indexable, not just indexable. Note that in contrast to the definition of indexable functions, the definition of fully indexable functions on spaces does not guarantee *a priori* that, for instance, the identity or constant functions are fully indexable, that fully indexable functions are closed under composition, or that fully indexable functions are indexable. In fact, we have not been able to prove yet that fully indexable functions are indexable and that they are closed under composition, and we will discuss this further in section 7.4.

7.1.9 An Alternative Formulation of Applicative Indices

We started this discussion by outlining a definition of applicative indices for the simple case of function spaces. The essential aspect of a function space that makes this the simple case is that it possesses an explicit applicative structure. Therefore, now that we have decided to equip our objects with an applicative structure we will be able to define the applicative index structure uniformly for all spaces, roughly as follows. If E is a space, then an applicative index over E is $[h \Rightarrow^{\mathrm{a}} r]$, determined by a fully indexable $h : E \to \mathsf{arg}E$ and an index $r \in \mathcal{I}_{\mathsf{res}E}$, with

$$(****) \qquad [h \Rightarrow^{\mathrm{a}} r] \checkmark_E z \quad \text{iff} \quad r \checkmark_{\mathsf{res}E} \iota_E z(hz).$$

Note the use of ι_E to obtain the function representation of $z \in E$, prior to applying it to the prompter hz. Of course, the same proviso about circularity of the definition applies here, and we need to use a hierarchical presentation; the details are left to the formal definition.

With this definition the index structure is intrinsically determined by the space, and ceases to be an extra component of the structure. This is not a problem here as it was earlier (section 4.4) because the applicative index structure already is adapted to the higher-order case, and specializes to the data-index structure on data spaces.

A first point to note is that the applicative indices over a function space $E \to E'$ will differ in general from the initial sketch. Initially we used the natural applicative structure $\mathsf{arg}(E \to E') = E$ and $\mathsf{res}(E \to E') = E'$ with the identity serving as a trivial embedding, but now that we require the result domains to be data spaces and represent elements of spaces by functions in their fully uncurried form, we employ a fully uncurried applicative structure on the function space.

A second point of interest is the relationship to the applicative indices we have previously outlined for the product. These may now be reformulated cleanly with the help of the applicative structure by taking for indices over $E_1 \times E_2$ objects of the form $[h \Rightarrow^{\mathrm{a}}_i r]$ where $i \in \{1, 2\}$ determines the main component of the index, $h : E_1 \times E_2 \to \mathsf{arg}E_i$ is fully indexable, $r \in \mathcal{I}_{\mathsf{res}E_i}$, and

$$[h \Rightarrow^{\mathrm{a}}_i r] \checkmark_{E_1 \times E_2} (x_1, x_2) \quad \text{iff} \quad r \checkmark_{E_i} \iota_{E_i} x_i(h(x_1, x_2)).$$

In contrast, specializing the uniform definition $(****)$ of applicative indices yields indices $[h' \Rightarrow^{\mathrm{a}} r']$ where $h' : E_1 \times E_2 \to \mathsf{arg}(E_1 \times E_2)$, $r' \in \mathcal{I}_{\mathsf{res}(E_1 \times E_2)}$, and

$$[h' \Rightarrow^{\mathrm{a}} r'] \checkmark_{E_1 \times E_2} (x_1, x_2) \qquad \text{iff} \qquad r' \checkmark_{\mathsf{res}(E_1 \times E_2)} (\iota_{E_1} x_1 \times \iota_{E_2} x_2)(h'(x_1, x_2)),$$

using $\iota_{E_1 \times E_2} = (\times) \circ (\iota_{E_1} \times \iota_{E_2})$. We will eventually reconcile the two definitions and show that they are in fact equivalent, but before we do so we will use $(****)$ as the definition of applicative indices. To justify the intuition leading to the proof of coincidence, we note that $r' \in \mathcal{I}_{\mathsf{res}(E_1 \times E_2)}$ is an index over a product of data spaces, since $\mathsf{res}(E_1 \times E_2)$ will be defined as $\mathsf{res}E_1 \times \mathsf{res}E_2$, so that r' must be induced by an index r in one of the components; this serves to identify the main component for presenting $[h' \Rightarrow^{\mathrm{a}} r']$ in an alternate form — and this is exactly the point made above for requiring result domains to be data spaces. The rest of the argument follows the lines laid out there, and the main technical obstacles will be in showing that fully indexable functions and applicative indices are closed under the operations to convert between the two forms.

7.1.10 On the Connection to Sazonov's Approach

This is perhaps a good place to remark that the basic ideas that lead to our development — for instance, that information about a function space may only be obtained by application — may also be seen as guiding Sazonov's approach to the formalization of higher-order sequential computation. Correspondingly, our work and the work by Sazonov and the Siberian school [Saz76c, Saz76b, Tra85] may be described in very similar terms. The counterpart of our indices, the queries sitting at internal nodes of trees that describe sequential computation strategies, are applicative terms built up from variables, including functional ones, that represent (components of) the input (or, provably sequential functions). Terms with a head variable of functional type amount to an applicative index that seeks to gain information about (an element of) the function space by applying it to some function of the entire context of computation.

There are some important differences between the two approaches. Sazonov starts with a known model of PCF — the continuous functions model — and seeks to identify the sequential elements. Every element of the model may be considered as having a type (in the sense of belonging to the domain that interprets the type); the type may therefore be used in assessing sequentiality of functions. Only functions of fully uncurried types are considered and, importantly, compositionality and the product do not pose a problem — there is no need to define the queries over the product in terms of the components. Another important aspect seems to be the explicit relationship between what we would call prompter functions, and applicative terms, which helps to relate sequentiality and definability.

In contrast, our goal is to give a construction of a category, independent of a model or type system (even if we need to compromise on this to some extent by introducing an applicative structure). This goal determines a different set of technical problems that we need to solve, most prominently the treatment of the product, with ramifications concerning compositionality, as presented in the sequel.

7.2 A Formalization

This section presents the definitions of spaces and stable, indexable, and fully indexable functions between them. We also define a product of spaces and the fully indexable function space $E \to_{\mathrm{fx}} E'$ between spaces E and E'. The definitions are given in four separate pieces, but they are mutually recursive, defining inductively a hierarchy of spaces and the indexable hierarchies.

The construction is parameterized by the choice of an underlying *data category* that embodies a choice of first-order sequentiality. We choose to use here dI-domains and data-sequential functions between them, as defined in chapter 4. Between the different flavors of data sequentiality we choose the primal data index structures (section 4.2.3), because they possess the inducement property for the product. See proposition 4.3.1 for the properties of data indices over a product of domains. Recall that primal data indices have uniquely determined anchors and main components.

We may refer to dI-domains as *data domains* to emphasize the abstract role that they play here.

7.2.1 Spaces

7.2.1 Definition [Part 1: Spaces] Spaces are defined hierarchically, with a map ord taking each space to its order, a natural number. Order 0 spaces are called ***data spaces***. Other spaces are called ***functional spaces***. Order 1 spaces are said to be ***first-order functional spaces***, and all other spaces are considered to be ***higher-order functional spaces***.

An ***order*** $\beta + 1$ ***space*** E is a d-bidcpo C together with an ***applicative structure*** $(\mathsf{arg}E, \mathsf{res}E, \iota_E)$ for C, meaning that the following are true:

- $\mathsf{arg}E$, the ***argument space*** of E, is a space of order β.

- $\mathsf{res}E$, the ***result space*** of E, is a data domain, drawn from the underlying data category.

- We write $\mathsf{rep}\,E$ for $\mathsf{arg}E \to_{\mathrm{x}} \mathsf{res}E$, and call this the ***functional representation*** of E. $\mathsf{arg}E \to_{\mathrm{x}} \mathsf{res}E$ is the space of functions indexable (*i.e.*, $\bullet$-continuous and $\star$-indexable) with respect to the applicative index structure on $\mathsf{arg}E$ and the primal data index structure on $\mathsf{res}E$, bi-ordered $\bullet$-pointwise and $\star$-stably; it is defined together with $\mathsf{arg}E$, at an earlier stage of the construction of the space hierarchy, so that it may safely participate as a component in the construction of E.

- ι_E is a stable function between the (biposets underlying the) space E and its functional representation $\mathsf{rep}\,E$. We refer to ι_E as E's ***applicative embedding***; it lets us view every element of E as a fully indexable function from E's argument space to its result space.

  ι_E must also be strict, injective and admissible, *i.e.*, it must preserve $\bullet$-meets and

$\star$-consistent $\star$-joins:

$$\iota_E \bot_E = \bot_{\mathsf{rep}\,E} = \mathsf{K}\bot_{\mathsf{res}E}$$
$$\iota_E x = \iota_E y \quad \Longrightarrow \quad x = y$$
$$\iota_E(x \wedge^\bullet y) = \iota_E x \wedge^\bullet \iota_E y$$
$$\iota_E(x \vee^\star y) = \iota_E x \vee^\star \iota_E y, \quad \text{if } x \Uparrow^\star y$$

for all $x, y \in E$.

At order 0, a ***data space*** E is defined by a choice of a data domain B, drawn from the underlying data category, together with a choice of some one element set, a terminal object of the data category, that serves as $\mathsf{arg}E$. The underlying bidcpo of E is taken to be B, viewed as a primal data indexed trivial bidomain, and an applicative structure for E is completed by setting $\mathsf{res}E = B$ and $\iota_E = \lambda x \in B\ .\ \mathsf{K}x$. In this case, $\mathsf{rep}\,E = \mathsf{arg}E \to \mathsf{res}E$ consists of all functions from $\mathsf{arg}B$ to B, so that it is essentially isomorphic to B. Any one element set may be seen as a trivial data space, by taking it to be its own argument space and result space.

A function between spaces is monotone, continuous (with suitable qualification by $\bullet$- or $\star$-) or stable iff it has the corresponding property as a function between the underlying bidcpos. $\bullet$

7.2.2 Remark The order of a space is closely related to the order of the types that it is meant to interpret; in fact, the role of the applicative structure may be seen as the provision of necessary "type information" missing from the semantic domains. Every space is finitely generated, in the sense of having a finite order and being built out of finitely many atomic components — data spaces. The constructor we employ — an applicative embedding into a functional representation — turns out to be at least as general as the arrow and the product constructors on types. It remains to be seen whether this approach may be used to treat type systems with more complicated type constructors, such as universal quantification. $\bullet$

The decision to assign to a data space E a trivial applicative structure means that we take the function $\mathsf{K}x : \mathsf{arg}E \to \mathsf{res}E$ to represent $x \in \mathsf{res}E$, for some one-element set $\mathsf{arg}E$. Doing so will enable us to give a uniform development[1]. The choice of $\mathsf{arg}B$ is immaterial, so that data spaces are essentially equivalent to data domains, the objects of the underlying data category.

Note again that the elements of a functional space — a space of non-zero order — need not necessarily be functions, they can be pairs of functions, *etc*; rather, the applicative structure lets us view them as functions (with a data result): $\iota_E x \in \mathsf{rep}\,E$ is the function that represents $x \in E$.

One may view ι as being the embedding in an embedding-projection pair (ι, θ) between E and $\mathsf{rep}\,E$, and one may consider defining the *projective spaces* to be those spaces for which

[1]The alternative would have been not to assign an applicative structure to data spaces. One would then need to define and treat the product by several different cases, depending on the nature of the components of the product. This would also have undermined the uniform treatment of decompositions, to be presented later. The price we pay is a spurious choice of $\mathsf{arg}B$, and a need to convert between $x \in B$ and $\iota_B x = \mathsf{K}x$, but we will usually gloss over these details.

there exists a projection θ that completes an embedding-projection pair. For the time being the definition given here suffices.

7.2.3 Proposition *For every $x, y \in E$, $x \leq^\bullet y$ iff $\iota x \leq^\bullet \iota y$. If $x \Uparrow^\star y$ then $x \leq^\star y$ iff $\iota x \leq^\star \iota y$.*

Proof: ($\Longrightarrow$) follows by monotonicity of ι in both cases. ($\Longleftarrow$) for $\leq^\bullet$ follows by injectiveness of ι and its preservation of $\bullet$-meets:

$$\begin{array}{ll} & \iota x \leq^\bullet \iota y \\ \text{iff} & \iota x = \iota x \wedge^\bullet \iota y = \iota(x \wedge^\bullet y) \\ \text{iff} & x = x \wedge^\bullet y \\ \text{iff} & x \leq^\bullet y. \end{array}$$

For $\leq^\star$ employ the same reasoning, and use stability of ι together with assumption that $x \Uparrow^\star y$ to get preservation of $\star$-bounded $\star$-meet. ∎

7.2.2 The Indexable Hierarchies

We equip all spaces uniformly with the applicative index structure, and will implicitly assume applicative index structures when talking of indexable functions between spaces; this means that the index structure is intrinsic to our objects. We now define the applicative index structure for a space, and indexable and fully indexable functions between spaces. These form the indexable hierarchies, induced by a hierarchical presentation of the applicative index structure, following definition 7.1.1; this is necessary to prevent circularity of the definition.

7.2.4 Definition [Part 2: The indexable hierarchies] The indexable functions between spaces are those functions that are indexable as functions between the underlying bidcpos, indexed by the applicative index structure. Data domains will be regarded as primal data indexed trivial bidcpos in order to determine indexability.

For a function $f : E \to E'$ between two spaces, define the ***generalized fully uncurried form*** of f, denoted $\hat{f}$, by

$$\hat{f} = \mathsf{app}_{\mathsf{arg}E',\mathsf{res}E'} \circ ((\iota_{E'} \circ f) \times \mathsf{id}_{\mathsf{arg}E'}) : E \times \mathsf{arg}E' \to \mathsf{res}E'.$$

f is ***fully indexable*** iff it is stable and its fully uncurried form $\hat{f}$ is indexable. f is ***α-fully indexable*** iff it is stable and $\hat{f}$ is stable and α-indexable, *i.e.*, its indexability may be established using only α indices over $E \times \mathsf{arg}E'$. We write $E \to_{\mathrm{fx}} E'$ for the fully indexable function space between E and E', bi-ordered $\bullet$-pointwise and $\star$-stably; we will show it a bidomain and give it an applicative structure in a later part of the definition.

The applicative index structure for a space E is taken to be the index structure $\mathcal{I}_E$, given here by a hierarchical presentation. At level 0 let $\mathcal{I}_E^0 = \emptyset$, and, for $\alpha \in \mathbb{N}$,

$$\mathcal{I}_E^{\alpha+1} = \{[h \Rightarrow^{\mathrm{a}} r_0] \mid h : E \to \mathsf{arg}E \ \alpha\text{-fully indexable} \quad \& \quad r_0 \in \mathcal{I}_{\mathsf{res}E}\}\,.$$

For $[h \Rightarrow^{\mathrm{a}} r_0] \in \mathcal{I}_E$ and $x \in E$ we set

$$[h \Rightarrow^{\mathrm{a}} r_0] \checkmark_E x \qquad \text{iff} \qquad r_0 \checkmark_{\mathsf{res}E} \tilde{h}x$$

where $\tilde{h}x = \iota_E x(hx)$, and

$$\mathbf{Ans}_E[h \Rightarrow^{\mathrm{a}} r_0] = \{[h \Rightarrow^{\mathrm{a}} s] \mid s \in \mathbf{Ans}_{\mathsf{res}E}\, r_0\}\,.$$

Thus, $[h \Rightarrow^{\mathrm{a}} r_0]$ is an answer iff r_0 is an answer. •

We remark that applicative indices may have empty extents.

$$\begin{tikzcd}
E \times \mathsf{arg}E' \arrow[r, "f \times \mathsf{id}"] \arrow[drr, "\hat{f}"'] & E' \times \mathsf{arg}E' \arrow[r, "\iota_{E'} \times \mathsf{id}"] \arrow[dr, "\widehat{\mathsf{id}}_{E'}"'] & \mathsf{rep}\, E' \times \mathsf{arg}E' \arrow[d, "\mathsf{app}"] \\
 & & \mathsf{res}E'
\end{tikzcd}$$

Figure 7.1: The definition of $\hat{f}$

Figure 7.1 illustrates the definition of $\hat{f} = \mathsf{app} \circ ((\iota \circ f) \times \mathsf{id})$, the generalized fully uncurried form of $f : E \to E'$. First, note that taking $E = E'$ and $f = \mathsf{id}_{E'}$ we have

$$\widehat{\mathsf{id}}_{E'} = \mathsf{app}_{\mathsf{arg}E',\mathsf{res}E'} \circ (\iota_{E'} \times \mathsf{id}_{\mathsf{arg}E'})$$

so that

$$\hat{f} = \widehat{\mathsf{id}}_{E'} \circ (f \times \mathsf{id}_{\mathsf{arg}E'}).$$

Next, we have an equivalent definition of $\hat{f}$

$$\hat{f} = \mathsf{uncurry}(\iota_{E'} \circ f)$$

by 2.1.6. These different presentations of $\hat{f}$ are useful idioms that will be used interchangeably in the sequel. $\hat{f}$ may be thought of as a generalized fully uncurried form of f, and $\widehat{\mathsf{id}}_{E'}$ may be thought of as a generalized application of elements of E' to suitable arguments in the argument space $\mathsf{arg}E'$. In both cases the generalization is that E' need not necessarily be an exponentiation object, and *full* uncurrying refers to the target of $\hat{f}$ being a data domain. This is in fact a conservative extension in the sense that when E' is a function space into a data space then we have[2] $\mathsf{rep}\, E' = E'$ and $\iota_{E'} = \mathsf{id}_{E'}$, and our definitions will essentially reduce to $\hat{f} = \mathsf{uncurry}\, f$ and $\widehat{\mathsf{id}}_{E'} = \mathsf{app}$.

