A Fully Abstract Model
for
Sequential Computation

Vom Fachbereich Mathematik
der Technischen Universität Darmstadt
zur Erlangung des akademischen Grades eines
Doktors der Naturwissenschaften (Dr. rer. nat.)
genehmigte Dissertation

von
Dipl.-Math. Michael Marz
aus Offenbach am Main

Referent:	Prof. Dr. Thomas Streicher
Korreferent:	Prof. Dr. Achim Jung
Tag der Einreichung:	9. Dezember 1999
Tag der mündlichen Prüfung:	21. Januar 2000

Darmstadt 2000
D 17

Acknowledgements

There are many people that supported me in writing this thesis. I would like to take this opportunity to say 'thank you'.

The research group *Logik und Mathematische Grundlagen der Informatik* at the mathematics department of the *Technische Universität Darmstadt* around Klaus Keimel was a very pleasant and stimulating environment for my research. I had plenty of useful discussions with all members, in particular with Martin Hofmann and Dirk Thierbach.

From October 1997 till September 1998 I spent one year at the *School of Computer Science* at the *University of Birmingham.* This was made possible by a grant from the *Gottlieb Daimler-und Karl Benz-Stiftung.* I had many interesting discussions, in particular with Mathias Kegelmann and Eike Ritter.

Furthermore, I am grateful to Martín Escardó for inviting me to the University of Edinburgh. I learned a lot from discussions with Gordon Plotkin and Marcello Fiore. Although we kept missing each other, A. Simpson made a big contribution to this work: he gave a useful hint concerning the definition of Kripke-logical relations on sums, besides, Proposition 3.22 is due to him.

Moreover, I had some very helpful discussions with Peter O'Hearn who pointed out a mistake in the definition of Kripke-logical relations on smash products. I also enjoyed the never ending e-mail arguments with Paul Levy about call-by-name, call-by-value, call-by-push and call-by-observable languages and combinations between them.

I say thank you to Stefan Brötz, Dirk Thierbach, Gernot Belger and Jochen Tappe for their careful proof reading.

Very special thanks go to my supervisors. Achim Jung invited me to come to Birmingham and was very helpful not only in supervising me, but also in finding accommodation and sorting out similar things. I am also grateful that I had the opportunity to do some teaching, which was a very interesting experience as it gave me some insights into a very different education system.

Last, but not least, I thank Thomas Streicher for spending so many hours in exhaustive (and often exhausting) discussions with me and for his patience. He had many ideas and it is no exaggeration to say that this thesis would hardly exist without his supervision. It was also very inspiring to pick up some bits of the rich vocabulary of Austrian mathematics that seems to consist of *urpatscherten, geruchelten Konstruktionen von Dopösern, die auf hatscherte Art und Weise oft die Kardinalkonfusion beianand bringen—eh kloa!*

Contents

1 Introduction

1.1 Some Introductory and Historical Remarks

When are two programs equal? The problem of comparing programs of a given programming language goes back to the very beginning of programming and has many different answers. From a very restrictive to a very liberal notion of equality, there are several possible ways of addressing this problem:

- Do programs have exactly the same code?
- Do programs have in principal the same code and differ only in the names of variables?
- Are they (possibly different) implementations of the same algorithm?
- Do they compute the same results, perhaps with different complexity behaviour?
- Do they compute the same results only for a certain kind of input which is in some sense relevant?

On the one hand, no two programs written by two different people are syntactically identical, on the other hand, there can hardly be a general criterion for 'relevant' input. A good criterion for distinguishing programs seems to be the following: two programs are considered to be 'really' different if, being given the same input, they produce different results.

Operational and Denotational Semantics

An appropriate way of studying programs is looking at their semantic behaviour. Talking about programming, there are two main kinds of semantics: *Operational Semantics* and *Denotational Semantics*.

Operational Semantics describes how programs are evaluated. It is often used to fix a notion of *observational equivalence* as, for instance, programs can be compared by studying their termination behaviour with respect to the operational semantics in certain contexts. This approach clearly fits with the above notion of comparing programs: two programs are considered observationally equal if, whenever given the same input, they produce the same output.

Denotational Semantics is different. Its primary aim is to describe the mathematical meaning of programs. As pointed out by P. Mosses [Mos90], it does not specify the techniques to be used in programming, but it establishes a standard for implementations—ensuring that each program gives essentially the same results on all implementations that are conform to the standard. In order to obtain a syntax independent notion, programs are interpreted in a (mathematical) model. Given a typed programming language, there are always trivial models such as the singleton model in which all programs of the same type are interpreted equally or the term model in which type denotations are sets of terms and every program is interpreted by itself. The two of these models are instances of what is called *set model* in which program types are interpreted as sets and the actual code is interpreted as a function from the interpretation of the input type to the one of the output type. Various machine independent notions of how to compare programs can be addressed by this approach, depending on how the denotational semantics is defined.

As already pointed out, Operational Semantics gives rise to notions of observational equivalence of programs, i. e. two programs are considered equal if their termination behaviour cannot be observed to be different on certain observation types (for instance on ground types) with respect to the evaluation rules. Such a notion of equivalence is normally less concrete than denotational equality as equally interpreted programs should not be observed to be different. On the other hand, Operational Semantics is more concrete than Denotational Semantics in the sense that two programs are interpreted equally in a denotational model if they can be proved to be equivalent using the evaluation rules. In general, these comparisons are sharp, i. e. there may be observationally equivalent programs that are interpreted as different elements in the model and there may be an element in the model the occur as interpretations of two different values (for instance, the programs $\lambda x.x$ and $\lambda x.(\lambda y.y)x$ in PCF).

See [Mos90] for many further remarks on the history of Denotational Semantics and its usage in the theory of programming languages.

Computational Adequacy and Full Abstraction

Given a programming language with an operational and a denotational semantics, the question arises how they fit together. Or, from a different point of view, is it possible to find an appropriate denotational semantics for a language with its already specified operational semantics? Or, as a third approach, given a nice mathematical structure, is there a language with an operational semantics describing it accurately?

Now, what does 'fitting together', 'appropriate' and 'accurately' mean? Firstly, the semantics does not seem to work well if the following is not satisfied: two denotationally equal programs should behave in the same way, i. e. they are supposed to be observationally equal with respect to an operational semantics. This property is called *computational adequacy*. Conversely, a denotational model is called *fully abstract* if observationally equal programs are interpreted the same. This is a much stronger requirement than one might expect and, as we will see later, quite natural looking models turn out to be not fully abstract.

Domains

To begin with, partially ordered sets form a suitable mathematical structure for interpreting programs. A program is considered to be 'better' than another one if it is more powerful, i. e. it computes everything the weaker one is able to and maybe even more. As it is possible that one program can compute results only for certain input and another one only for different input and as the results are usually different, the underlying partial order is in general not linear. Furthermore, there is always the most 'stupid' program of a certain type that never computes anything. Hence, for a complete denotational semantics we focus on partially ordered sets with a least element. Finally, suppose there is an unbounded chain of programs that are more and more powerful. For example, think of programs P_i ($i \in \mathbb{N}$) defining a function $f\colon \mathbb{N} \to \mathbb{N}$ where the i-th program produces the correct output only for the first i numbers and nothing else. It is then natural to think of a program that computes the correct result $f(n)$ for all numbers $n \in \mathbb{N}$ as the least upper bound of all programs P_i's—whether this program actually exists is a different question. These requirements lead to the definition of *domains*: a domain is a partially ordered set with a least element which is closed under directed suprema (the precise definition of 'directed' will be given later).

Domain Theory was first introduced in an unpublished, but nevertheless very influential manuscript by D. Scott in 1969 [Sco69], finally published 24 years later [Sco93]. His intention was to provide a mathematical foundation for program verification. At the same time, Y. Ershov examined domains as a tool for his *Theory of Enumerations*, see e.g. [Ers75], not being aware of Scott's results.