Figure 7.2 illustrates the definition of $\tilde{h}$ for $h : E \to \mathsf{arg}E$, as used in the definition of $\checkmark$ to reduce a request for information about $x \in E$ into a residual request for information about $\tilde{h}x = \iota_E x(hx) \in \mathsf{res}E$. As the figure illustrates, there are several equivalent ways of presenting $\tilde{h}$:

$$\begin{aligned}
\tilde{h} &= \lambda x \in E\,.\,\iota_E x(hx) \\
&= \mathsf{app}_{\mathsf{arg}E,\mathsf{res}E} \circ \langle \iota_E\,, h\rangle \\
&= \widehat{\mathsf{id}}_E \circ \langle \mathsf{id}_E\,, h\rangle\,.
\end{aligned}$$

[2]Modulo the distinction between data domains and data spaces. Strictly speaking, if E' is a data space then E' and $\mathsf{rep}\, E'$ are only isomorphic, via $\iota_{E'} = \lambda x \in E'\,.\,\mathsf{K}x$.

$$\begin{tikzcd}
E \arrow[r, "\langle \mathsf{id}\,, h\rangle"] \arrow[drr, "\tilde{h}"'] & E \times \mathsf{arg}E \arrow[r, "\iota_E \times \mathsf{id}"] \arrow[dr, "\widehat{\mathsf{id}}_E"] & \mathsf{rep}\,E \times \mathsf{arg}E \arrow[d, "\mathsf{app}"] \\
& & \mathsf{res}E
\end{tikzcd}$$

Figure 7.2: The definition of $\tilde{h}$

Again, these are useful idioms that will be used interchangeably in the sequel. Note that $\tilde{h}$ is stable, and inherits strictness from ι.

7.2.5 Proposition *For every space E, the applicative index structure $\mathcal{I}_E$ is a well defined stable index structure.*

Proof: (IS1), (IS2) and (IS3) are inherited directly from the corresponding properties for the primal data index structure $\mathcal{I}_{\mathsf{res}E}$. For (IS1), for instance, we have:

$$\begin{array}{llrll}
 & & [h \Rightarrow^{\mathrm{a}} r] & \checkmark_E & x \\
\text{iff} & & r & \checkmark_{\mathsf{res}E} & \tilde{h}x \\
\text{iff} & \exists s \in \mathbf{Ans}\, r\,. & s & \checkmark_{\mathsf{res}E} & \tilde{h}x \\
\text{iff} & \exists [h \Rightarrow^{\mathrm{a}} s] \in \mathbf{Ans}[h \Rightarrow^{\mathrm{a}} r]\,. & [h \Rightarrow^{\mathrm{a}} s] & \checkmark_E & x.
\end{array}$$

A prompter function $h : E \to \mathsf{arg}E$ is assumed stable, so that $\tilde{h}$ is also stable (as a function between the underlying biposets; we haven't shown $E \times \mathsf{arg}E$ to be a space yet). To establish $\bullet$- and $\star$-monotonicity of $\mathcal{I}_E$ let $x \leq^{\bullet} y$ (which will be the case if $x \leq^{\star} y$), so that:

$$\begin{array}{lrll}
 & [h \Rightarrow^{\mathrm{a}} r_0] & \checkmark & x \\
\text{iff} & r_0 & \checkmark & \tilde{h}x \\
\Longrightarrow & r_0 & \checkmark & \tilde{h}y \\
\text{iff} & [h \Rightarrow^{\mathrm{a}} r_0] & \checkmark & y.
\end{array}$$

To establish $\bullet$- and $\star$-continuity of $\mathcal{I}_E$ let X be an $\bullet$-directed set (which will be the case if X is $\star$-directed, in which case $\bigvee^{\bullet} X = \bigvee^{\star} X$) so that:

$$\begin{array}{llrll}
 & & [h \Rightarrow^{\mathrm{a}} r_0] & \checkmark & (\bigvee^{\bullet} X) \\
\text{iff} & & r_0 & \checkmark & \tilde{h}(\bigvee^{\bullet} X) = \bigvee(\tilde{h}X) \\
\text{iff} & \exists x \in X & r_0 & \checkmark & \tilde{h}x \\
\text{iff} & \exists x \in X & [h \Rightarrow^{\mathrm{a}} r_0] & \checkmark & x.
\end{array}$$

For $\star$-stability let $x \Uparrow^{\star} y$, so that $\tilde{h}x \Uparrow \tilde{h}y$, and:

$$\begin{array}{lrll}
 & [h \Rightarrow^{\mathrm{a}} r_0] & \checkmark & x,\ y \\
\text{iff} & r_0 & \checkmark & \tilde{h}x,\ \tilde{h}y \\
\text{iff} & r_0 & \checkmark & (\tilde{h}x \wedge \tilde{h}y) = \tilde{h}(x \wedge y) \\
\text{iff} & [h \Rightarrow^{\mathrm{a}} r_0] & \checkmark & x \wedge^{\star} y.
\end{array}$$

∎

7.2.6 Remark We imposed stability on fully indexable functions as a means of simplifying the development. It should be possible to weaken the stability requirement and require instead only $\bullet$-continuity and indexability of $\hat{f}$; stability would follow, in analogy with proposition 3.6.2. But doing so would unnecessarily complicate the development, because the proof that the applicative index structure is a stable index structure and that fully indexable functions are stable would need to be a combined inductive proof, interwoven with the other inductive proofs in the sequel. A main component of such a proof, showing that if h is stable then $[h \Rightarrow^{\mathrm{a}} r_0]$ has a stable extent, is presented above, and the main remaining hole — made irrelevant by explicitly requiring stability — is inferring stability of a fully indexable function f from stability of its generalized fully uncurried form $\hat{f}$. $\bullet$

7.2.3 Product of Spaces

$$\begin{tikzcd} E_1 \times E_2 \arrow[r, "\iota_{E_1} \times \iota_{E_2}"] \arrow[dr, "\iota_{E_1 \times E_2}"'] & \mathsf{rep}\, E_1 \times \mathsf{rep}\, E_2 \arrow[d, "(\times)"] \\ & \mathsf{rep}(E_1 \times E_2) \end{tikzcd}$$

Figure 7.3: The applicative embedding for the product

7.2.7 Definition [Part 3: Product of spaces] For a product of spaces E_1 and E_2 we take the biposet product $E_1 \times E_2$, with applicative structure

$$\begin{aligned} \mathsf{arg}(E_1 \times E_2) &= \mathsf{arg} E_1 \times \mathsf{arg} E_2 \\ \mathsf{res}(E_1 \times E_2) &= \mathsf{res} E_1 \times \mathsf{res} E_2 \\ \iota_{E_1 \times E_2} &= (\times) \circ (\iota_{E_1} \times \iota_{E_2}) \\ &= \lambda(x_1, x_2) \in E_1 \times E_2 \,.\, \iota_{E_1} x_1 \times \iota_{E_2} x_2 \\ &= \lambda(x_1, x_2) \in E_1 \times E_2 \,.\, \lambda(y_1, y_2) \in \mathsf{arg} E_1 \times \mathsf{arg} E_2 \,.\, (\iota_{E_1} x_1 y_1, \iota_{E_2} x_2 y_2). \end{aligned}$$

$\bullet$

Figure 7.3 illustrates the definition of the applicative embedding for the product.

The next proposition establishes that spaces are closed under product.

7.2.8 Proposition *Space product is well defined, spaces are closed under product, and*

$$\mathsf{ord}(E_1 \times E_2) = \mathsf{max}\, \{\mathsf{ord}\, E_1, \mathsf{ord}\, E_2\}$$

for all spaces E_1 and E_2.

Proof: By induction on $\beta = \mathsf{max}\,\{\mathsf{ord}\, E_1, \mathsf{ord}\, E_2\}$.

This is easily seen for $\beta = 0$, since the product of data spaces trivially yields a data space.

For $\beta+1$, assuming that the induction hypothesis holds for β, the product of E_1 and E_2 is well defined, since d-bidcpos are closed under product, and by induction hypothesis the product $\mathsf{arg}E_1 \times \mathsf{arg}E_2$ is an order β space.

In both cases, $\iota_{E_1 \times E_2}$ is easily seen to be strict, injective, stable and admissible, since all of these properties are preserved by function product and function composition, and possessed by $(\times)$, ι_{E_1} and ι_{E_2}. (For admissibility see proposition 6.1.14.) ∎

7.2.4 The Fully Indexable Function Space

A definition of the fully indexable function space completes the combined inductive definition.

7.2.9 Definition [Part 4: Fully indexable function space] The ***fully indexable function space*** $E \to_{\mathrm{fx}} E'$ of two spaces E and E' is the collection of fully indexable functions between them, bi-ordered by $\leq^{\bullet\mathrm{p}}$ and $\leq^{\star\mathrm{s}}$, and with the "fully uncurried" applicative structure

$$\begin{array}{rcl} \mathsf{arg}(E \to_{\mathrm{fx}} E') & = & E \times \mathsf{arg}E' \\ \mathsf{res}(E \to_{\mathrm{fx}} E') & = & \mathsf{res}E' \\ \iota_{E \to_{\mathrm{fx}} E'} & = & \lambda f \in E \to_{\mathrm{fx}} E' \,.\, \hat{f}. \end{array}$$

•

$\iota_{E \to_{\mathrm{fx}} E'}$ is the restriction to $E \to_{\mathrm{fx}} E'$ of the generalized full uncurrying operation, denoted by $(\hat{\ })_{E,E'}$ and defined for every function from E to E'. Note that by remarks made above,

$$(\hat{\ })_{E,E'} = (\mathsf{app}\,\circ) \circ (\ \times \mathsf{id}_{\mathsf{arg}E'}) \circ (\iota_{E'} \circ\ ) = \mathsf{uncurry} \circ (\iota_{E'} \circ\ ).$$

In order to show that the underlying object of a fully indexable function space is a d-bidcpo, we would like to show now that the fully indexable functions between spaces are an admissible subset of the stable functions.

7.2.10 Proposition *$\iota_{E \to_{\mathrm{fx}} E'}$ is a well-defined applicative embedding; that is, it is a strict, injective and admissible stable function.*

The fully indexable functions between spaces E and E' are an admissible subset of the stable functions between the spaces.

Proof: Start by looking at $(\hat{\ }) = \mathsf{uncurry} \circ (\iota_{E'} \circ\ )$ restricted to the stable function space $E \to_{\mathrm{bs}} E'$. This is a strict, injective and stable function into the stable function space $E \times \mathsf{arg}E' \to_{\mathrm{bs}} \mathsf{res}E'$. It is admissible, using proposition 6.1.14.

In order to show that the fully indexable functions are an admissible subset it suffices to show that full indexability is preserved by the necessary operations; To do so, note that $(\hat{\ })$ preserves directed lubs and meets with respect to $\leq^{\bullet}$, and consistent joins with respect to $\leq^{\star}$. These operations are all determined pointwise (with respect to the appropriate order) in the stable function space. Moreover, the indexable function space $E \times \mathsf{arg}E' \to_{\mathrm{bx}} \mathsf{res}E'$ is admissible, by 6.3.2; thus, if we restrict $(\hat{\ })$ to the fully indexable function space $E \to_{\mathrm{fx}} E'$, the image $(\hat{\ })(E \to_{\mathrm{fx}} E')$ will be closed under the relevant operations within the larger stable function space. But full indexability of functions in $E \to_{\mathrm{bs}} E'$ is determined by their images under $(\hat{\ })$ (and stability is already assured), and it follows that $E \to_{\mathrm{fx}} E'$ is closed under these operations in the larger stable function space. We may conclude that $E \to_{\mathrm{fx}} E'$ is admissible, and $\iota_{E \to_{\mathrm{fx}} E'}$, since its restriction to $E \to_{\mathrm{fx}} E'$ is indeed a function into $E \times \mathsf{arg}E' \to_{\mathrm{bx}} E'$, is a well defined applicative embedding. ∎

7.2.11 Proposition *The fully indexable function space $E \to_{\mathrm{fx}} E'$ of spaces E and E' is well defined and is a space. Also,*

$$\mathsf{ord}(E \to_{\mathrm{fx}} E') = \max\{1 + \mathsf{ord}\, E, \mathsf{ord}\, E'\}\,.$$

Proof: Since $E \to_{\mathrm{fx}} E'$ is an admissible subset of the stable function space, it follows by 6.1.11 that it is a d-bidcpo when ordered $\bullet$-pointwise and $\star$-stably. $E \times \mathsf{arg}E'$ is a well-defined space so that the applicative structure is well defined. Moreover,

$$\begin{aligned}\mathsf{ord}(E \to_{\mathrm{fx}} E') &= 1 + \mathsf{ord}(E \times \mathsf{arg}E') \\ &= 1 + \max\{\mathsf{ord}\, E, \mathsf{ord}\,\mathsf{arg}E\} = \max\{1 + \mathsf{ord}\, E, \mathsf{ord}\, E'\}\,.\end{aligned}$$

∎

7.2.5 Product First

The main problem posed by the definition of fully indexable functions is the reliance on the product $E \times \mathsf{arg}E'$ in order to determine the indexability of a function $f : E \to E'$ in its uncurried form $\hat{f} : E \times \mathsf{arg}E' \to \mathsf{res}E'$. This feature dominates the structure of the technical development in this chapter.

To illustrate this point, let us sketch a proof that a constant function $\mathsf{K}y : E \to E'$ is fully indexable. Stability is known, and we need to show that

$$\begin{aligned}\widehat{\mathsf{K}y} &= \mathsf{app} \circ ((\iota \circ \mathsf{K}y) \times \mathsf{id}) \\ &= \mathsf{app} \circ (\mathsf{K}(\iota y) \times \mathsf{id}) \\ &= (\iota y) \circ \pi_2\end{aligned}$$

is indexable. ιy is indexable, by definition of ι, but we still cannot vouch for indexability of $\pi_2 : E' \times \mathsf{arg}E' \to \mathsf{arg}E'$. We have thus proved a partial result:

7.2.12 Proposition *If $\pi_2 : E' \times \mathsf{arg}E' \to \mathsf{arg}E'$ is indexable then $\mathsf{K}y : E \to E'$ is a fully indexable function, for every $y \in E'$.*

Nevertheless, one constant function remains constant under $(\hat{\ })$ and will yield at this point:

7.2.13 Proposition *$\mathsf{K}\bot_{E'} : E \to E'$ is fully indexable.*

Proof: $\widehat{\mathsf{K}\bot_{E'}} = \mathsf{K}\bot_{\mathsf{res}E'} : E \times \mathsf{arg}E' \to \mathsf{res}E'$ is indexable, hence $\mathsf{K}\bot_{E'} : E \to E'$ is fully indexable. ■

As another example for the dependence on the product we examine the identity on spaces. Note that there is no a-priori guarantee that the identity on a space is fully indexable. We need to establish the indexability of

$$\begin{aligned} \widehat{\mathsf{id}}_E &: E \times \mathsf{arg}E \to \mathsf{res}E \\ &= \mathsf{app} \circ (\iota_E \times \mathsf{id}) = \mathsf{uncurry}\, \iota_E. \end{aligned}$$

This composition is shown in figure 7.4, and figure 7.5 shows the definition of $(\widetilde{\rotatebox[origin=c]{180}{π}_1 \circ \pi_2})$. Note that $\widehat{\mathsf{id}}_E = \pi_1 \circ (\widetilde{\rotatebox[origin=c]{180}{π}_1 \circ \pi_2})$. Therefore, if we take a primal data index r of $\mathsf{res}E$, we have that

$$\begin{tikzcd}
E \times \mathsf{arg}E \arrow[r, "\iota_E \times \mathsf{id}"] \arrow[dr, "\widehat{\mathsf{id}}_E"'] & \mathsf{rep}\, E \times \mathsf{arg}E \arrow[d, "\mathsf{app}"] \\
 & \mathsf{res}E
\end{tikzcd}$$

Figure 7.4: A diagram of $\widehat{\mathsf{id}}_E$

$$\begin{tikzcd}
E \times \mathsf{arg}E \arrow[r, "{\langle \mathsf{id}\,, \rotatebox[origin=c]{180}{π}_1 \circ \pi_2 \rangle}"] \arrow[drr, "{\widetilde{\rotatebox[origin=c]{180}{π}_1 \circ \pi_2}}"'] & (E \times \mathsf{arg}E) \times \mathsf{arg}(E \times \mathsf{arg}E) \arrow[r, "\iota_{E \times \mathsf{arg}E} \times \mathsf{id}"] \arrow[dr, "\widehat{\mathsf{id}}_{E \times \mathsf{arg}E}"] & \mathsf{rep}(E \times \mathsf{arg}E) \times \mathsf{arg}(E \times \mathsf{arg}E) \arrow[d, "\mathsf{app}"] \\
 & & \mathsf{res}(E \times \mathsf{arg}E)
\end{tikzcd}$$

Figure 7.5: A diagram of $(\widetilde{\rotatebox[origin=c]{180}{π}_1 \circ \pi_2})$

$$\begin{array}{llll}
 & r & \checkmark_{\mathsf{res}E} & \widehat{\mathsf{id}}_E(x, y) \\
\text{iff} & r & \checkmark_{\mathsf{res}E} & \pi_1((\widetilde{\rotatebox[origin=c]{180}{π}_1 \circ \pi_2})(x, y)) \\
\text{iff} & \rotatebox[origin=c]{180}{π}_1\, r & \checkmark_{\mathsf{res}(E \times \mathsf{arg}E)} & (\widetilde{\rotatebox[origin=c]{180}{π}_1 \circ \pi_2})(x, y) \\
\text{iff} & [\rotatebox[origin=c]{180}{π}_1 \circ \pi_2 \Rightarrow^{\mathrm{a}} \rotatebox[origin=c]{180}{π}_1\, r] & \checkmark_{E \times \mathsf{arg}E} & (x, y)
\end{array}$$

so that full indexability of $\rotatebox[origin=c]{180}{π}_1 \circ \pi_2$ would allow us to construct indices showing full indexability of id_E. Thus, we have shown the following proposition:

7.2.14 Proposition *If $\rotatebox[origin=c]{180}{π}_1 \circ \pi_2 : E \times \mathsf{arg}E \to \mathsf{arg}(E \times \mathsf{arg}E)$ is α-fully indexable then id_E is α-fully indexable.*

7.3 Product

We have seen examples illustrating the need to treat the product — show full indexability of morphisms related to the product — prior to proving results that are usually considered more basic, such as full indexability of constant functions or the identity. A similar need arises when we consider, at a later point, the compositionality of fully indexable functions.