An introduction to Domain Theory covering most of the fundamental concepts used in this thesis is [GS90]. A more mathematically motivated approach towards domains can be found in [AJ94], computer scientists may prefer [Plo81] as a reference.

For an overview article addressing the role of Domain Theory in Denotational Semantics with many references, see [JFM$^+$96].

The Language PCF

A very simple functional programming language is the sequential language PCF (programming computable functionals). In this context, the expression *sequentiality* describes the fact that, while running a program, all computations are done in a fixed order. In particular, once a computation of a subprogram has been started, it has to be finished (if possible) and cannot be aborted in order to start another one. This means that PCF does not support parallel computations, i.e. it is not possible to start calculating several subprograms at once, waiting for one of them to finish.

The language PCF was developed by D. Scott as a calculus for studying logical properties of programs, originally named LCF [Sco69, Sco93]. It can be considered as the simply typed λ-calculus—i.e. it has abstraction and application—augmented by basic arithmetic and a fixed point combinator for every type, supporting recursive definitions. Stimulated by Scott's seminal article, the theory of PCF became a popular research area in the early seventies. V. Sazonov gave a characterisation of PCF-definable functions [Saz75, Saz76]. The first fully abstract model for PCF—which is in some sense up to isomorphism even uniquely determined—was found by R. Milner [Mil77]. However, Milner's presentation was based on terms and purely syntactic, hence, it did not provide a 'proper' semantic description of sequentiality. In the same year, G. Plotkin reformulated PCF as a programming language and introduced the new name [Plo77]. He pointed out the problem of finding a syntax free fully abstract model for PCF. This question turned out to be one of the most enduring problems of semantics.

An introduction to PCF-style languages with their operational semantics can be found in [Gun92, Win93].

The Scott model

The traditional domain theoretic model for PCF is the *Scott model*, also called *standard model*. The natural number type is interpreted as the *flat* domain $\mathbb{N}_\perp$ of natural numbers augmented by a least element $\perp$ ('*bottom*') where all numbers are maximal elements (as a result of a computation the number 3 is neither better nor worse than the number 5). The Boolean type is interpreted as the three element domain $\mathbb{B}_\perp$ with two maximal elements true and false and a bottom element. Function spaces in the Scott model consist of functions preserving directed suprema, called *Scott continuous* functions.

The Scott model is computationally adequate, however, it is not fully abstract as already observed in [Sco69]. The problem is that, whereas PCF is a sequential language, also inherently parallel functions are part of the model. Consider, for instance, the parallel-or function $\mathsf{por}\colon \mathbb{B}_\perp \times \mathbb{B}_\perp \to \mathbb{B}_\perp$ where $\mathsf{por}(x, y)$ is true, if at least one of the arguments x or y is true, false if both arguments are false and $\perp$ otherwise. In contrast to usual sequential implementations of or-functions in programming languages where typically one argument is evaluated completely before the second one may be considered (semantically rephrased as 'undefined or true equals undefined'), the behaviour of por is different as it computes both arguments in parallel.

It is well known that parallel-or is not definable in PCF—a generalisation of this claim will be proved later in this thesis. However, it is easy to implement a program **portest** of type (**boolean** $\times$ **boolean** $\to$ **boolean**) $\to$ **boolean** that returns true when applied to a program defining the parallel-or function and that diverges if not (which corresponds to $\perp$ in the model). Since por is not definable in PCF, **portest** cannot be observed to be different from the program Ω that diverges for any input. However, as por is a Scott continuous function, **portest** and Ω are interpreted differently and, hence, the Scott model is not fully abstract for PCF.

As shown by G. Plotkin [Plo77], the Scott model is fully abstract for PCF augmented by a parallel-or constructor. An alternative proof, which goes back to A. Meyer, can be found in [Mit96].

The Full Abstraction Problem

As already mentioned, the first presentations of the fully abstract model for PCF [Mil77, Mul87] have a strong syntactic flavour and do not provide a satisfactory answer to the full abstraction problem. Even worse, R. Loader showed that observational equivalence is not decidable even for finitary PCF, i. e. PCF based on type **boolean** [Loa96]. In particular, there cannot be a uniform criterion to decide what functions are admitted in a fully abstract model. In other words, a syntax free presentation of the fully abstract model must be looking really complicated.

In order to give a domain theoretic description of a fully abstract model for PCF, the Scott model fails as it contains too many functions such as the parallel-or function. In 1992, K. Sieber presented a syntax free characterisation of sequential functions for types up to order two and proved a full abstraction result for PCF types up to order three [Sie92]. Two years later, P. O'Hearn and J. Riecke used Sieber's technique and gave the first domain theoretic description of a fully abstract model for PCF [OR95b] by applying the concept of *Kripke relations* [JT93], a generalisation of *logical relations* going back to G. Plotkin [Plo73, Plo80]. The basic idea is the following: domains are equipped with an infinite family of relations of varying arity and only those Scott continuous functions preserving these relations are admitted in the model. It is then possible to choose a particular relation which is not preserved by por, hence, por is not a function in the model.

J. Riecke and A. Sandholm modified this approach to present a fully abstract model of FPC, a call-by-value-language with recursive types and sum types [RS97]. The main part of this thesis is a simplification of Riecke's and Sandholm's model and an extension of their result to a general call-by-name language encompassing the call-by-value paradigm.

A different successful approach towards the full abstraction problem was made by several research groups elaborating fully abstract models for PCF by using *Game Semantics* [AJM94, HO95, Nic94], G. McCusker extended this approach to sum types and recursive types [McC96]. A brief comparison between the game model and the domain theoretic approach can be found in the last section.

There is some more work that deserves mention. Although failing to construct a fully abstract model for PCF, many new techniques were developed while studying the phenomenon of sequentiality. For instance, G. Berry's *stable functions* model [Ber79] imposes further order theoretic conditions in the hope to cut down the spaces of Scott continuous functions to just the sequential ones, however, he does not capture sequentiality properly. Another idea is to construct a model where terms are interpreted as abstract computations, called *sequential algorithms* [Cur86, Cur93]. This model is not *extensional*, i.e. there are terms of the same type whose interpretations, although being different in the model, cannot be distinguished by applying them to arguments. Forcing extensionality, A. Bucciarelli and T. Ehrhard constructed the *extensional collapse* of Curien's sequential algorithms [BE91], however, this new model cannot be presented in pure order theoretic terms. An extension of sequential algorithms to *observable algorithms* [CCF94] provides a fully abstract model for PCF augmented by error values.

More on the full abstraction problem for PCF can be found in an exhaustive survey article by L. Ong, illustrating the different approaches until the early nineties [Ong95].

Category Theory

Categories provide a suitable language for Denotational Semantics. Basically, a category consists of *objects* and *morphisms* between them, satisfying some elementary conditions. Typical examples of categories are classes of mathematical structures such as groups with group homomorphisms or topological spaces with continuous functions. In particular, domains with functions preserving directed suprema, called *Scott continuous* functions, form a category as well.

As an example of how categories can be applied in semantics, we have a look at some intuition behind the idea of *cartesian closed categories*, the precise definition is given later. For any two objects A and B in a cartesian closed category $\mathcal{C}$, there exists the product $A \times B$ of A and B as an object, moreover, there is another object B^A in $\mathcal{C}$ which has properties making it resemble the collection of morphisms from A to B. In particular, there is a morphism $\mathsf{eval}\colon B^A \times A \to B$ 'evaluating' the result $\mathsf{eval}(f, a) = f(a)$ for every $f \in B^A$ and $a \in A$. As products and functions are very fundamental components of programming languages, cartesian closed categories arise as natural structures for describing models.

There are a lot of other features of programming languages including sum types, recursive types and even polymorphic types that can be formulated and dealt with categorically. See [Cro93] for an introduction to basic Category Theory and its applications in Denotational Semantics.

1.2 Organisation of the Thesis

After this preliminary part (Section 1), Section 2 gives an introduction to the domain theoretic concepts and notations used in this work.