We therefore proceed now to prove that space product is a cartesian product in the "category" of spaces and fully indexable functions. The problem, of course, is that we have not shown this a category, and we will have to tread very carefully. In particular, we cannot assume that composition preserves full indexability; rather, a main ingredient of the development in this section is devoted to proving full indexability of certain restricted forms of composition.

7.3.1 Doubling of Product Structure

As a motivation for the ensuing development, we start with an attempt to prove the full indexability of the pairing of fully indexable functions. We suspend the proof part way through, pending further discussion, and encapsulate the remaining hole in the proof as an assumption concerning full indexability of compositions of a certain form. Note that, since we are not using the simple product, we may not rely on indexability of the product morphisms.

We say that assumption (P) holds of spaces E, E_1 and E_2, iff

> For every $\ell \in \{1,2\}$, if $h : E \times E_\ell \to \mathsf{arg}E \times \mathsf{arg}E_\ell$ is an α-fully indexable function, then so is the function
>
> $$\begin{array}{rcl} \Phi^{\ell}_{(E,E_1,E_2)}h & = & (\mathsf{id}_{\mathsf{arg}E} \times \amalg_{\ell;\mathsf{arg}E_1,\mathsf{arg}E_2}) \circ h \circ (\mathsf{id}_E \times \pi_{\ell;E_1,E_2}) \\ & : & E \times (E_1 \times E_2) \to \mathsf{arg}E \times (\mathsf{arg}E_1 \times \mathsf{arg}E_2). \end{array}$$

$$\begin{tikzcd} E \times (E_1 \times E_2) \arrow[r, "\Phi^{\ell} h"] \arrow[d, "\mathsf{id}_E \times \pi_\ell"'] & \mathsf{arg}E \times (\mathsf{arg}E_1 \times \mathsf{arg}E_2) \\ E \times E_\ell \arrow[r, "h"'] & \mathsf{arg}E \times \mathsf{arg}E_\ell \arrow[u, "\mathsf{id}_E \times \amalg_\ell"'] \end{tikzcd}$$

7.3.1 Proposition *For all spaces E, E_1 and E_2, if (P) holds of E, $\mathsf{arg}E_1$ and $\mathsf{arg}E_2$ then for every two fully indexable functions $f_1 : E \to E_1$ and $f_2 : E \to E_2$, the pairing $\langle f_1, f_2\rangle : E \to E_1 \times E_2$ is fully indexable. Moreover,* $\mathsf{level}\,\langle f_1, f_2\rangle = \max\{\mathsf{level}\,f_1, \mathsf{level}\,f_2\}$.

Proof: Let $f = \langle f_1, f_2\rangle$. To show full indexability of f we need to show indexability of $\hat{f} : E \times (\mathsf{arg}E_1 \times \mathsf{arg}E_2) \to \mathsf{res}E_1 \times \mathsf{res}E_2$. Assume that $s_0 \in \mathcal{I}(\hat{f}; z, Z)$ for $z \in E \times (\mathsf{arg}E_1 \times \mathsf{arg}E_2)$ and non-empty finite $Z \subseteq \uparrow^{\star} z$.

By properties of primal data indices over products (proposition 4.3.1), s_0 projects onto an index in one of the components of the product $\mathsf{res}E_1 \times \mathsf{res}E_2$ of data spaces. Let

$\ell \in \{1,2\}$ be that component, so that we have $r_0 = \pi_\ell\, s_0$ for some $r_0 \in \mathcal{I}_{\mathsf{res}E_\ell}$, and $r_0 \in \mathcal{I}(\pi_\ell \circ \hat{f}; z, Z)$. Note now that

$$\pi_\ell \circ \hat{f} = \hat{f}_\ell \circ (\mathsf{id} \times \pi_\ell),$$

so that $r_0 \in \mathcal{I}(\hat{f}_\ell; z_\ell, Z_\ell)$, where $z_\ell = (id \times \pi_\ell)z$ and $Z_\ell = (id \times \pi_\ell)Z$. Therefore, by full indexability of f_ℓ, there exists $r \in \mathcal{I}(z_\ell, Z_\ell)$.

How can we obtain from this $r \in \mathcal{I}_{E \times E_\ell}$ the desired index of $\hat{f}$ for $s_0 \in \mathcal{I}(\hat{f}; z, Z)$? We would like to be able to "inject" r into $E \times (\mathsf{arg}E_1 \times \mathsf{arg}E_2)$ with a trivial $\mathsf{arg}E_{(3-\ell)}$ component, since this component cannot contribute to an increase of the value of $\hat{f}$ in the $\mathsf{res}E^\ell$ component of its output. If we can find an index s of $E \times (E_1 \times E_2)$ such that

$$(s \checkmark\ ) = (\mathsf{id}_E \times \amalg_\ell)(r \checkmark\ )$$

then it is easy to see that s is an index as desired:

$$\begin{array}{rrcl} & s_0 & \checkmark & \hat{f} z' \\ \Longrightarrow & s_0 & \checkmark & \hat{f}_\ell((\mathsf{id} \times \pi_\ell)z') \\ \Longrightarrow & r & \checkmark & ((\mathsf{id} \times \pi_\ell)z') \\ \Longrightarrow & s & \checkmark & z'. \end{array}$$

Does there exist such an s? The operation of obtaining s from r via an injection of its extent has already been justified for primal data indices (proposition 4.3.1), but to show that this may be done for applicative indices we need to present s as an applicative index. To do so, assume that $r = [h \Rightarrow^{\mathrm{a}} r_{00}]$, for $h : E \times \mathsf{arg}E_\ell \to \mathsf{arg}E \times \mathsf{arg}(\mathsf{arg}E_\ell)$ α-fully indexable and an appropriate r_{00}. We now claim that setting

$$s = [\Phi^\ell_{(E,\mathsf{arg}E_1,\mathsf{arg}E_2)} h \Rightarrow^{\mathrm{a}} (\mathsf{id}_{\mathsf{res}E} \times \amalg_{\ell;\mathsf{res}(\mathsf{arg}E_1),\mathsf{res}(\mathsf{arg}E_2)}) r_{00}]$$

yields an index s with extent as desired:

$$\begin{array}{lrcl} & s & \checkmark & z' \\ \text{iff} & (\mathsf{id} \times \amalg_\ell) r_{00} & \checkmark & \widetilde{\Phi^\ell h} z' \\ \text{iff} & r_{00} & \checkmark & (\mathsf{id} \times \pi_\ell)(\widetilde{\Phi^\ell h z'}) \\ \text{iff} & r_{00} & \checkmark & \tilde{h}((\mathsf{id} \times \pi_\ell)z') \\ \text{iff} & r & \checkmark & (\mathsf{id} \times \pi_\ell)z'. \end{array}$$

(Using the equality $(\mathsf{id} \times \pi_\ell) \circ \widehat{\mathsf{id}}_{E \times (E_1 \times E_2)} = \widehat{\mathsf{id}}_{E \times E_\ell} \circ ((\mathsf{id} \times \pi_\ell) \times (\mathsf{id} \times \pi_\ell))$.)

But does this define a valid applicative index? It is not difficult to see that $(\mathsf{id} \times \amalg_\ell) r_{00}$ is a primal data index over the product of data domains. It remains to be shown, then, that $\Phi^\ell h$ is fully indexable at some finite level. But this is exactly what our assumption — that (P) holds of E, $\mathsf{arg}E_1$ and $\mathsf{arg}E_2$ — allows us to assume. (More precisely, at this point we choose to suspend the proof, pending further discussion, by assuming the "hole" in proof.)

Finally, our proof of full indexability of f used indices at the same level as indices used to show f_1 and f_2 fully indexable, so that $\mathsf{level}\, f = \mathsf{max}\,\{\mathsf{level}\, f_1, \mathsf{level}\, f_2\}$. ∎

Let us now turn our attention to proving (P) for general E, E_1 and E_2. We try to explain now the idea behind the proof, before the details and the notation get in our way. Assume that for some ℓ and α-fully indexable function h we want to show that

$$\phi = \Phi^{\ell} h = (\mathsf{id} \times \rotatebox{180}{π}_{\ell}) \circ h \circ (\mathsf{id} \times \pi_{\ell})$$

is α-fully indexable, as we would need to do in order to discharge the assumption of (P). When we try to do so, we essentially end up trying to show for some α'-fully indexable h', derived from some index for $\hat{h}$, that

$$\phi' = ((\mathsf{id} \times \rotatebox{180}{π}_{\ell}) \times (\mathsf{id} \times \rotatebox{180}{π}_{\ell})) \circ h' \circ ((\mathsf{id} \times \pi_{\ell}) \times (\mathsf{id} \times \pi_{\ell}))$$

is α'-fully indexable. Just to carry this one step further, when we try to prove the full indexability of ϕ', we would end up attempting to show, for some h'', derived from some index for $\widehat{h'}$, that

$$\begin{aligned} \phi'' \quad = \quad & (((\mathsf{id} \times \rotatebox{180}{π}_{\ell}) \times (\mathsf{id} \times \rotatebox{180}{π}_{\ell})) \times ((\mathsf{id} \times \rotatebox{180}{π}_{\ell}) \times (\mathsf{id} \times \rotatebox{180}{π}_{\ell}))) \circ h'' \circ \\ & \circ(((\mathsf{id} \times \pi_{\ell}) \times (\mathsf{id} \times \pi_{\ell})) \times ((\mathsf{id} \times \pi_{\ell}) \times (\mathsf{id} \times \pi_{\ell}))) \end{aligned}$$

is fully indexable.

There is a process of "structural doubling" going on here. In schematic, informal terms, if $h : E \to \mathsf{arg} E$ then $\hat{h} : E \times \mathsf{arg}(\mathsf{arg} E) \to \mathsf{res}(\mathsf{arg} E)$. Now, h', being drawn from an index for $\hat{h}$, will be of type $h' : E \times \mathsf{arg}(\mathsf{arg} E) \to \mathsf{arg}(E \times \mathsf{arg}(\mathsf{arg} E))$. Recall that arg distributes over product, so that the product structure of $E \times \mathsf{arg}(\mathsf{arg} E)$ "doubles" the product structure of E. More precisely, if we think of E as being a binary tree with product at each internal node, then $E \times \mathsf{arg}(\mathsf{arg} E)$ consists of two copies of this tree under a new root (modulo applications of arg at the leaves). Analogously, if we want to inject, in some way, an index for h to obtain an applicative index for ϕ, we must also "double" the generalized projections and injections that we compose with h', as reflected in the definition of ϕ'.

Are we faced then with some infinite regression that precludes us from establishing full indexability of ϕ? No: the solution is provided by the indexable hierarchy, when we observe that we may choose h', and then h'', so that

$$\mathsf{level}\, h'' < \mathsf{level}\, h' < \mathsf{level}\, h.$$

That is, at each step in this process we increase the complexity of the product structure, but we can decrease the level in the indexable hierarchy. Thus, by the well foundedness of the indexable hierarchy, the progression will only be finite. The base case is easy to deal with: if h is 0-fully indexable then so is ϕ, because $\hat{h}$ and $\hat{\phi}$ must both be constant functions.

Once we have made this observation, we need an appropriate formalism for talking about the doubling of product structure. With it in place, we will be able to formulate a sufficient generalization of (P), to be proved by induction on the level in the indexable hierarchy.

7.3.2 Decompositions

We give here a definition and some examples of decompositions, specialized to the category PSt of spaces and stable functions.

7.3.2 Definition In the following definitions take $k \geq 1$, let $\bar{E}$ range over k-tuples of spaces, and let $\bar{B}$ range over k-tuples of data domains.

A k-ary ***productor*** over spaces is a k-ary operation S on spaces, together with a k-ary operation S on data domains, such that

(D1a) S distributes over arg: $\mathsf{arg}(S\bar{E}) = S(\mathsf{arg}\bar{E})$.

(D1b) S distributes over res: $\mathsf{res}(S\bar{E}) = S(\mathsf{res}\bar{E})$.

(D1c) Data spaces are closed under S: $S\bar{E}$ is a data space if $\bar{E}$ is a k-tuple of data spaces.

where $\mathsf{arg}\bar{E}$ and $\mathsf{res}\bar{E}$ refer to componentwise application of arg or res to the elements of the tuple.

Instead of (D1b) and (D1c) we could have taken S to be an operation over spaces that may be uniquely extended to data domains by setting $S\bar{B} = \mathsf{res}(S\bar{E})$ for any k-tuple $\bar{B}$ of data domains and k-tuple $\bar{E}$ of data spaces such that $\mathsf{res}\bar{E} = \bar{B}$.

A ***component*** of a k-ary productor S is a k-ary productor R together with transformations $\rho : S \to R$ and $\sigma : R \to S$ over PST and, separately, over DATA (*i.e.*, for every k-tuple of spaces $\bar{E}$, $\rho_{\bar{E}}$ is a stable function from $\bar{S}$ to $\bar{R}$, and likewise for every k-tuple of data domains, and for σ) such that

(D2a) $R\bar{E} \trianglelefteq^{\sigma_{\bar{E}}}_{\rho_{\bar{E}}} S\bar{E}$ in PST, for every $\bar{E}$, that is,

- $\rho_{\bar{E}} : S\bar{E} \to R\bar{E}$ and $\sigma_{\bar{E}} : R\bar{E} \to S\bar{E}$ are stable, and
- $\rho_{\bar{E}} \circ \sigma_{\bar{E}} = \mathsf{id}_{\bar{E}}$, and
- $\sigma_{\bar{E}} \circ \rho_{\bar{E}} \leq^{\star} \mathsf{id}_{\bar{E}}$.

And likewise for every k-tuple $\bar{B}$ of data spaces.

(D2b) ρ distributes over $\widehat{\mathsf{id}}$, in the following sense:

$$\rho_{\mathsf{res}\bar{E}} \circ \widehat{\mathsf{id}}_{S\bar{E}} = \widehat{\mathsf{id}}_{R\bar{E}} \circ (\rho_{\bar{E}} \times \rho_{\mathsf{arg}\bar{E}}).$$

An equivalent formulation of this condition is:

For all $x \in S\bar{E}$ and $y \in S(\mathsf{arg}\bar{E})$, $\rho_{\mathsf{res}\bar{E}}(\iota_{S\bar{E}}xy) = \iota_{R\bar{E}}(\rho_{\bar{E}}x)(\rho_{\mathsf{arg}\bar{E}}y)$.

The equivalence follows from easy manipulation of the definition of $\widehat{\mathsf{id}}$:

$$\begin{array}{llrcl} & & \rho \circ \widehat{\mathsf{id}} & = & \widehat{\mathsf{id}} \circ (\rho \times \rho) \\ \text{iff} & \forall x, y\,. & \rho(\widehat{\mathsf{id}}(x, y)) & = & \widehat{\mathsf{id}}(\rho x, \rho y) \\ \text{iff} & \forall x, y\,. & \rho(\iota x y) & = & \iota(\rho x)(\rho y). \end{array}$$

(D2c) Injection of data space indices: if $r_0 \in \mathcal{I}_{R\bar{B}}$ then $\sigma r_0 \in \mathcal{I}_{S\bar{B}}$ and for every $x \in S\bar{B}$, $\sigma r_0 \checkmark x$ iff $r_0 \checkmark \rho x$.

$$\begin{tikzcd}
 & S\bar{E} \arrow[dl, bend left, "\rho_{\bar{E}}"] \arrow[dr, bend left, "\rho'_{\bar{E}}"] & \\
R\bar{E} \arrow[ur, bend left, "\sigma_{\bar{E}}"] & & R'\bar{E} \arrow[ul, bend left, "\sigma'_{\bar{E}}"]
\end{tikzcd}$$

Figure 7.6: A decomposition $\Xi = (R, R', S)$ at the tuple $\bar{E}$

We may call ρ a *(generalized product) projection*, and σ a *(generalized product) injection*. We write $R \trianglelefteq^{\sigma}_{\rho} S$ when R is a component of S via ρ and σ.

A k-ary ***decomposition*** $\Xi = (R, R', S)$ over PST consists of three k-ary productors R, R' and S and associated transformations such that R and R' are components of S ($R \trianglelefteq^{\sigma}_{\rho} S$ and $R' \trianglelefteq^{\sigma'}_{\rho'} S$), and, additionally,

(D3a) The components are disjoint: $\rho'_{\bar{E}} \circ \sigma_{\bar{E}} = \mathsf{K}\bot_{R'\bar{E}}$ and $\rho_{\bar{E}} \circ \sigma'_{\bar{E}} = \mathsf{K}\bot_{R\bar{E}}$.

(D3b) Inducement of data space indices: if $s_0 \in \mathcal{I}_{S\bar{B}}$ then either there exists $r_0 \in \mathcal{I}_{R\bar{B}}$ with $s_0 = \sigma r_0$ or there exists $r_0 \in \mathcal{I}_{R'\bar{B}}$ with $s_0 = \sigma' r_0$.