A cartesian closed category *SD* of domains is constructed in Section 3, capturing the concept of sequentiality. It follows the relational approach in [OR95b], simplifying and continuing the work in [RS97]. Morphisms in *SD* are continuous functions between domains satisfying the additional property of being *uniform*. The category *SD* is the first cartesian closed category based on domains that hosts a fully abstract model for sequential functional languages with sum types. Moreover, it has *smash products* making the subcategory $SD_\perp$ with strict functions symmetric monoidal closed. This is important in order to provide a model not only for call-by-name, but also for call-by-value languages. By specifying the class of *SSB-objects* we get a semantic characterisation of those objects which are possible denotations of *computational types* of a programming language, i. e. types for which it is sequentially semidecidable whether their terms are not interpreted as bottom. We discuss several properties of *SD* and show how to construct $SD_\perp$ as a category of algebras over the subcategory of SSB-objects.

In Section 4, we have a look at the sequential language SFL (<u>s</u>equential <u>f</u>unctional <u>l</u>anguage), a generalisation of PCF with a host of important data type constructors (two forms of sums and products, lifting, strict and ordinary function spaces and recursive types). This is a new language that is close to the mathematical structure of domains and, hence, can be used as a metalanguage for (non-polymorphic) sequential programming languages. An operational semantics is defined for SFL, showing that SFL is powerful enough to deal with call-by-name evaluation strategies as well as call-by-value strategies. Furthermore, we have a look at some concrete data types in SFL. The section finishes with an equational theory expressing β-η-equivalence for SFL terms.

The interpretation of SFL in the category *SD* is discussed in Section 5. In particular, SFL types are interpreted as Scott domains. We prove that SFL is computationally adequate with respect to its interpretation and that *SD* gives rise to a fully abstract model for SFL. This is the main result of the thesis as the category *SD* provides the first fully abstract domain theoretic model for a language with sum types and recursive types that subsumes call-by-name and call-by-value evaluation strategies.

Whereas SFL can be considered as a call-by-name language augmented by call-by-value features, the language SFPL (<u>s</u>equential <u>f</u>unctional <u>p</u>rogramming <u>l</u>anguage) looks more like a call-by-value functional programming language subsuming the call-by-name paradigm. This language is introduced in Section 6. It is equally expressive as SFL and can be modelled in *SD* in a fully abstract way.

Section 7 finishes the thesis with a discussion of the results.

2 Fundamental Domain Theoretic Concepts and Notations

This section gives a short summary of the Domain Theory used in this thesis. More details can be found, for instance, in [AJ94] or [GS90]. As this work is not mainly about categories, we mostly restrict ourselves to look at categorical notions only in connection with domains and not in the most general way. See [Mac71] for a detailed introduction to Category Theory.

2.1 Category Theory

Categories

A *category* $\mathcal{C}$ is a collection of *objects* and *morphisms*, each morphism $f\colon A \to B$ having its uniquely determined domain A and codomain B. We usually write $\mathsf{Obj}(\mathcal{C})$ for the objects of $\mathcal{C}$ and $\mathcal{C}(A, B)$ for the collection of morphisms from A to B in $\mathcal{C}$. A binary operation $\circ$ composes morphisms in the following way: if $f\colon A \to B$ and $g\colon B \to C$ are morphisms, then so is $g \circ f\colon A \to C$; this operation has to be associative. Finally, for every object A, there is a unique morphism $\mathsf{id}_A\colon A \to A$ with $\mathsf{id}_A \circ f = f$ and $g \circ \mathsf{id}_A = g$ for all morphisms $f\colon B \to A$ and $g\colon A \to B$.

In every category in this thesis, objects are sets (usually equipped with some structure) and morphisms are functions between them.