Note that if $s_0 = \sigma r_0$ then $\rho s_0 = \rho(\sigma r_0) = r_0$, so the above is equivalent to saying that, either $\rho s_0 \in \mathcal{I}_{R\bar{B}}$ and $s_0 = \sigma(\rho s_0)$, or $\rho' s_0 \in \mathcal{I}_{R'\bar{B}}$ and $s_0 = \sigma'(\rho' s_0)$.

•

We present now some basic decompositions — think of these, or other derived decompositions, as being at the leaves of the tree structure described above. In addition, we show that decompositions are closed under doubling. Thus decompositions are suitable for expressing the inductive doubling process we have described.

7.3.3 Proposition *Space product is a binary decomposition* PR *over* PST*, defined for every* E_1 *and* E_2 *by*

$$\begin{array}{rcl@{\qquad}rcl@{\qquad}rcl}
S(E_1, E_2) & = & E_1 \times E_2 & & & & & & \\
R(E_1, E_2) & = & E_1 & \rho_{(E_1,E_2)} & = & \pi_{1;E_1,E_2} & \sigma_{(E_1,E_2)} & = & \iota_{1;E_1,E_2} \\
R'(E_1, E_2) & = & E_2 & \rho'_{(E_1,E_2)} & = & \pi_{2;E_1,E_2} & \sigma'_{(E_1,E_2)} & = & \iota_{2;E_1,E_2}\,.
\end{array}$$

A trivial[3] *unary identity decomposition* ID *is given by taking* $SE = RE = E$, $R'E = \mathbf{1}$, $\sigma_E = \rho_E = \mathsf{id}_E$, *and* $\sigma'_E = \mathsf{K}\bot_E$, $\rho'_E = \mathsf{K}\bot_{\mathbf{1}}$.

A ternary associative decomposition ASSOC *is given by taking*

$$\begin{array}{l@{\qquad}l@{\qquad}l}
S(E_1, E_2, E_3) = (E_1 \times E_2) \times E_3 & & \\
R(E_1, E_2, E_3) = E_1 \times (E_2 \times E3) & \rho_{(E_1,E_2,E_3)} = \mathsf{assr} & \sigma_{(E_1,E_2,E_3)} = \mathsf{assl} \\
R'(E_1, E_2, E_3) = \mathbf{1} & \rho'_{(E_1,E_2,E_3)} = \mathsf{K}\bot_{\mathbf{1}} & \sigma'_{(E_1,E_2,E_3)} = \mathsf{K}\bot_{(E_1 \times E_2) \times E_3}\,,
\end{array}$$

[3] *The identity and associative decompositions are trivial in their* R' *component, and actually express isomorphisms, since one may exchange the role of* S *and* R*, but this formulation makes it possible to treat them within the general framework of decompositions.*

with the associative isomorphisms

$$\begin{array}{rcl} \mathsf{assl} & = & \langle \langle \pi_1 \,, \pi_1 \circ \pi_2 \rangle \,, \pi_2 \circ \pi_2 \rangle \\ \mathsf{assr} & = & \langle \pi_1 \circ \pi_1 \,, \langle \pi_2 \circ \pi_1 \,, \pi_2 \rangle \rangle \,. \end{array}$$

Proof: Details omitted. For injection and projection of primal data indices rely on proposition 4.3.1. ∎

Given a k_1-ary decomposition $\Xi_1 = (R_1, R_1', S_1)$ and a k_2-ary decomposition $\Xi_2 = (R_2, R_2', S_2)$, we define their product $\Xi_1 \times \Xi_2$ to be the $(k_1 + k_2)$-ary decomposition $\Xi = (R, R', S)$ as defined below. Set $k = k_1 + k_2$. Given a k-tuple $\bar{E}$ it may clearly be uniquely decomposed into $\bar{E}_1$, the k_1-tuple of its leftmost elements, with their order preserved, and $\bar{E}_2$, the k_2-tuple of its rightmost elements, with their order preserved. As a general notation, we write $(\bar{E}_1 \,; \bar{E}_2)$ for the concatenation of the two tuples, which in this case gives back $\bar{E}$. The product of the decompositions is now defined at $\bar{E}$ — essentially componentwise — as follows:

$$\begin{array}{rclcrclcrcl} S\bar{E} & = & (S_1\bar{E}_1) \times (S_2\bar{E}_2) & & & & & & & & \\ R\bar{E} & = & (R_1\bar{E}_1) \times (R_2\bar{E}_2) & & \rho_{\bar{E}} & = & (\rho_{1\bar{E}_1}) \times (\rho_{2\bar{E}_2}) & & \sigma_{\bar{E}} & = & (\sigma_{1\bar{E}_1}) \times (\sigma_{2\bar{E}_2}) \\ R'\bar{E} & = & (R_1'\bar{E}_1) \times (R_2'\bar{E}_2) & & \rho'_{\bar{E}} & = & (\rho'_{1\bar{E}_1}) \times (\rho'_{2\bar{E}_2}) & & \sigma'_{\bar{E}} & = & (\sigma'_{1\bar{E}_1}) \times (\sigma'_{2\bar{E}_2}). \end{array}$$

7.3.4 Proposition *Product of decompositions is well defined: if Ξ_1 and Ξ_2 are decompositions then so is $\Xi_1 \times \Xi_2$.*

Proof: Details omitted. We only present a representative case, showing that ρ distributes over $\widehat{\mathsf{id}}$: for every $x = (x_1, x_2) \in S\bar{E} = (S_1\bar{E}_1) \times (S_2\bar{E}_2)$ and $y = (y_1, y_2) \in S(\mathsf{arg}\bar{E}) = S_1(\mathsf{arg}\bar{E}_1) \times S_2(\mathsf{arg}\bar{E}_2)$,

$$\begin{array}{rcl} \rho(\widehat{\mathsf{id}}(x, y)) & = & (\rho_1 \times \rho_2)(\iota_\times(x_1, x_2)(y_1, y_2)) \\ & = & (\rho_1 \times \rho_2)((\times)((\iota_1 \times \iota_2)(x_1, x_2))(y_1, y_2)) \\ & = & (\rho_1 \times \rho_2)(\iota_1 x_1 y_1, \iota_2 x_2 y_2) \\ & = & (\rho_1(\iota_1 x_1 y_1), \rho_2(\iota_2 x_2 y_2)) \\ & = & (\iota_1(\rho_1 x_1)(\rho_1 y_1), \iota_2(\rho_2 x_2)(\rho_2 y_2)) \\ & = & (\times)((\iota_1 \times \iota_2)(\rho_1 x_1, \rho_2 x_2))(\rho_1 y_1, \rho_2 y_2) \\ & = & \iota_\times(\rho x)(\rho y) \\ & = & \widehat{\mathsf{id}}((\rho \times \rho)(x, y)). \end{array}$$

(We use here a straightforward convention for partially naming the ι's.) ∎

The ***doubling*** operation is defined as a special case of decomposition product. For a k-ary decomposition Ξ, we let its doubling Ξ^2 be the $2k$-ary decomposition $\Xi \times \Xi$. This is precisely the doubling operation we have alluded to in our discussion.

Another, almost trivial, operation on decompositions is ***commuting***, which arises from the simple observation that, thanks to the symmetric nature of the definitions, if S may be decomposed into R and R' then it may also be decomposed into R' and R. Thus, if $\Xi = (R, R', S)$ is a decomposition, we let its commuting, Ξ', be the decomposition (R', R, S). This will let us phrase some assertions in what seems to be an asymmetric fashion; the symmetry is restored by applying the assertion to the commuting of the original decomposition. Note that $(\Xi_1 \times \Xi_2)' = \Xi_1' \times \Xi_2'$.

To relate this development to our motivation, and to present an example of somewhat less trivial decompositions, consider the ternary decompositions

$$\begin{aligned} \Xi_1 &= \mathsf{ID} \times \mathsf{PR} \\ \Xi_2 &= \mathsf{ID} \times (\mathsf{PR}'). \end{aligned}$$

These decompositions, and their doublings, are the ones featured in the course of our attempted proof for the full indexability of the pairing of fully indexable functions[4]. (Ξ_1 for $\ell = 1$, and Ξ_2 for $\ell = 2$, with ℓ as used there.)

7.3.3 The Injective and Projective Lemmas

Now that we have our tools in place, we are able to formalize the inductive process, and prove that it terminates. At first we give an injective lemma for product indices, that shows that the injective property lifts from indices of (products of) data spaces to indices of (generalized products of) arbitrary spaces. We simultaneously prove full indexability of certain restricted forms of composition of fully indexable functions with decomposition projections and injections — even though these maps have not been stipulated to be fully indexable themselves, only to respect the structure as outlined above.

Given a k-ary decomposition $\Xi = (R, R', S)$, we define the ***injection*** of an applicative index $[g \Rightarrow^{\mathrm{a}} r_0] \in \mathcal{I}_{R\bar{E}}$ into $S\bar{E}$ to be

$$\sigma_{\bar{E}}[g \Rightarrow^{\mathrm{a}} r_0] = [\sigma_{\mathsf{arg}\bar{E}} \circ g \circ \rho_{\bar{E}} \Rightarrow^{\mathrm{a}} \sigma_{\mathsf{res}\bar{E}} r_0].$$

The next lemma shows that this in fact produces a valid applicative index.

7.3.5 Lemma (Injection of Product Indices) *For every α, $k \geq 1$, k-ary decomposition $\Xi = (R, R', S)$ over* PST, *k-tuples $\bar{E}$ and $\bar{F}$ of spaces,*

(A) If $r \in \mathcal{I}^{\alpha}_{R\bar{E}}$ then $\sigma_{\bar{E}} r \in \mathcal{I}^{\alpha}_{S\bar{E}}$ and for every $x \in S\bar{E}$,

$$r \checkmark \rho x \qquad \text{iff} \qquad \sigma r \checkmark x.$$

(B) If $g : R\bar{E} \to R\bar{F}$ is α-fully indexable then so is

$$\sigma_{\bar{F}} \circ g \circ \rho_{\bar{E}} : S\bar{E} \to S\bar{F}.$$

Proof: We prove (A) and (B) simultaneously by induction on α.

[4] Strictly speaking, $\mathsf{ID} \times \mathsf{PR}$ decomposes $E \times (E_1 \times E_2)$ into $E \times E_1$ and $\mathbf{1} \times E_2$. We disregard here the difference between $\mathbf{1} \times E_2$ and E_2; it should be clear that this does not pose a problem. For full rigor, it is easy to prove directly that $E \times (E_1 \times E_2)$ may be decomposed into $E \times E_1$ and E_2, and similarly for the other variants needed.

(A) Note that $r = [g \Rightarrow^{\mathrm{a}} r_0]$ for some $g : R\bar{E} \to R(\mathsf{arg}\,\bar{E})$ fully indexable with $\mathsf{level}\, g < \alpha$, and $r_0 \in \mathcal{I}_{R(\mathsf{res}\bar{E})}$.

Since $\mathsf{level}\, g < \alpha$, induction hypothesis (B) applies, and thus, if we let h be

$$h = \sigma_{\mathsf{arg}\bar{E}} \circ g \circ \rho_{\bar{E}} : S\bar{E} \to S(\mathsf{arg}\bar{E})$$

then h is fully indexable, and $\mathsf{level}\, h = \mathsf{level}\, g < \alpha$. Since σr_0 is an index of $\mathsf{res}(S\bar{E})$ by (D2c), σr is a valid α-index of $S\bar{E}$.

Clearly, $\rho_{\mathsf{arg}\bar{E}} \circ h = g \circ \rho_{\bar{E}}$, and from this we get

$$\rho_{\mathsf{res}\bar{E}} \circ \tilde{h} = \tilde{g} \circ \rho_{\bar{E}}, \tag{7.1}$$

by the following development:

$$\begin{array}{rcll}
\rho \circ \tilde{h} & = & \rho \circ \widehat{\mathsf{id}} \circ \langle \mathsf{id}\,, h\rangle & \\
& = & \widehat{\mathsf{id}} \circ (\rho \times \rho) \circ \langle \mathsf{id}\,, h\rangle & ((\text{D2b}):\ \rho \circ \widehat{\mathsf{id}} = \widehat{\mathsf{id}} \circ (\rho \times \rho)) \\
& = & \widehat{\mathsf{id}} \circ \langle \rho\,, \rho \circ h\rangle & \\
& = & \widehat{\mathsf{id}} \circ \langle \rho\,, g \circ \rho\rangle & (\rho \circ h = g \circ \rho) \\
& = & \widehat{\mathsf{id}} \circ \langle \mathsf{id}\,, g\rangle \circ \rho & \\
& = & \tilde{g} \circ \rho. &
\end{array}$$

Finally, the desired relation between the extents of r and σr is shown by the following chain of equivalences, for every $z \in S\bar{E}$:

$$\begin{array}{lrcll}
 & [g \Rightarrow^{\mathrm{a}} r_0] & \checkmark & \rho z & \\
\text{iff} & r_0 & \checkmark & \tilde{g}(\rho z) & \\
\text{iff} & r_0 & \checkmark & \rho(\tilde{h} z) & \text{(Equation 7.1)} \\
\text{iff} & \sigma r_0 & \checkmark & \tilde{h} z & \text{(by (D2c))} \\
\text{iff} & [h \Rightarrow^{\mathrm{a}} \sigma r_0] & \checkmark & z. &
\end{array}$$

(B) Let $h = \sigma_{\bar{F}} \circ g \circ \rho_{\bar{E}}$. We need to show α-indexability of

$$\hat{h} : S\bar{E} \times S(\mathsf{arg}\bar{F}) \to S(\mathsf{res}\bar{F}).$$

Note that

$$\rho_{\mathsf{res}\bar{F}} \circ \hat{h} = \hat{g} \circ (\rho_{\bar{E}} \times \rho_{\mathsf{arg}\bar{F}}), \tag{7.2}$$

by the following development:

$$\begin{array}{rcll}
\rho \circ \hat{h} & = & \rho \circ \widehat{\mathsf{id}} \circ (h \times \mathsf{id}) & \\
& = & \widehat{\mathsf{id}} \circ (\rho \times \rho) \circ (h \times \mathsf{id}) & ((\text{D2b}):\ \rho \circ \widehat{\mathsf{id}} = \widehat{\mathsf{id}} \circ (\rho \times \rho)) \\
& = & \widehat{\mathsf{id}} \circ ((\rho \circ h) \times \rho) & \\
& = & \widehat{\mathsf{id}} \circ ((g \circ \rho) \times \rho) & (\rho \circ h = g \circ \rho) \\
& = & \widehat{\mathsf{id}} \circ (g \times \mathsf{id}) \circ (\rho \times \rho) & \\
& = & \hat{g} \circ (\rho \times \rho). &
\end{array}$$

On the other hand, if we project to the other component, we get bottom,

$$\rho'_{\mathsf{res}\bar{F}} \circ \hat{h} = \mathsf{K}\bot_{R'(\mathsf{res}\bar{F})}, \tag{7.3}$$

as shown by:

$$\begin{aligned}
\rho' \circ \hat{h} &= \rho' \circ \widehat{\mathsf{id}} \circ (h \times \mathsf{id}) && \text{(definition)}\\
&= \widehat{\mathsf{id}} \circ (\rho' \times \rho') \circ (h \times \mathsf{id}) && \text{((D2b): } \rho' \circ \widehat{\mathsf{id}} = \widehat{\mathsf{id}} \circ (\rho' \times \rho'))\\
&= \widehat{\mathsf{id}} \circ ((\rho' \circ \sigma \circ g \circ \rho) \times \rho') &&\\
&= \widehat{\mathsf{id}} \circ (((\mathsf{K}\bot_{R'\bar{F}}) \circ g \circ \rho) \times \rho') && \text{((D3a): } \rho' \circ \sigma = \mathsf{K}\bot)\\
&= \mathsf{K}\bot_{R'(\mathsf{res}\bar{F})} && (\widehat{\mathsf{id}} \text{ left-strict}).
\end{aligned}$$

Choose now an arbitrary $s_0 \in \mathcal{I}(\hat{h}; z, Z)$ for $z \in S\bar{E} \times S(\mathsf{arg}\bar{F})$ and a non-empty and finite $Z \subseteq \uparrow^{\star} z$. Since s_0 is a data space index, it must by (D3b) be induced by an index in one of the components[5]: $s_0 = \sigma r_0$ for some $r_0 \in \mathcal{I}_{R(\mathsf{res}\bar{F})}$, or $s_0 = \sigma' r'_0$ for some $r'_0 \in \mathcal{I}_{R'(\mathsf{res}\bar{F})}$.

Assume that the latter is the case. Then, for every $z' \in S\bar{E} \times S(\mathsf{arg}\bar{F})$, $s_0 \checkmark \hat{h}z'$ iff $r'_0 \checkmark \rho'(\hat{h}z') = \bot_{R'(\mathsf{res}\bar{F})}$, by equation 7.3. But this contradicts $s_0 \in \mathcal{I}(\hat{h}; z, Z)$.