A category $\mathcal{D}$ is called *subcategory* of $\mathcal{C}$ if all objects and morphisms in $\mathcal{D}$ are also objects and morphisms, respectively, in $\mathcal{C}$. It is a *full subcategory* if, for all objects $A, B \in \mathsf{Obj}(\mathcal{D})$, the morphisms $\mathcal{D}(A, B)$ are the same as $\mathcal{C}(A, B)$. Given a category $\mathcal{C}$, we can obtain its *dual* category $\mathcal{C}^{op}$ by reverting the arrows. If $\mathcal{D}$ is another category, then $\mathcal{C} \times \mathcal{D}$ defines the *product category* of $\mathcal{C}$ and $\mathcal{D}$: objects and morphisms in $\mathcal{C} \times \mathcal{D}$ are pairs of objects and morphisms in $\mathcal{C}$ and $\mathcal{D}$. Analogously, one can define the n-ary product $\mathcal{C}_1 \times \cdots \times \mathcal{C}_n$ of categories $\mathcal{C}_1, \ldots, \mathcal{C}_n$.

Functors

Let $\mathcal{C}$ and $\mathcal{D}$ be categories. A *covariant functor* $F\colon \mathcal{C} \to \mathcal{D}$ maps objects in $\mathcal{C}$ to objects in $\mathcal{D}$ and morphisms $f\colon A \to B$ in $\mathcal{C}$ to morphisms $Ff\colon FA \to FB$ in $\mathcal{D}$ such that the following conditions are satisfied:

- $F\mathsf{id}_A = \mathsf{id}_{FA}$ holds for all objects A in $\mathcal{C}$,
- $F(f \circ g) = Ff \circ Fg$ holds for all morphisms $f\colon B \to C$ and $g\colon A \to B$ in $\mathcal{C}$.

A functor $F\colon \mathcal{C} \to \mathcal{D}$ is called *contravariant* if it is a covariant functor from $\mathcal{C}^{op}$ to D. Using product categories allows us to consider functors of arity n (for every $n \in \mathbb{N}$) of the form $F\colon \mathcal{C}_1 \times \cdots \times \mathcal{C}_n \to \mathcal{C}$ (for categories $\mathcal{C}_1, \ldots, \mathcal{C}_n, \mathcal{C}$) in the same way as unary ones. An n-ary functor is called *mixed variant* or *bivariant*, if it is covariant in some arguments and contravariant in others. In particular, we write $F\colon \mathcal{C}^{op} \times \mathcal{C} \to \mathcal{C}$ to indicate that the binary functor F on $\mathcal{C}$ is mixed variant.

2.2 Domains

The category that is constructed in this thesis as a contribution to the full abstraction problem appears as a modification of the category of domains and Scott continuous functions. These concepts will be defined next.

A nonempty set D with a partial order $\sqsubseteq$ is called *directed* if each two elements $x, y \in D$ have an upper bound in D, i.e. there is an element $z \in D$ with $x \sqsubseteq z$ and $y \sqsubseteq z$. The supremum of a directed set D is usually denoted by $\bigsqcup^{\uparrow} D$ instead of $\bigsqcup D$, emphasising the directedness of D. A partially ordered set A is called *domain* if it has a least element $\perp_A$ and if every directed subset $D \subseteq A$ has a supremum $\bigsqcup^{\uparrow} D$. When it is clear from the context, we often omit the index and write $\perp$ instead of $\perp_A$. Simple examples of domains are finite partially ordered sets with a least element or *flat domains* where every element except $\perp$ is maximal.