It must therefore be the case that $s_0 = \sigma r_0$ for $r_0 \in \mathcal{I}_{R(\mathsf{res}\bar{F})}$. Thus, $r_0 \in \mathcal{I}(\rho \circ \hat{h}; z, Z)$, and by 7.2 $r_0 \in \mathcal{I}(\hat{g} \circ (\rho \times \rho); z, Z)$. By α-full indexability of g, there exists an α-index $r \in \mathcal{I}(\rho \times \rho; z, Z)$, and by (A), applied to the doubled decomposition $\Xi \times \Xi$ and the $2k$-tuple $(\bar{E}\,; \mathsf{arg}\bar{F})$, we have that $(\sigma \times \sigma)r \in \mathcal{I}(z, Z)$, an α-index. This establishes α-full indexability of h. ∎

While the injectiveness of indices lifts unchanged from data spaces — it is always possible to create an index over the product given an index in one of the components — projection of indices generalizes in a more restricted way. The product assumption ceases to hold over functional spaces; examples of indices over a product that are not induced by, and do not project to an index in any one of the components have been given during previous discussion of the inadequacy of the product assumption at higher order. For instance, the index that establishes indexability of the application: $[\amalg_1 \circ \pi_2 \Rightarrow^{\mathrm{a}} \amalg_1 r_0] \checkmark_{(E\to B)\times E} (f, x)$ iff $r_0 \checkmark_B fx$. (Subject to our showing that this is indeed a valid index. For simplicity we take here B a data domain and ignore an extra trivial component in $\mathsf{arg}(E \to B) = E \times \mathbf{1}$.)

Thus, we need to find out which indices of a product do project into an index of one of the components. The answer, roughly, is that this is the case when the other component of the index is trivial, as formalized in the next lemma.

Other than the extra care that is needed to specify when a projection is allowed, the statement and proofs for the projective case are very similar to those in the injective case.

Given a k-ary decomposition $\Xi = (R, R', S)$, an applicative index $s = [h \Rightarrow^{\mathrm{a}} s_0] \in \mathcal{I}_{S\bar{E}}$ is said to be ***trivial in the*** R' ***component*** iff there exist $x', x'' \in R\bar{E}$ such that $s \checkmark \sigma x'$ and $s \not\checkmark \sigma x''$; we define the ***projection*** of $[h \Rightarrow^{\mathrm{a}} s_0] \in \mathcal{I}_{S\bar{E}}$ into $R\bar{E}$ to be

$$\rho_{\bar{E}}[h \Rightarrow^{\mathrm{a}} s_0] = [\rho_{\mathsf{arg}\bar{E}} \circ h \circ \sigma_{\bar{E}} \Rightarrow^{\mathrm{a}} \rho_{\mathsf{res}\bar{E}} s_0].$$

[5] Note that this proof requires that an index over the product of data spaces be induced by an index in one of the components; it is not sufficient for an index over the product to project to one of the components. This is the technical reason for employing the primal data index structures on data domains.

The following lemma shows that this in fact produces a valid applicative index whenever $[h \Rightarrow^{\mathrm{a}} s_0]$ is trivial in R'.

7.3.6 Lemma (Projection of Product Indices) *For every α, $k \geq 1$, k-ary decomposition system (R, R', S) over* PST*, k-tuples $\bar{E}$ and $\bar{F}$ of spaces,*

(A) If $s \in \mathcal{I}^{\alpha}_{S\bar{E}}$ is trivial in the R' component then $\rho s \in \mathcal{I}^{\alpha}_{R\bar{E}}$ and for every $x \in R\bar{E}$,

$$\rho s \checkmark x \qquad \text{iff} \qquad s \checkmark \sigma x.$$

(B) If $h : S\bar{E} \to S\bar{F}$ is α-fully indexable then so is

$$\rho_{\bar{F}} \circ h \circ \sigma_{\bar{E}} : R\bar{E} \to R\bar{F}.$$

Proof: We prove (A) and (B) simultaneously by induction on α.

(A) Note that $s = [h \Rightarrow^{\mathrm{a}} s_0]$ for some $h : S\bar{E} \to S(\mathsf{arg}\,\bar{E})$ fully indexable with $\mathsf{level}\, h < \alpha$, and $s_0 \in \mathcal{I}_{S(\mathsf{res}\bar{E})}$.

Assume that s_0 projects into $\rho' s_0 \in \mathcal{I}_{R'(\mathsf{res}\bar{E})}$. But then for every $x \in R\bar{E}$, $s \checkmark \sigma x$ iff $s_0 \checkmark \tilde{h}(\sigma x)$, which in turn is the case iff $\rho' s_0 \checkmark \rho'(\tilde{h}(\sigma x))$. But $\rho'(\tilde{h}(\sigma x)) = \bot$:

$$\begin{aligned}
\rho'(\tilde{h}(\sigma x)) &= \rho'(\widehat{\mathsf{id}} \circ \langle \mathsf{id}\,, h\rangle\,(\sigma x)) \\
&= \widehat{\mathsf{id}} \circ (\rho' \times \rho') \circ \langle \mathsf{id}\,, h\rangle \circ \sigma x \quad && (\rho' \circ \widehat{\mathsf{id}} = \widehat{\mathsf{id}} \circ (\rho' \times \rho')) \\
&= \widehat{\mathsf{id}} \circ \langle \rho' \circ \sigma\,, \rho' \circ h \circ \sigma\rangle\, x \\
&= \widehat{\mathsf{id}} \circ \langle \bot\,, \rho' \circ h \circ \sigma\rangle\, x && (\rho' \circ \sigma = \bot,\ \widehat{\mathsf{id}} \text{ left-strict}) \\
&= \bot.
\end{aligned}$$

This contradicts the triviality of s in R', *i.e.*, the existence of x' and x'' with $s \checkmark \sigma x'$ and $s \not\checkmark \sigma x''$. It follows by (D3b) that $\rho s_0 \in \mathcal{I}_{\mathsf{res}(R\bar{E})}$.

Since $\mathsf{level}\, h < \alpha$, induction hypothesis (B) applies, and thus, if we let g be

$$g = \rho_{\mathsf{arg}\bar{E}} \circ h \circ \sigma_{\bar{E}} : R\bar{E} \to R(\mathsf{arg}\,\bar{E})$$

then g is fully indexable, and $\mathsf{level}\, g = \mathsf{level}\, h < \alpha$. Therefore ρs is a valid α-index of $R\bar{E}$.

Note now that

$$\tilde{g} = \rho_{\mathsf{arg}\bar{E}} \circ \tilde{h} \circ \sigma_{\bar{E}}, \tag{7.4}$$

by the following:

$$\begin{aligned}
\tilde{g} &= \widehat{\mathsf{id}} \circ \langle \mathsf{id}\,, \rho \circ h \circ \sigma\rangle \\
&= \widehat{\mathsf{id}} \circ \langle \rho \circ \sigma\,, \rho \circ h \circ \sigma\rangle && (\rho \circ \sigma = \mathsf{id}) \\
&= \widehat{\mathsf{id}} \circ (\rho \times \rho) \circ \langle \mathsf{id}\,, h\rangle \circ \sigma \\
&= \rho \circ \widehat{\mathsf{id}} \circ \langle \mathsf{id}\,, h\rangle \circ \sigma && (\rho \circ \widehat{\mathsf{id}} = \widehat{\mathsf{id}} \circ (\rho \times \rho)) \\
&= \rho \circ \tilde{h} \circ \sigma.
\end{aligned}$$

It follows that

$$\rho s \checkmark x \qquad \text{iff} \qquad s \checkmark \sigma x,$$

by the following chain of equivalences, for all $x \in R\bar{E}$.

$$\begin{array}{lrcll} & \rho s = [g \Rightarrow^{\mathrm{a}} \rho s_0] & \checkmark & x & \\ \text{iff} & \rho s_0 & \checkmark & \tilde{g}x = (\rho \circ \tilde{h} \circ \sigma)x & \text{(Equation 7.4)} \\ \text{iff} & s_0 = \sigma(\rho s_0) & \checkmark & \tilde{h}(\sigma x) & \text{((D3b): } s_0 = \sigma(\rho s_0)\text{)} \\ \text{iff} & s = [h \Rightarrow^{\mathrm{a}} s_0] & \checkmark & \sigma x. & \end{array}$$

(B) Let $g = \rho_{\bar{F}} \circ h \circ \sigma_{\bar{E}}$. We need to show α-indexability of

$$\hat{g} : R\bar{E} \times R(\mathsf{arg}\bar{F}) \to R(\mathsf{res}\bar{F}).$$

Observe that

$$\hat{g} = \rho \circ \hat{h} \circ (\sigma \times \sigma),$$

by the following development:

$$\begin{array}{rcll} \hat{g} & = & \widehat{\mathsf{id}} \circ ((\rho \circ h \circ \sigma) \times \mathsf{id}) & \\ & = & \widehat{\mathsf{id}} \circ ((\rho \circ h \circ \sigma) \times (\rho \circ \sigma)) & (\rho \circ \sigma = \mathsf{id}) \\ & = & \rho \circ \widehat{\mathsf{id}} \circ ((h \circ \sigma) \times \sigma) & (\rho \circ \widehat{\mathsf{id}} = \widehat{\mathsf{id}} \circ (\rho \times \rho)) \\ & = & \rho \circ \widehat{\mathsf{id}} \circ (h \times \mathsf{id}) \circ (\sigma \times \sigma) & \\ & = & \rho \circ \hat{h} \circ (\sigma \times \sigma). & \end{array}$$

Choose an arbitrary index $r_0 \in \mathcal{I}(\hat{g}; z, Z)$ for $z \in R\bar{E} \times R(\mathsf{arg}\bar{F})$ and a non-empty and finite $Z \subseteq \uparrow^\star z$. But now $\sigma r_0 \in \mathcal{I}(\hat{h} \circ (\sigma \times \sigma); z, Z)$, because

$$\begin{array}{lrcl} & r_0 & \checkmark & \hat{g}z' = \rho \circ \hat{h} \circ (\sigma \times \sigma)z' \\ \text{iff} & \sigma r_0 \checkmark \hat{h}(\sigma \times \sigma)z'. & & \end{array}$$

By α-full indexability of h, there exists some α-index $s \in \mathcal{I}(\sigma \times \sigma; z, Z)$. But now note that this means that (A) may be used, with respect to the doubled decomposition $\Xi \times \Xi$ and the $2k$-tuple $(\bar{E}\,; \mathsf{arg}\bar{F})$, — s is trivial in the $R' \times R'$ component of $\Xi \times \Xi$, and we may project s to an α-index $\rho s \in \mathcal{I}(z, Z)$, which serves to prove g α-fully indexable. ■

7.3.4 Full Indexability of the Identity and Product Morphisms

The full indexability of the identity and various product morphisms will follow from the injective lemma and corollaries.

7.3.7 Proposition *For every k-ary decomposition Ξ, space E and k-tuple $\bar{E}$ of spaces, if $g : R\bar{E} \to E$ is α-fully indexable then so is*

$$g \circ \rho_{\bar{E}} : S\bar{E} \to E.$$

Proof: Let $h = g \circ \rho_{\bar{E}}$. We need to show α-indexability of

$$\hat{h} : S\bar{E} \times \mathsf{arg}E \to \mathsf{res}E.$$

Choose now an arbitrary index $r_0 \in \mathcal{I}(\hat{h}; z, Z)$ for some $z \in S\bar{E} \times \mathsf{arg}E$ and $Z \subseteq \uparrow^{\star} z$. It is easy to see that $\hat{h} = \hat{g} \circ (\rho_{\bar{E}} \times \mathsf{id}_{\mathsf{arg}E})$, so that $r_0 \in \mathcal{I}(\hat{g} \circ (\rho \times \mathsf{id}); z, Z)$, and, by α-full indexability of g, there exists an α-index $r \in \mathcal{I}(\rho \times \mathsf{id}; z, Z)$.

By the development for decompositions, $\Xi \times \mathsf{ID}$ is a $k+1$-ary decomposition, which may be seen here to be taken at the $k+1$-tuple $(\bar{E}\,; \mathsf{arg}E)$. (As per footnote on page 160, we ignore here a spurious component introduced by ID). Therefore, by the injective lemma, we may inject r to obtain an α-index $(\sigma_{\bar{E}} \times \mathsf{id}_{\mathsf{arg}E})r \in \mathcal{I}(z, Z)$, as required to show α-full indexability of h. ∎

7.3.8 Corollary *For all spaces E, E_1 and E_2 and $\ell \in \{1, 2\}$, if $f : E_\ell \to E$ is α-fully indexable then so is the composition $f \circ \pi_{\ell;E_1,E_2} : E_1 \times E_2 \to E$.*

Proof: Follows from previous proposition, applied to the decompositions PR and PR'. ∎

In order to establish that the projections themselves are fully indexable, we need to assume, at this point of the development, that the identity is fully indexable.

7.3.9 Corollary *For all spaces E_1 and E_2 and $\ell \in \{1, 2\}$, if the identity on E_ℓ is α-fully indexable then the projection $\pi_{\ell;E_1,E_2} : E_1 \times E_2 \to E_\ell$ is α-fully indexable.*

Proof: By the previous corollary, if E_ℓ is α-fully indexable then so is $\pi_\ell = \mathsf{id}_{E_\ell} \circ \pi_\ell$. ∎

As another corollary to the injective lemma, we are now able to discharge the assumption (P) from the proof of proposition 7.3.1.

7.3.10 Proposition *Assumption (P) holds for all spaces E, E_1 and E_2, that is, for every $\ell \in \{1, 2\}$, if $h : E \times E_\ell \to \mathsf{arg}E \times \mathsf{arg}E_\ell$ is an α-fully indexable function, then so is the function*

$$\begin{aligned} \Phi^{\ell}_{(E,E_1,E_2)} h &= (\mathsf{id}_{\mathsf{arg}E} \times \amalg_{\ell;\mathsf{arg}E_1,\mathsf{arg}E_2}) \circ h \circ (\mathsf{id}_E \times \pi_{\ell;E_1,E_2}) \\ &: E \times (E_1 \times E_2) \to \mathsf{arg}E \times (\mathsf{arg}E_1 \times \mathsf{arg}E_2). \end{aligned}$$

Proof: Follows directly from the injective lemma for the decompositions $\Xi_1 = \mathsf{ID} \times \mathsf{PR}$ for $\ell = 1$ and $\Xi_2 = \mathsf{ID} \times \mathsf{PR}'$ for $\ell = 2$ (modulo the proviso in footnote on page 160). ∎

7.3.11 Corollary *For all spaces E, E_1 and E_2, if $f_1 : E \to E_1$ and $f_2 : E \to E_2$ are α-fully indexable then so is their pairing $\langle f_1\,, f_2 \rangle$.*

Proof: Follows from previous proposition together with proposition 7.3.1. ∎

In conventional treatments of categorical product, once it has been established that the projections are morphisms in the category and that morphisms of a category are closed under pairing, one automatically obtains that the morphisms are also closed under product, thanks to the defining identity

$$f_1 \times f_2 = \langle f_1 \circ \pi_1 \,, f_2 \circ \pi_2 \rangle$$

for $f_1 : E_1 \to E_1'$ and $f_2 : E_2 \to E_2'$. This is true in our case only because the compositions here are of the restricted class already shown to preserve full indexability.

7.3.12 Corollary *For all spaces E_1, E_2, E_1' and E_2', if $f_1 : E_1 \to E_1'$ and $f_2 : E_2 \to E_2'$ are α-fully indexable then so is their product $f_1 \times f_2$.*

Proof: Follows from corollaries 7.3.8 and 7.3.11 and the defining identity. ∎

We turn now to the product injection. As for the projection, we need to assume full indexability of the identity.

7.3.13 Proposition *For spaces E, E_1 and E_2 and $\ell \in \{1, 2\}$, if $f : E \to E_\ell$ is α-fully indexable then so is $\amalg_{\ell;E_1,E_2} \circ f$. Moreover, if the identity on E_ℓ is α-fully indexable then so is the injection $\amalg_\ell : E_\ell \to E_1 \times E_2$.*

Proof: Without loss of generality look at $\ell = 1$. By definition, $\amalg_1 = \langle \mathsf{id}_{E_1} \,, \mathsf{K}\bot_{E_2} \rangle$, so that $\amalg_1 \circ f = \langle f \,, \mathsf{K}\bot_{E_2} \rangle$. The constant bottom function is 0-fully indexable (proposition 7.2.13), so that the composition $\amalg_1 \circ f$, being the pairing of two α-fully indexable functions, is also α-fully indexable. By the same reasoning, if id_{E_1} is α-fully indexable then so is $\amalg_1 = \langle \mathsf{id}_{E_1} \,, \mathsf{K}\bot_{E_2} \rangle$. ∎

7.3.14 Proposition *For every space E, the identity id_E is fully indexable, and $\mathsf{level}\,\mathsf{id}_E = 1 + \mathsf{ord}\,E$.*

Proof: We need to establish indexability of $\widehat{\mathsf{id}}_E : E \times \mathsf{arg}E \to \mathsf{res}E$. The proof proceeds by induction on $\mathsf{ord}\,E$.

Assume that we have shown the proposition for β and lower orders, and would like to prove it for E of order $\beta + 1$. Choose an arbitrary $r_0 \in \mathcal{I}(\widehat{\mathsf{id}}; z, Z)$ for $z \in E \times \mathsf{arg}E$ and a non-empty and finite $Z \subseteq \uparrow^\star z$. The desired index is provided by $r = [\amalg_{1;\mathsf{arg}E,\mathsf{arg}(\mathsf{arg}E)} \circ \pi_{2;E,\mathsf{arg}E} \Rightarrow^{\mathrm{a}} \amalg_1\, r_0]$, as is seen by:

$$\begin{array}{rrcl} & r & \checkmark & (x, y) \\ \text{iff} & \amalg_1\, r_0 & \checkmark & (\amalg_1 \circ \pi_2)(x, y) = (\iota_E xy, \iota_{\mathsf{arg}E}\, y \bot) \\ \text{iff} & r_0 & \checkmark & \pi_1\, (\amalg_1 \circ \pi_2)(x, y) = \iota_E xy = \widehat{\mathsf{id}}(x, y). \end{array}$$

$\amalg_1\, r_0$ is a valid index over the product of data domains, and it remains to establish that $\amalg_{1;\mathsf{arg}E,\mathsf{arg}(\mathsf{arg}E)} \circ \pi_{2;E,\mathsf{arg}E}$ is fully indexable. Since $\mathsf{ord}(\mathsf{arg}E) = \beta$, we may apply the induction hypothesis to $\mathsf{arg}E$, and assume that the identity on $\mathsf{arg}E$ is $\beta + 1$-fully indexable. Now, by the corollaries to the injective lemma, both $\amalg_{1;\mathsf{arg}E,\mathsf{arg}(\mathsf{arg}E)}$ and

$\pi_{2;E,\mathsf{arg}E}$, as well as their composition, are β-fully indexable, so that we have in fact defined r to be a valid $\beta+1$ index over E, and established that id_E is $\beta+1$ fully indexable.

For the base case, $\mathsf{ord}\,E = 0$, we repeat the same development, but cannot appeal to the induction hypothesis to establish full indexability of $\underline{\rotatebox{180}{\(\pi\)}}_1 \circ \pi_2$. Instead, consider directly $\underline{\rotatebox{180}{\(\pi\)}}_1 : \mathsf{arg}E \to \mathsf{arg}E \times \mathsf{arg}(\mathsf{arg}E)$. Because $\mathsf{arg}E$ is a one element domain, this is a constant function and, moreover, $\widehat{\underline{\rotatebox{180}{\(\pi\)}}_1}$ is a constant function that is 0-indexable. Therefore $\underline{\rotatebox{180}{\(\pi\)}}_{1;\mathsf{arg}E,\mathsf{arg}(\mathsf{arg}E)}$ is 0-fully indexable, and by corollaries above, so is the composition $\underline{\rotatebox{180}{\(\pi\)}}_1 \circ \pi_2$. Thus, r is a valid index at level 1, and id_E is 1-fully indexable. ∎

7.3.5 Applicative Indices over the Product

Before we complete our treatment of the product, we formalize the intuition that indices over the product, even if not necessarily induced by an index in one of the components, still have a preferential role for one of the components. This leads to the definition of an alternative notation for indices over the products, that is closer to the original notation suggested in the motivating discussion at the beginning of this chapter.

We define the ***main component*** $\mathsf{main}\,s$ of an applicative index $s = [h \Rightarrow^{\mathrm{a}} s_0]$ over a product $E_1 \times E_2$ to be $\mathsf{main}\,s_0$, *i.e.*, the unique $\ell \in \{1,2\}$ such that $\pi_\ell\, s_0 \in \mathcal{I}_{E_\ell}$.

Since the main component of $s = [h \Rightarrow^{\mathrm{a}} s_0]$ is solely determined by s_0, it turns out that only one component, $\pi_{\mathsf{main}\,s} \circ h$ of $h = \langle \pi_1 \circ h\,, \pi_2 \circ h\rangle$ really matters for determining the extent of s. The other component, $\pi_{(3-\mathsf{main}\,s)} \circ h$, has no import. This is formalized by the following proposition:

7.3.15 Proposition *The indices $s = [h \Rightarrow^{a} s_0]$ and $s' = [h' \Rightarrow^{a} s_0]$ over a product $E_1 \times E_2$ have the same extents if $\pi_{\mathsf{main}\,s_0} \circ h = \pi_{\mathsf{main}\,s_0} \circ h'$.*

Proof: Note that for $(x,y) \in E_1 \times E_2$,

$$\begin{array}{rcl} \tilde{h}(x,y) & = & \iota_{E_1 \times E_2}(x,y)(h(x,y)) \\ & = & (\iota_{E_1} x \times \iota_{E_2} y)(h(x,y)) \\ & = & (\iota_{E_1} x(\pi_1(h(x,y))), \iota_{E_2} y(\pi_2(h(x,y)))). \end{array}$$

Now, without loss of generality assume that $\mathsf{main}\,s_0 = 1$.

$$\begin{array}{lrcl} & s & \checkmark & (x,y) \\ \text{iff} & s_0 & \checkmark & \tilde{h}(x,y) = (\iota_{E_1} x(\pi_1(h(x,y))), \iota_{E_2} y(\pi_2(h(x,y)))) \\ \text{iff} & \pi_1\, s_0 & \checkmark & \pi_1(\tilde{h}(x,y)) = \iota_{E_1} x(\pi_1(h(x,y))) \\ \text{iff} & \pi_1\, s_0 & \checkmark & \pi_1(\tilde{h}'(x,y)) = \iota_{E_1} x(\pi_1(h'(x,y))) \\ \text{iff} & s_0 & \checkmark & \tilde{h}'(x,y) = (\iota_{E_1} x(\pi_1(h'(x,y))), \iota_{E_2} y(\pi_2(h'(x,y)))) \\ \text{iff} & s' & \checkmark & (x,y). \end{array}$$

∎

As a canonical choice, h in an index $[h \Rightarrow^{\mathrm{a}} s_0]$ over a product may always be replaced by $\underline{\rotatebox{180}{\(\pi\)}}_{\mathsf{main}\,s_0} \circ \pi_{\mathsf{main}\,s_0} \circ h$, which is the least function that shares the $\mathsf{main}\,s_0$ component $\pi_{\mathsf{main}\,s_0} \circ h$ with h.

Of course, we need to show that $\underline{\iota}_{\mathsf{main}\,s_0} \circ \pi_{\mathsf{main}\,s_0} \circ h$ is α-fully indexable for an α-fully indexable f. Post-composition with the injection has already been shown to preserve α-full indexability, as a corollary of the injective lemma. We now prove preservation of α-full indexability by post-composition with projection, as a corollary of the projective lemma. (This happens to be the only use we make of the projective lemma.)

7.3.16 Proposition *For every k-ary decomposition Ξ, space E and k-tuple $\bar{E}$ of spaces, if $h : E \to S\bar{E}$ is α-fully indexable then so is*

$$\rho \circ h : E \to R\bar{E}.$$

Proof: Let $g = \rho \circ h$. We need to show α-full indexability of

$$\hat{g} : E \times R(\mathsf{arg}\bar{E}) \to R(\mathsf{res}\bar{E}).$$

Choose an arbitrary $r_0 \in \mathcal{I}(\hat{g}; z, Z)$ for $z \in E \times R\bar{E}$ and a non-empty and finite $Z \subseteq \uparrow^{\star} z$. Note that

$$\begin{aligned} \hat{g} &= \widehat{\mathsf{id}} \circ ((\rho \circ h) \times \mathsf{id}) \\ &= \widehat{\mathsf{id}} \circ ((\rho \circ h) \times (\rho \circ \sigma)) \\ &= \rho \circ \widehat{\mathsf{id}} \circ (h \times \sigma) \\ &= \rho \circ \hat{h} \circ (\mathsf{id} \times \sigma). \end{aligned}$$

Thus, $r_0 \in \mathcal{I}(\rho \circ \hat{h} \circ (\mathsf{id} \times \sigma); z, Z)$, so that $\sigma r_0 \in \mathcal{I}(\hat{h} \circ (\mathsf{id} \times \sigma); z, Z)$. By full indexability of h, $\hat{h}$ has an α-index $s \in \mathcal{I}(\mathsf{id} \times \sigma; z, Z)$. Obviously, we would like to project s so as to obtain the desired index in $E \times R(\mathsf{arg}\bar{E})$; may we do so? The fact that s separates injected points guarantees that this is permissible — the projective lemma applies, with respect to the $k+1$ decomposition $\mathsf{ID} \times \mathsf{PR}$, and $k+1$-tuple $E; \mathsf{arg}\bar{E}$. Therefore we have an α-index $(\mathsf{id} \times \rho)s \in \mathcal{I}(z, Z)$, and we conclude that g is α-fully indexable. ∎

7.3.17 Corollary *For all spaces E, E_1 and E_2 and $\ell \in \{1, 2\}$, if $f : E \to E_1 \times E_2$ is α-fully indexable then so is the composition $\pi_\ell \circ f : E \to E_\ell$.*

Proof: Follows from previous proposition, applied to the decompositions PR and PR'. ∎

Now define the following alternate notation — call it the ***explicit notation*** — for indices over the product of two spaces E_1 and E_2, that designates explicitly a main component. For any $\ell \in \{1, 2\}$, fully indexable function $h' : E_1 \times E_2 \to \mathsf{arg}E_\ell$, and r_0 index of $\mathsf{res}E_\ell$ we define $[h' \Rightarrow^{\mathrm{a}}_{\ell} r_0]$ to stand for the index $[\underline{\iota}_\ell \circ h' \Rightarrow^{\mathrm{a}} \underline{\iota}_\ell\, r_0]$ of $E_1 \times E_2$.

Clearly, $\mathsf{main}[h' \Rightarrow^{\mathrm{a}}_{\ell} r_0] = \ell$. Moreover, the level and extent of $[h' \Rightarrow^{\mathrm{a}}_{\ell} r_0]$ may be directly inferred from this presentation: Since $\mathsf{level}\, h' = \mathsf{level}(\underline{\iota}_\ell \circ h')$, $[h' \Rightarrow^{\mathrm{a}}_{\ell} r_0]$ is at level $\mathsf{level}\, h' + 1$, in analogy with the usual notation for applicative indices. Also, for any $z \in E_1 \times E_2$,

$$[h' \Rightarrow^{\mathrm{a}}_{\ell} r_0] \checkmark_{E_1 \times E_2} z \qquad \text{iff} \qquad r_0 \checkmark \iota_{E_\ell}(\pi_\ell\, z)(h' z) = (\widehat{\mathsf{id}}_{E_\ell} \circ \langle \pi_\ell\,, h' \rangle) z,$$

by the distribution of projection over $\widehat{\mathsf{id}}$ and properties of indices over data spaces:

$$\begin{array}{lrcl} & [h' \Rightarrow^{\mathrm{a}}_{\ell} r_0] & \checkmark & z \\ \text{iff} & [\iota_\ell \circ h' \Rightarrow^{\mathrm{a}} \iota_\ell\, r_0] & \checkmark & z \\ \text{iff} & \iota_\ell\, r_0 & \checkmark & \widetilde{(\iota_\ell \circ h')} z \\ \text{iff} & r_0 & \checkmark & \pi_\ell(\widehat{\mathsf{id}}_{E_1 \times E_2}(z, \iota_\ell(h'z))) \\ \text{iff} & r_0 & \checkmark & \widehat{\mathsf{id}}_{E_\ell}(\pi_\ell\, z, h'z). \end{array}$$

As an example, the index $[\iota_1 \circ \pi_2 \Rightarrow^{\mathrm{a}} \iota_1\, r_0]$ employed in the proof of 7.3.14 may be written, using explicit notation, as $[\pi_2 \Rightarrow^{\mathrm{a}}_1 r_0]$.

Can every index over $E_1 \times E_2$ be written down in explicit notation? That is, does there exist an index $[h' \Rightarrow^{\mathrm{a}}_{\ell} r_0]$ corresponding to any given index $[h \Rightarrow^{\mathrm{a}} s_0]$ of $E_1 \times E_2$? The answer is yes and no. On the one hand, as we have shown above, only the main component $\pi_{\mathsf{main}\, s_0} \circ h$ of h matters for determining the extent of the index; this leaves a lot of latitude for varying the other component $\pi_{(3-\mathsf{main}\, s_0)} \circ h$, which cannot be reflected in the choice of an h' such that $\iota_{\mathsf{main}\, s_0} \circ h' = h$. On the other hand, if instead of $s = [h \Rightarrow^{\mathrm{a}} s_0]$ we consider $s' = [\iota_{\mathsf{main}\, s_0} \circ \pi_{\mathsf{main}\, s_0} \circ h \Rightarrow^{\mathrm{a}} s_0]$, an index with the same extent, as described above, then s' may be given in explicit notation as $[\pi_{\mathsf{main}\, s_0} \circ h \Rightarrow^{\mathrm{a}}_{\mathsf{main}\, s_0} \pi_{\mathsf{main}\, s_0}\, s_0]$. Thus, not every index may be described by explicit notation, but for every index there exists an index with the same extent that does have an explicit description. This is sufficient to allow us to use explicit notation when reasoning about indices over products.

7.3.6 Additional Corollaries

We are able to draw at this point some corollaries from the development related to the product. First, not only are decomposition projections and injections fully indexable, but they are also indexable. This lets us complete a proof of full indexability of constant functions, and show that uncurrying preserves full indexability, and application is fully indexable.

7.3.18 Proposition *For every k-ary decomposition $\Xi = (R, R', S)$ and k-tuple $\bar{E}$ of spaces, the projection $\rho_{\bar{E}}$ and injection $\sigma_{\bar{E}}$ are indexable.*

Proof: If $r \in \mathcal{I}_{R\bar{E}}(\rho_{\bar{E}}; z, Z)$ then by the injective lemma $\sigma_{\bar{E}} r \in \mathcal{I}_{S\bar{E}}(z, Z)$.

If $s \in \mathcal{I}_{S\bar{E}}(\sigma_{\bar{E}}; x, X)$ then s is trivial in the R' component, and by the projective lemma $\rho_{\bar{E}} s \in \mathcal{I}_{R\bar{E}}(x, X)$. ∎

7.3.19 Proposition *A constant function $\mathsf{K}y : E \to E'$ is fully indexable, for every $y \in E'$.*

Proof: Recall (see proposition 7.2.12) that, for $y \in E'$, $\widehat{\mathsf{K}y} = (\iota_{E'} y) \circ \pi_2$. ιy is indexable by definition, and by the previous proposition π_2 is indexable by virtue of being a projection for the product decomposition. ∎

7.3.20 Proposition *Full indexability is preserved by uncurrying, and the application operation is fully indexable.*

Proof: Let $f : E_1 \to (E_2 \to_{\mathrm{fx}} E')$ be a fully indexable function, and let $g = \mathsf{uncurry}\, f$.

Observe that for $(x, (y, z)) \in E_1 \times (E_2 \times \mathsf{arg} E')$,

$$\begin{array}{rcl} \hat{f}(x,(y,z)) & = & \iota_{E_2 \to_{\mathrm{fx}} E'}(fx)(y,z) \\ & = & \widehat{fx}(y,z) \\ & = & \iota_{E'}(fxy)z \\ & = & \iota_{E'}(g(x,y))z \\ & = & \hat{g}((x,y),z). \end{array}$$

That is, $\hat{g} = \hat{f} \circ \mathsf{assl}$. But the associative isomorphisms are indexable, since are the projections and injections in the associative decomposition; hence the composition $\hat{f} \circ \mathsf{assl}$ is indexable, and $g = \mathsf{uncurry}\, f$ is fully indexable.

The application $\mathsf{app} : (E \to_{\mathrm{fx}} E') \times E \to E'$ is fully indexable, being the uncurrying of the identity on $E \to_{\mathrm{fx}} E'$ which has already been shown fully indexable. ∎

7.4 Compositionality and Indexability

$$\begin{tikzcd}
E_1 \times \mathsf{arg} E_3 \arrow[r, "f_1 \times \mathsf{id}"] \arrow[rrrd, "\widehat{f_2 \circ f_1}"'] & E_2 \times \mathsf{arg} E_3 \arrow[r, "f_2 \times \mathsf{id}"] \arrow[rrd, "\hat{f}_2"'] & E_3 \times \mathsf{arg} E_3 \arrow[r, "\iota_{E_3} \times \mathsf{id}"] \arrow[rd, "\widehat{\mathsf{id}}_{E_3}"'] & \mathsf{rep}\, E_3 \times \mathsf{arg} E_3 \arrow[d, "\mathsf{app}"] \\
 & & & \mathsf{res} E_3
\end{tikzcd}$$

Figure 7.7: Composition of fully indexable functions

There are two open questions that we consider in this section, but are not able to settle yet.

Firstly, are fully indexable functions closed under composition? As opposed to indexable functions, this is by no means obvious.

Secondly, the "full" in full indexability refers to a requirement that functions be indexable in their (generalized) fully uncurried forms. This is intended as a restriction above and beyond indexability of the functions themselves. But is it in fact the case that all fully indexable functions are indexable? (Indexability here is meant to be taken with respect to the applicative index structures.)

The next two propositions show that these questions are closely related.

7.4.1 Proposition *If fully indexable functions are indexable then fully indexable functions are closed under composition.*

Proof: To show that fully indexable functions are closed under composition, one would need to show that for fully indexable functions $f_1 : E_1 \to E_2$ and $f_2 : E_2 \to E_3$, the function

$$\widehat{f_2 \circ f_1} = \hat{f}_2 \circ (f_1 \times \mathsf{id}_{\mathsf{arg} E_3}) : E_1 \times \mathsf{arg} E_3 \to \mathsf{res} E_3$$

is indexable, see figure 7.7. By full indexability of f_2, $\hat{f}_2$ is indexable. By the product development, $f_1 \times \mathsf{id}$ is already known to be a fully indexable function. Thus, if we assume that fully indexable functions are indexable then $f_1 \times \mathsf{id}$ would be indexable, and so would its composition with $\hat{f}_2$, leading to the full indexability of $f_2 \circ f_1$. ∎

7.4.2 Proposition *If fully indexable functions are closed under composition and $f_2 \circ f_1$ is at a finite level in the fully indexable hierarchy whenever f_2 is, then for every $f : E \to E'$, f is indexable, and $\langle \mathsf{id}_E\,, f\rangle$ is indexable.*

Proof: The proof proceeds by induction on the order of E'.