A function $f\colon A \to B$ is called *Scott continuous* or just *continuous* if it preserves directed suprema, i.e. for every directed subset D of A, $f(\bigsqcup^{\uparrow} D)$ is directed and $f(\bigsqcup^{\uparrow} D) = \bigsqcup^{\uparrow} f(D)$ holds. In particular, in order to verify continuity it does not suffice to check that $f(\bigsqcup^{\uparrow} D)$ is the supremum of $f(D)$, one has to show the directedness of $f(D)$ as well. Equivalently, one can prove that f is monotone, which implies that $f(D)$ is directed if D is directed. We write $f\colon A \circ\!\!\to B$ to indicate that a function is *strict*, i.e. it maps $\perp_A$ to $\perp_B$. (Strict functions can be used as interpretations of those programs that do not terminate if their input does not.) The largest strict function $\mathsf{str}(f)\colon A \circ\!\!\to B$ below a function $f\colon A \to B$ is called the *strictification* of f, it maps $\perp_A$ to $\perp_B$ and all other elements $a \in A$ to $f(a)$. A function $f\colon A \to B$ is called *total*[1] if it reflects the bottom element, i.e. $f(x) = \perp_B$ implies $x = \perp_A$; in this case we write $f\colon A \to_t B$ and $f\colon A \circ\!\!\to_t B$, respectively. When ordered pointwise, the set of all continuous functions between two domains A and B forms a domain itself, it is denoted by $[A \to B]$, analogously, we write $[A \circ\!\!\to B]$ for the domain of all strict continuous functions. It can be easily checked that every continuous function $f\colon A \to A$ between two domains has a least fixed point $\mathsf{fix}(f) := \bigsqcup^{\uparrow}{}_{n\in\mathbb{N}} f^n(\perp_A) \in A$.

If $r\colon w \to A$ is a function from a set w to a domain A and if v is a subset of w, then the *restriction* $r \restriction v\colon w \to A$ is defined by $(r \restriction v)(x) = r(x)$ if $x \in v$ and $(r \restriction v)(x) = \perp_A$ otherwise. Since the domains of r and $r \restriction v$ are the same, this is a slight modification of the usual meaning of 'restriction'. We use $\lambda x{:}A.e[x]$ or $\lambda x^A.e[x]$ (where $e[x]$ is an expression that may contain x) for abstraction on the meta-level, i.e. $\lambda x{:}A.e[x]$ stands for the function taking an element $x \in A$ to $e[x]$.

It is not difficult to check that domains and continuous functions form a category, this category is called DOM. By restricting the morphisms to strict functions we get the subcategory $\mathit{DOM}_\perp$. The category of domains with strict total functions as morphisms is denoted by DOM_t. Two of these categories, DOM and $\mathit{DOM}_\perp$, are *cpo-enriched* which means that the sets of morphisms $\mathit{DOM}(A, B)$ and $\mathit{DOM}_\perp(A, B)$, respectively, of two objects A and B (in DOM given by $[A \to B]$) are domains themselves and that the composition function $\circ$ is continuous. The categories DOM and $\mathit{DOM}_\perp$ have the singleton domain $\mathbf{1}$ as a *terminal object*, i.e. there is a unique morphism from A to $\mathbf{1}$ for each

[1] This notion differs from other domain theoretic notions of totality such as Ulrich Berger's *total elements*, [Ber90].

domain A. The terminal object for DOM_t is the *Sierpiński domain* $\mathbf{2} := \{\bot, \top\}$ with two elements. Moreover, since each two morphisms $f, g\colon A \to B$ are equal if, and only if, $f \circ e = g \circ e$ holds for every morphism $e\colon \mathbf{1} \to A$, the category DOM is *well-pointed.* Analogously, the DOM_t is well-pointed as well. Since, for all $e\colon \mathbf{1} \multimap A$ and $f\colon A \multimap B$, $f \circ e$ is the constant bottom function, the category $DOM_\bot$ is not well-pointed.

2.3 Finitary Constructions of Domains

Cartesian Products

The *cartesian product* $A \times B$ of two domains A and B is given by the componentwise ordered set $\{\langle a, b\rangle \mid a \in A,\ b \in B\}$. It is the categorical product in DOM, i.e. it satisfies the following universal property:

$$\begin{tikzcd}
 & C \arrow[dl, "f"'] \arrow[d, dotted, "\exists!\ \langle f, g\rangle"] \arrow[dr, "g"] & \\
 & A \times B \arrow[dl, "\mathsf{pr}_1"] \arrow[dr, "\mathsf{pr}_2"'] & \\
A & & B
\end{tikzcd}$$

where pr_1 and pr_2 are the projections. This diagram is to be read in the following way: for any two morphisms $f\colon C \to A$ and $g\colon C \to B$ there is a unique morphism $\langle f, g\rangle\colon C \to A \times B$ making the diagram commute, i.e. $\mathsf{pr}_1 \circ \langle f, g\rangle = f$ and $\mathsf{pr}_2 \circ \langle f, g\rangle = g$. It is easy to verify that $\times\colon DOM \times DOM \to DOM$ is a covariant binary functor where, for morphisms $f\colon A \to A'$ and $g\colon B \to B'$, $f \times g\colon A \times B \to A' \times B'$ is defined componentwise.