Start by showing indexability of f. For the base case, if $\mathsf{ord}\, E' = 0$ then full indexability of f coincides with indexability. For E' at higher order, let $x \in E$, and $X \subseteq {\uparrow^\star} x$ non-empty and finite. If $r' \in \mathcal{I}_{E'}(f; x, X)$ then $r' = [h' \Rightarrow^{\mathsf{a}} r'_0]$, with $h' : E' \to \mathsf{arg} E'$ fully indexable and $r'_0 \in \mathcal{I}_{\mathsf{res} E'}(\tilde{h'} \circ f; x, X)$. Note that

$$\tilde{h'} \circ f = \hat{f} \circ \langle \mathsf{id}_E\,, h' \circ f\rangle\,.$$

By assumption regarding composition of fully indexable functions, $h' \circ f$ is fully indexable. Since

$$h' \circ f : E \to \mathsf{arg} E'$$

and $\mathsf{ord}(\mathsf{arg} E') < \mathsf{ord}\, E'$ we may appeal to the induction hypothesis to obtain that $\langle \mathsf{id}_E\,, h' \circ f\rangle$ is indexable. By full indexability of f, $\hat{f}$ is indexable as well, so that the composition

$$\hat{f} \circ \langle \mathsf{id}_E\,, h' \circ f\rangle : E \to \mathsf{res} E'$$

is indexable; it thus has an index $r \in \mathcal{I}_E(x, X)$, as desired to prove indexability of f.

Turn now to indexability of $\langle \mathsf{id}\,, f\rangle$. If $r' \in \mathcal{I}_{E \times E'}(\langle \mathsf{id}\,, f\rangle\,; x, X)$ then $\mathsf{main}\, r'$ is either 1 or 2.

If $\mathsf{main}\, r' = 1$ then $r' = [h' \Rightarrow^{\mathsf{a}}_1 r'_0]$ for $h' : E \times E' \to \mathsf{arg} E$ fully indexable and $r'_0 \in \mathcal{I}_{\mathsf{res} E}(\hat{\imath}_E \circ \langle \mathsf{id}_E\,, h' \circ \langle \mathsf{id}_E\,, f\rangle\rangle\,; x, X)$.

By product development, $\langle \mathsf{id}_E\,, f\rangle$ is fully indexable. By assumption about compositionality of fully indexable functions, $h' \circ \langle \mathsf{id}_E\,, f\rangle$ is fully indexable, and is at a finite level in the fully indexable hierarchy. But now we may construct an index $r = [h' \circ \langle \mathsf{id}_E\,, f\rangle \Rightarrow^{\mathsf{a}} r'_0]$, so that $r \in \mathcal{I}_E(x, X)$, as desired.

If, on the other hand, $\mathsf{main}\, r' = 2$, then $r' = [h' \Rightarrow^{\mathsf{a}}_2 r'_0]$ for $h' : E \times E' \to \mathsf{arg} E'$ fully indexable and $r'_0 \in \mathcal{I}_{\mathsf{res} E'}(\hat{\imath}_{E'} \circ \langle f\,, h' \circ \langle \mathsf{id}\,, f\rangle\rangle\,; x, X)$. Note that

$$\hat{\imath}_{E'} \circ \langle f\,, h' \circ \langle \mathsf{id}\,, f\rangle\rangle \quad = \quad \hat{f} \circ \langle \mathsf{id}_E\,, h' \circ \langle \mathsf{id}_E\,, f\rangle\rangle\,.$$

Again, by product development, $\langle \mathsf{id}_E\,, f\rangle$ is fully indexable. By assumption that fully indexable functions are closed under composition $h' \circ \langle \mathsf{id}_E\,, f\rangle$ is fully indexable. If $\mathsf{ord}\, E' > 0$ then, since this composition has a lower-order target —

$$h' \circ \langle \mathsf{id}_E\,, f\rangle : E \to \mathsf{arg} E',$$

and $\mathsf{ord}(\mathsf{arg}E') < \mathsf{ord}\,E'$ — the induction hypothesis applies, and $\langle \mathsf{id}_E\,, h' \circ \langle \mathsf{id}_E\,, f\rangle\rangle$ is indexable. If $\mathsf{ord}\,E' = 0$ then

$$\langle \mathsf{id}_E\,, h' \circ \langle \mathsf{id}_E\,, f\rangle\rangle = \langle \mathsf{id}_E\,, \mathsf{K}\bot_{\mathsf{arg}E'}\rangle = \perp\!\!\!\!\bot_1 \circ \mathsf{id}_E$$

is indexable, by the product development.

In either case, $\langle \mathsf{id}_E\,, h' \circ \langle \mathsf{id}_E\,, f\rangle\rangle$ is indexable, and by full indexability of f, $\hat{f}$ is indexable as well, so that the composition $\hat{f} \circ \langle \mathsf{id}_E\,, h' \circ \langle \mathsf{id}_E\,, f\rangle\rangle$ is indexable, and has an index $r \in \mathcal{I}_E(x, X)$, as desired. ∎

At this point we do not have a conclusive proof or disproof of either of these two related questions, and we are forced to leave this as an open conjecture:

7.4.3 Conjecture *Fully indexable functions are indexable, and they are closed under composition.*

If this conjecture is true then, in addition to the development thus far, we have exhibited a CCC of spaces and fully indexable functions, which serves as a model of PCF, via the standard interpretation of PCF in a CCC. This model will still fail to be fully abstract, due to the problem of *partitioning*, as discussed in section 7.5.1; yet it is a smaller model than the model of bidomains and stable functions, so that it is closer to the goal of a fully abstract model.

If this conjecture fails to hold then spaces and fully indexable functions do not even form a category. Nevertheless, within the category PST of spaces and stable functions we have designated a subclass of morphisms — the fully indexable functions — for which we can provide a notion of a product and exponentiation; that is to say, using space product and the fully indexable function space, the product and exponentiation diagrams commute for spaces and fully indexable functions, since the restricted compositions employed in these diagrams have been shown to be fully indexable. At this point, this not-quite-subcategory has not been shown to provide a model for PCF, since the interpretation of an application term

$$[\![\Gamma \vdash^{\mathrm{ca}} MM' : \sigma']\!] = \mathsf{app} \circ \langle [\![\Gamma \vdash^{\mathrm{ca}} M : \sigma \to \sigma']\!]\,, [\![\Gamma \vdash^{\mathrm{ca}} M' : \sigma]\!]\rangle$$

is constructed using a composition that has not been shown to preserve full indexability.

7.4.1 Pairing and Indexability

An alternative approach to resolving the conjecture at hand could have been to work with morphisms that are required to be both fully indexable and indexable. In this case, an unresolved problem occurs when trying to prove that the pairing $\langle f_1\,, f_2\rangle$ of such morphisms is indexable. The balance of this section devotes some attention to showing indexability of the pairing or product of fully indexable functions.

First, recall that not all indexable functions are fully indexable. Even at first order, the curried forms of parallel-or and Gustave's function are indexable, and we were in fact driven to full indexability in order to eliminate functions such as these. We can now make the following observations:

7.4.4 Proposition *If $f \times \mathsf{id}_{\mathsf{arg}E'}$ is indexable for $f : E \to E'$ then f is fully indexable. Indexable functions are not closed under product or pairing.*

Proof: If $f \times \mathsf{id}_{\mathsf{arg}E'}$ is indexable then $\hat{f} = \widehat{\mathsf{id}}_{E'} \circ (f \times \mathsf{id}_{\mathsf{arg}E'})$ is indexable by full indexability of $\mathsf{id}_{E'}$ and compositionality of indexable functions.

If $f : E \to E'$ is indexable and indexable functions are closed under product then $f \times \mathsf{id}_{\mathsf{arg}E'}$ is indexable, from which it follows by the above that f is fully indexable. But this contradicts the existence of functions that are indexable yet not fully indexable. Indexable functions are therefore not closed under product, and since product $f_1 \times f_2 = \langle f_1 \circ \pi_1 , f_2 \circ \pi_2 \rangle$ is defined in terms of pairing, indexable functions are not closed under pairing. ∎

The above proposition essentially confirms that the product of spaces is tailored for fully indexable functions, and not indexable ones. But if our desired result — that fully indexable functions are indexable — is to hold, then pairing and product had better preserve indexability when restricted to fully indexable functions.

The difficulty in showing indexability of a pairing $\langle f_1 , f_2 \rangle : E \to E_1 \times E_2$ lies in decomposing an index over the product in order to make use of properties of the constituent morphisms; this is trivially possible under the simple product assumption, but cannot in general be done for the applicative index structure. Nevertheless, we are able to give a positive result in this direction for an important test case: the diagonal map

$$\Delta_E = \langle \mathsf{id}_E , \mathsf{id}_E \rangle .$$

Indexability of the diagonal would imply that every applicative index over $E \times E$ may be reduced to an applicative index over E, which is by no means a foregone conclusion.

The proof of indexability of the diagonal map proceeds by induction, and necessitates a new notion of *level preservation*, that lets us prove indexability incrementally: f is ***level preserving up to*** α if it is stable and for every $\alpha' \leq \alpha$, f has an input index at level α' for every output index at level α':

$$\forall \alpha' \leq \alpha \;.\; \forall \text{ non-empty finite } X \subseteq {\uparrow^\star}\, x \;.\; (\exists r' \in \mathcal{I}^{\alpha'}(f; x, X)) \implies (\exists r \in \mathcal{I}^{\alpha'}(x, X)).$$

f is ***level preserving*** if it is level preserving up to α for every α. If f is level preserving then it is certainly indexable. (But f may be level preserving up to some α_0, and yet fail to be indexable, if it has no index for some index in at level $\alpha_0 + 1$ or higher over the output.)

7.4.5 Proposition *For all α*

(1) For all spaces E and E_0, $\Delta_E \times \mathsf{id}_{E_0} : E \times E_0 \to (E \times E) \times E_0$ is level preserving up to α.

(2) For all spaces E, E_0 and E_1 and $h : (E \times E) \times E_0 \to E_1$ α-fully indexable, the function $h \circ (\Delta_E \times \mathsf{id}_{E_0}) : E \times E_0 \to E_1$ is α-fully indexable.

Proof: By induction on α. For $\alpha = 0$, (1) is true, since every function is vacuously level preserving up to 0, and (2) is true since a 0-fully indexable h is constant (proposition 7.2.13) and remains constant and 0-fully indexable under post-composition. Consider now the case for $\alpha > 0$.

(1) For $\alpha' < \alpha$ we can appeal to the induction hypothesis, so we may consider only α indices.

Let $[h \Rightarrow^{\mathsf{a}} s_0] \in \mathcal{I}^{\alpha}(\Delta \times \mathsf{id}; z, Z)$ for appropriate z and Z. That is, h is $(\alpha - 1)$-fully indexable, and $s_0 \in \mathcal{I}(\tilde{h} \circ (\Delta \times \mathsf{id}); z, Z)$. Being an index over data domains, s_0 is induced by an index in one of the components of the product $(\mathsf{res}E \times \mathsf{res}E) \times \mathsf{res}E_0$.

If $s_0 = \iota_2\, r_0$ for $r_0 \in \mathcal{I}(\pi_2 \circ \tilde{h} \circ (\Delta \times \mathsf{id}); z, Z)$ then $[\pi_2 \circ h \circ (\Delta \times \mathsf{id}) \Rightarrow^{\mathsf{a}}_2 r_0] \in \mathcal{I}^{\alpha}(z, Z)$; to see this, note that

$$\begin{aligned}
\pi_2 \circ \tilde{h} \circ (\Delta \times \mathsf{id}) &= \pi_2 \circ \widehat{\mathsf{id}} \circ \langle \mathsf{id}\,, h\rangle \circ (\Delta \times \mathsf{id}) \\
&= \widehat{\mathsf{id}} \circ (\pi_2 \times \pi_2) \circ \langle \mathsf{id}\,, h\rangle \circ (\Delta \times \mathsf{id}) \\
&= \widehat{\mathsf{id}} \circ \langle \pi_2 \circ (\Delta \times \mathsf{id})\,, \pi_2 \circ h \circ (\Delta \times \mathsf{id})\rangle \\
&= \widehat{\mathsf{id}} \circ \langle \pi_2\,, \pi_2 \circ h \circ (\Delta \times \mathsf{id})\rangle\,.
\end{aligned}$$

This defines a valid index at level α, since $\pi_2 \circ h$ is $(\alpha - 1)$-fully indexable by product development, and, by induction hypothesis (2), $\pi_2 \circ h \circ (\Delta \times \mathsf{id})$ is $(\alpha - 1)$-indexable.

On the other hand, if $s_0 = \iota_1(\iota_\ell\, r_0)$ for $\ell \in \{1, 2\}$ and $r_0 \in \mathcal{I}(\pi_\ell \circ \pi_1 \circ \tilde{h} \circ (\Delta \times \mathsf{id}); z, Z)$ then, by similar reasoning, $[\pi_\ell \circ \pi_1 \circ h \circ (\Delta \times \mathsf{id}) \Rightarrow^{\mathsf{a}}_1 r_0] \in \mathcal{I}^{\alpha}(z, Z)$; again, this is a valid index at level α by product development and by induction hypothesis (2).

(2) Note that

$$(\hat{\ })(h \circ (\Delta \times \mathsf{id}_{E_0})) = \hat{h} \circ ((\Delta \times \mathsf{id}_{E_0}) \times \mathsf{id}_{\mathsf{arg}E_1}) = \hat{h} \circ (\Delta \times \mathsf{id}_{E_0 \times \mathsf{arg}E_1}) \circ \mathsf{assl}\,.$$

$\hat{h}$ is α-indexable by assumption, so that, by (1), its composition with $\Delta \times \mathsf{id}_{E_0 \times \mathsf{arg}E_1}$ is α-indexable, and by the product development a further composition with the associative isomorphism preserves α-indexability. ∎

7.4.6 Corollary *Δ_E is indexable and level preserving for every space E.*

Proof: If $[h \Rightarrow^{\mathsf{a}}_\ell r_0] \in \mathcal{I}^{\alpha}(\Delta; z, Z)$ then $[h \circ \Delta \Rightarrow^{\mathsf{a}} r_0] \in \mathcal{I}^{\alpha}(z, Z)$. ∎

7.5 Towards Definability

Indices are meant to convey sufficient information about computation strategies to be able to give a defining term for an indexable function (in its fully uncurried form). The feasibility of this approach is exemplified by the proof of definability of isolated elements in the ca-sequential model, proposition 5.3.3. Can this proof be now adapted to the fully indexable model?

As we have indicated, even if the fully indexable functions are closed under composition, so that spaces and fully indexable functions form a CCC where PCF may be given a model, this model will fail to be fully abstract. The reason is that our development in this chapter has not taken into account the phenomenon of *partitioning*. This section presents the phenomenon, and sketches a possible way to refine the concept of indexability so as to take it into account. We also discuss here briefly additional considerations related to definability.

7.5.1 Partitioning

By the term *partitioning* we refer to a problematic behavior of functions that may not occur for PCF definable functions, and must therefore be ruled out in order to obtain a fully abstract model for PCF. The problem is raised by the following observation by Curien: the functional $\phi_1 = [\mathsf{lor} \Rightarrow \mathsf{T}] \vee^{\mathrm{p}} [\mathsf{ror} \Rightarrow \mathsf{T}]$ is PCF-definable, using imbrication, by the term

$$\lambda\texttt{f}\ .\ \texttt{if}\ (\texttt{f}\,\mathsf{T}\,\Omega)\&(\texttt{f}\,\Omega\,\mathsf{T})\&\neg(\texttt{f}\,\mathsf{F}\,\mathsf{F})\ \texttt{then}\ \mathsf{T}\ \texttt{else}\ \Omega,$$

but the functional $\phi_2 = [\mathsf{lor} \Rightarrow \mathsf{T}] \vee^{\mathrm{p}} [\mathsf{ror} \Rightarrow \mathsf{F}]$ is not PCF definable [Cur93].

However, both ϕ_1 and ϕ_2 are indexable with respect to the applicative index structure on $\mathbf{Bool} \times \mathbf{Bool} \to \mathbf{Bool}$ (taking primal data indices over $\mathbf{Bool}$), because the index

$$r_0 = [\lambda f\ .\ f(f(\mathsf{T}, \bot), f(\bot, \mathsf{T})) \Rightarrow^{\mathrm{a}} \{\mathsf{T}\}]$$

will separate lor and ror from their meet, sor.

This shortcoming is related to the fact that the indexable functions have a generalized topological characterization. An immediate corollary of the observation is that PCF-definable functions may not be given a pure (generalized) topological characterization as the functions continuous with respect to some collection of open sets in model where ϕ_2 exists (ϕ_1 must be in the model, since it is definable). To see this, assume that $\{\mathsf{T}, \mathsf{F}\}$ is an open set, which should be a reasonable assumption to make; then the set $\{\mathsf{lor}, \mathsf{ror}\}$ must be open, since it is $\phi_1^{-1}\{\mathsf{T}, \mathsf{F}\}$, and it must at the same time not be open, since it equals $\phi_2^{-1}\{\mathsf{T}, \mathsf{F}\}$, and is the only available "witness" in the purported (generalized) topology that shows that ϕ_2 is not PCF-definable.

The framework of indexed computation provides concepts and terminology necessary to describe and address this phenomenon. The key observation to make is that the index r_0 used above has a single answer — that is, it represents a query with only one non-diverging answer (roughly: are you above lor or ror?) — but, whereas ϕ_1 uses it to answer a query with a single possible answer, ϕ_2 uses r_0 to answer a query with *two* answers, $\{\mathsf{T}\}$ and $\{\mathsf{F}\}$. ϕ_2 is essentially making a decision between different answers that is not based on information it has about the input; this is the phenomenon that we refer to as partitioning of answers, or just ***partitioning***. This is obviously not a valid behavior on part of reasonable deterministic computational agents, and we must rule it out.

Partitioning was not a problem in chapters 4 and 5: No monotone function may partition points of a lobe into different lobes of the target index, and it is easy to check that data indices have single lobes as their answers, and the constructions of the simple product indices and the constant-applicative indices preserve this property. For applicative indices this identification of answers with single lobes ceases to hold, however, and r_0 is, appropriately, an example of an answer index whose extent consists of two separate lobes $\{\mathsf{lor}\}$ and $\{\mathsf{ror}\}$. Note that lor and ror are not consistent in the stably- or pointwise-ordered indexable function space (even though the phantom of their pointwise join, parallel-or, still appears to haunt them).

Proper Indexability

One approach to remedy the problem posed by partitioning is to strengthen the basic definition of indexability to rule out indexable functions that partition answers. We tentatively define a function $f : B \to B'$ between indexed dcpos to be ***properly indexable*** iff it is continuous and for every $x \in B$, non-empty finite $X \subseteq \uparrow x$ and index $r' \in \mathcal{I}'(f; x, X)$ there exists an index $r \in \mathcal{I}(x, X)$ and, moreover, for every answer $s \in \mathbf{Ans}\, r$ there exists an answer $s' \in \mathbf{Ans}\, r'$ such that

$$f(X \cap (s \checkmark\ )) \subseteq (s' \checkmark\ ).$$

In other words, every answer over the input may be mapped to a *single* answer over the output; we may speak about an *answer mapping*. There is some arbitrariness in the choice of s' for s with $X \cap (s \checkmark\ )$ empty, but this does not pose a serious problem, since $\mathbf{Ans}\, r'$ is not empty. Looking back at the example, it is easy to see that ϕ_1 is properly indexable, while ϕ_2 is not.

Relating this to the envisioned generalization of the proof of definability, proposition 5.3.3, we see that the answer mapping is necessary to establish a correspondence between the answers of the index over the output and answers of the index over the input that is used by the function to be defined; the existence of such a mapping could be tacitly relied on in the proof for the ca-sequential case 5.3.3, but must be required explicitly now.

Since ϕ_1 is properly indexable and ϕ_2 is not, proper indexability may not in general be given a generalized topological characterization as we have given for indexable functions. Instead, we conjecture that there is a further generalization of topological characterizations of functions, and of critical sets, that incorporates a consideration of partition, and with respect to which the properly indexable functions, and perhaps the PCF-definable functions, may be said to be continuous[6].

Improved Substrates

An alternative approach to the resolution of the problem posed by partitioning is to improve the underlying substrate over which we formulate indexed computation. In fact, the existence of the functional ϕ_2 in the bidomain model is tantamount to stating that it is *not* the common restriction of the continuous function model for PCF and the stable model (see *e.g.* [JM91]).

It therefore seems advisable to improve the substrate employed; one could, for instance, enrich the structure of bidomains by an additional predicate of *consistency*, which would be taken pointwise over function spaces. This consistency relation may then be used to single out the functions that respect it. For instance, ϕ_2 would be excluded from such an improved model, because the left- and right-strict are pointwise consistent, and ϕ_2 maps them to inconsistent values (This, in effect, resurrects the phantom of parallel-or).

[6] A possible idea here, briefly, is to endow open sets with a *required partition* — *i.e.*, designate the parts of the set that must be partitionable into different answers. Such a partitioned set may be represented as a family of subsets whose union is the original set. Rather than working with a collection of *open sets* $O \subseteq \mathcal{P}\, A$ one would work with a collection of *open families* $O \subseteq \mathcal{P}(\mathcal{P}\, A)$. Thus, we obtain a new concept of separation by an index: $\{\{\mathsf{lor}, \mathsf{ror}\}\}$ has an index (r_0) that separates it from sor, but $\{\{\mathsf{lor}\}, \{\mathsf{ror}\}\}$ does not, because there is no index that does this while assigning lor and ror to different answers.

Such an approach is taken by Winskel in [Win94], where an extensional consistency predicate is employed. And closely related is the work by Bucciarelli and Ehrhard [BE91a] that constructs the common refinement of the continuous functions model and stable model by means of an extensional embedding of the latter in the former.

This approach to the resolution of the problem of partitioning — excluding the problematic functions, even before imposing indexability — seems preferable to the imposition of proper indexability as proposed above. But of course, its effect would be to restore the equivalence between indexable and properly indexable functions that holds for data indexability and ca-indexability, but failed to hold for applicative indexability over bidomains.

Chapter 8

Conclusion

This dissertation has presented progress towards an abstract characterization of (higher-order) sequential computation. We briefly survey some of the main points, outline some of the connections to related work, and propose some directions for continued work.

We have formulated a general framework of indexed computation, intended to formalize incremental demand-driven computation with restricted sets of primitives. We claim that an appropriate choice of the available demands should make it possible to capture various computational disciplines. The framework arises as a natural generalization of Kahn and Plotkin's formalization of first-order sequential computation [KP78], and it has been motivated by the desire to characterize sequential computation, but it should prove helpful in an attempt to characterize other compositionally-closed classes of functions, such as classes of continuous or stable functions that are dominated by sequential functions (sometimes called subsequential functions).

More generally, indexed computation may prove useful in characterizing classes of functions of differing degrees of sequentiality and parallelism. It is known that there is a rich hierarchy of such classes. Relevant work is presented in [Tra85, Saz76a, Tra75, Buc95]. To attack this goal using the indexed computation approach, it would seem necessary to formulate a general concept of computation strategy — that would encompass parallel computation — and then single out within it the computation strategies that correspond to various classes of functions, such as the sequential, stable, subsequential functions. First attempts to formulate a parallel computation strategy are reported in [BG92].

Of course, every choice of a (restricted notion of) computation strategy — *e.g.*, sequential, stable, or subsequential — will give rise to a different model, and in order to relate these models it may be necessary to talk not about the models in isolation, but about the operation mapping a choice of a computation strategy to a model.

We have not yet examined properly the connections between this work and recent game-theoretic approaches [AJM94, HO94] that have led to the construction of fully abstract models. The two approaches are quite different in flavor, but on the face of it, there should be close connections, since both approaches may be formulated in terms of strategies and permissible moves. An apparently significant difference is that the moves of the game-theoretic approach are atomic, and the constructions of complex domains impose structural constraints on the ordering or interleaving of the atomic moves, whereas our approach employs structured moves (*i.e.*, indices) over compound domains, with uniform structural constraints (definition of indexability). An explanation or bridging of this gap may prove fruitful for both approaches.

We have shown in section 3.6.3 a connection between our work and Bucciarelli and Ehrhard's strongly stable functions between domains with coherence [BE91b]. They also use the accepted notion of first-order sequentiality — KP-sequentiality — and equip domains with a parameterized structure that makes it possible to give a different treatment at higher order. But it would seem that their approach does not have the operational discipline that is encapsulated by index structures. Thus, stable index structures give rise to the coherence structure of critical sets, but not every coherence structure is generated in this way. In particular, the coherence structure that they use for the function space does not correspond to an index structure. Thus, they are able to obtain a ccc, with (what amounts to a use of) the simple product assumption. It remains to be seen whether additional axioms for coherence structures may identify those are the critical sets for some index structure. Technically, some of our development here — for instance, the limited results concerning the pointwise-ordered indexable function space, and the bi-ordered case — should transfer with little effort to the more general corresponding strongly stable function spaces.

The operational nature of our approach facilitates a proof of definability in the restricted case of ca-sequential computation, and it seems likely that a straightforward generalization of this proof will apply for general higher-order sequential computation, if other difficulties are successfully resolved. If this effort is indeed successful then it should provide a decision procedure for (equivalently, a means of enumerating) the elements of the sequential hierarchy over finite ground types, a proposed criterion for a satisfactory solution of the full abstraction problem [JS93].

We believe that the further refinement necessary to prevent the problem of partitioning, as outlined in section 7.5.1, should be applicable with relatively little impact on the technical development related to indexability. Ideally, it should take the form of employing a better substrate.

Section 7.4 left the compositionality of fully indexable functions as an unresolved conjecture. If the conjecture does not hold then it would seem that additional insights are necessary in order to integrate the ideas presented thus far into a satisfactory package. If it does hold, then the difficulty in proving it probably stems from the complexity of the constructions, and especially the need to assess sequentiality of functions in their fully uncurried form (sections 3.4.1, 5.2.3, and 7.1.8).

One way to alleviate the complexity might be to find a concrete representation for spaces, in the same sense that concrete data structures represent concrete domains. Initial starting points for such an investigation might be bistructures [Win94], and generalized concrete data structures [BG91].

Bibliography

[AJM94] S. Abramsky, R. Jagadeesan, and P. Malacaria. Full abstraction for PCF (extended abstract). In M. Hagiya and J. C. Mitchell, editors, *Theoretical Aspects of Computer Software*, number 789 in Lecture Notes in Computer Science, pages 1–15. Springer-Verlag, 1994.

[BC82] G. Berry and P.-L. Curien. Sequential algorithms on concrete data structures. *Theoretical Computer Science*, 20:265–321, 1982.

[BC85] G. Berry and P.-L. Curien. Theory and practice of sequential algorithms: the kernel of the applicative language CDS0. In M. Nivat and J. C. Reynolds, editors, *Algebraic Methods in Semantics*, chapter 2, pages 35–87. Cambridge University Press, 1985.

[BCL85] G. Berry, P.-L. Curien, and J.-J. Lévy. Full abstraction for sequential languages: the state of the art. In M. Nivat and J. C. Reynolds, editors, *Algebraic Methods in Semantics*, chapter 3, pages 89–132. Cambridge University Press, 1985.

[BE91a] A. Bucciarelli and T. Ehrhard. Extensional embedding of a strongly stable model of PCF. In *Proc. 18^{th} International Colloquium on Automata, Languages and Programming, Madrid*, number 510 in Lecture Notes in Computer Science, pages 35–47. Springer Verlag, 1991.

[BE91b] A. Bucciarelli and T. Ehrhard. Sequentiality and strong stability. In *Proc. Sixth Annual IEEE Symposium on Logic in Computer Science, Amsterdam.* IEEE Computer Society Press, July 1991.

[Ber76] G. Berry. Bottom-up computations of recursive programs. *RAIRO Informatique Theorique*, 10(3):47–82, March 1976.

[Ber78] G. Berry. Stable models of typed λ-calculi. In *Proc. 5^{th} Coll. on Automata, Languages and Programming*, number 62 in Lecture Notes in Computer Science, pages 72–89. Springer-Verlag, July 1978.

[Ber79] G. Berry. *Modèles Complètement Adéquats et Stables des Lambda-Calculs Typés.* Thèse de doctorat d'etat, Université Paris VII, March 1979.

[BG91] S. Brookes and S. Geva. Continuous functions and parallel algorithms on concrete data structures. In *Mathematical Foundations of Programming Semantics, 7^{th}*

International Conference, Carnegie Mellon University, Pittsburgh, Lecture Notes in Computer Science. Springer-Verlag, 1991.

[BG92] S. Brookes and S. Geva. Towards a theory of parallel algorithms on concrete data structures. *Theoretical Computer Science*, 101(2):177–221, July 1992. Also appeared as technical report CMU-CS-91-157, School of Computer Science, Carnegie-Mellon University.

[Buc95] A. Bucciarelli. Degrees of parallelism in the continuous type hierarchy. Technical Report LIENS-DMI 95/16, Ecole Normale Supérieure de Paris, 1995.

[BW90] M. Barr and C. Wells. *Category Theory for Computer Science*. Prentice Hall, 1990.

[Cur93] P.-L. Curien. *Categorical Combinators, Sequential Algorithms and Functional Programming*. Progress in Theoretical Computer Science. Birkhäuser, 1993. Second edition, expanded and updated.

[DP90] B. A. Davey and H. A. Priestley. *Introduction to Lattices and Order*. Cambridge University Press, 1990.

[GHK+80] G. Gierz, K. H. Hofmann, K. Keimel, J. D. Lawson, M. Mislove, and D. S. Scott. *A Compendium of Continuous Lattices*. Springer Verlag, 1980.

[HO94] J. M. E. Hyland and C.-H. L. Ong. On full abstraction for PCF. Draft Manuscript, October 1994.

[JM91] T. Jim and A. R. Meyer. Full abstraction and the context lemma. In T. Ito and A. R. Meyer, editors, *Theoretical Aspects of Computer Software*, number 526 in Lecture Notes in Computer Science, pages 131–151. Springer Verlag, September 1991.

[JS93] A. Jung and A. Stoughton. Studying the fully abstract model of PCF within the continuous function model. In *Proc. Int. Conf. Typed Lambda Calculi and Applications, Utrecht*, number 664 in Lecture Notes in Computer Science, pages 230–245. Springer Verlag, 1993.

[Kel82] G. M. Kelly. *Basic Concepts of Enriched Category Theory*. Number 64 in London Mathematical Society Lecture Notes Series. Cambridge University Press, 1982.

[KM77] G. Kahn and D. B. MacQueen. Coroutines and networks of parallel processes. In *Information Processing 1977*, pages 993–998. North Holland, 1977.

[KP78] G. Kahn and G. D. Plotkin. Domaines concrets. Rapport 336, IRIA-LABORIA, 1978. English translation published in the Böhm Festschrift special issue of *Theoretical Computer Science*, 121:187–277, 1993.

[Lam93] F. Lamarche. Stable domains are generalized topological spaces. *Theoretical Computer Science*, 111:103–123, 1993.

[Mil77] R. Milner. Fully abstract models of typed lambda-calculi. *Theoretical Computer Science*, 4:1–22, 1977.

[ML71] Saunders Mac Lane. *Categories for the Working Mathematician.* Springer-Verlag, 1971.

[Ong95] C.-H. L. Ong. Correspondence between operational and denotational semantics: the full abstraction problem for PCF. In S. Abramsky, D. Gabbay, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 4. Oxford Univ. Press, 1995. To appear.

[OR95] P. W. O'Hearn and J. G. Riecke. Kripke logical relations and PCF. *Information and Computation*, 120(1):107–116, July 1995.

[Pie91] B. C. Pierce. *Basic Category Theory for Computer Scientists.* MIT Press, 1991.

[Plo76] G. D. Plotkin. A powerdomain construction. *SIAM Journal on Computing*, 5(3):452–488, September 1976.

[Plo77] G. D. Plotkin. LCF considered as a programming language. *Theoretical Computer Science*, 5(3):223–255, 1977.

[PS87] P. Panangaden and V. Shanbhogue. On the expressive power of indeterminate primitives. Technical Report 87-891, Cornell University, Computer Science Department, November 1987.

[Saz76a] V. Yu. Sazonov. Degrees of parallelism in computations. In *Proc. Conference on Mathematical Foundations of Computer Science*, number 45 in Lecture Notes in Computer Science, pages 517–523. Springer-Verlag, 1976.

[Saz76b] V. Yu. Sazonov. Expressibility of functions in Scott's LCF language. *Algebra and Logic*, 15(3):308–330, 1976. (in Russian, English translation available).

[Saz76c] V. Yu. Sazonov. Functionals computable in series and in parallel. *Siberian Mathematical Journal*, 17(3):648–672, 1976. (in Russian, English translation available).

[Sco69] D. S. Scott. A type theoretical alternative to CUCH, ISWIM, OWHY, 1969. Published in the Böhm Festschrift special issue of *Theoretical Computer Science*, 121:411–440, 1993.

[Sco70] D. S. Scott. Outline of a mathematical theory of computation. In *Proc. Fourth Annual Princeton Conf. on Information Sciences and Systems*, pages 169–176. Princeton University, 1970. A revised and slightly expanded version appeared as Technical Monograph PRG-2, Oxford University Computing Laboratory, Programming Research Group, November 1970.

[Sco72] D. S. Scott. Continuous lattices. In F. W. Lawvere, editor, *Toposes, Algebraic Geometry and Logic*, number 274 in Lecture Notes in Mathematics, pages 97–136. Springer Verlag, 1972. Also appeared as Technical Monograph PRG-7, Oxford University Computing Laboratory, Programming Research Group, 1971.

[Sto77] J. E. Stoy. *Denotational Semantics: The Scott-Strachey Approach to Programming Language Theory.* MIT Press, 1977.

[Sto88] A. Stoughton. *Fully Abstract Models of Programming Languages.* Research Notes in Theoretical Computer Science. Pitman, 1988.

[Sto90] A. Stoughton. Equationally fully abstract models of PCF. In *Mathematical Foundations of Programming Semantics, 5th International Conference, New Orleans, March-April 1989.* Springer-Verlag, 1990.

[Tra75] M. B. Trakhtenbtrot. On representation of sequential and parallel functions. In *Proc. 4th Symp. on Mathematical Foundations of Computer Science*, number 32 in Lecture Notes in Computer Science, pages 411–417, 1975.

[Tra85] B. A. Trakhtenbrot. Topics in typed programming languages. Class notes for graduate course at CMU, December 1985.

[Vic89] S. Vickers. *Topology via Logic.* Number 5 in Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1989.

[Vui73] J. Vuillemin. *Proof techniques for recursive programs.* PhD thesis, Stanford University, 1973.

[Win80] G. Winskel. *Events in Computation.* PhD thesis, University of Edinburgh, 1980.

[Win87] G. Winskel. Event structures. In *Petri Nets: Application and Relationships to Other Models of Concurrency*, number 255 in Lecture Notes in Computer Science, pages 325–392. Springer-Verlag, 1987.

[Win94] G. Winskel. Stable bistructure models of PCF. In *Proceedings of MFCS '94*, number 841 in Lecture Notes in Computer Science, 1994.

[Zha91] G. Q. Zhang. *Logic of Domains.* Progress in Theoretical Computer Science. Birkhäuser, Boston, 1991.