Function Spaces, Cartesian Closed Categories

The category DOM is *cartesian closed* which means that it has a terminal object and that, for every object C, the functor $\cdot \times C$ (taking A to $A \times C$ and $f\colon A \to B$ to $\langle f, \mathsf{id}_C\rangle\colon B \times C$) has a right adjoint. This is given by the functor $\cdot^C$ where $B^C = [C \to B]$ is the *exponential* in DOM. More concretely, for every objects A, B, C and for each morphism $f\colon (A \times C) \to B$, there is a unique morphism $\mathsf{curry}\, f\colon A \to B^C$ such that the following diagram commutes:

$$\begin{tikzcd}
B^C & B^C \times C \arrow[r, "\mathsf{eval}"] & B \\
A \arrow[u, dotted, "\exists!\ \mathsf{curry}\, f"] & A \times C \arrow[u, "\mathsf{curry}\, f \times \mathsf{id}_C"] \arrow[ur, "\forall f"'] &
\end{tikzcd}$$

In DOM, the morphism $\mathsf{curry}\, f$ is given by the map which maps an element a of A to the function $\lambda c{:}C.f(\langle a, c\rangle)$; the evaluation function eval takes $(f, c) \in B^C \times C$ to $f(c)$. It is easy to check that all functions involved are continuous and, therefore, morphisms in DOM. The binary functor that maps two domains A and B to $B^A = [A \to B]$ is an example of a mixed variant functor on DOM. If A, A', B, B' are domains and $f\colon A' \to A$ and $g\colon B \to B'$ are morphisms, then $g^f\colon [A \to B] \to [A' \to B']$ is given by $g^f(h) = g \circ h \circ f$.

It has already been argued in Section 1 why cartesian closed categories are suitable for modelling programming languages.

Liftings

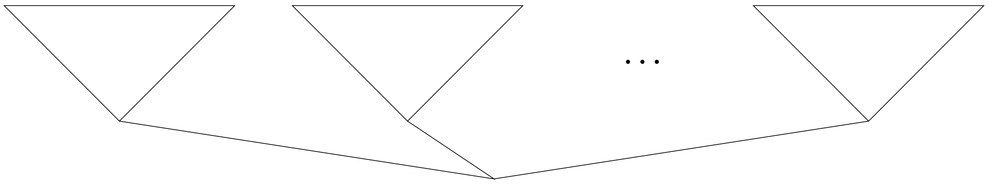
The *lifting* $A_\perp$ of a domain A is the simple result of adding a new bottom element to A. Associated with it are continuous functions $\mathsf{up}_A\colon A \to A_\perp$ and $\mathsf{down}_A\colon A_\perp \circ\!\!\rightarrow A$, defined by $\mathsf{down}_A(\mathsf{up}_A\, x) = x$ and $\mathsf{down}_A(\perp_{A_\perp}) = \perp_A$. Thus, $A_\perp$ is the set $\{\mathsf{up}_A\, a \mid a \in A\} \cup \{\perp_{A_\perp}\}$ with the order induced by A. Besides, if $f\colon A \to B$ is a continuous function in DOM, then so is $f_\perp\colon A_\perp \circ\!\!\rightarrow_t B_\perp$ with $f_\perp(\mathsf{up}_A\, a) = \mathsf{up}_B\, f(a)$ and $f_\perp(\perp_{A_\perp}) = \perp_{B_\perp}$. Hence, the lifting operator is a functor on the category DOM.

Separated Sums

The *separated sum* $\sum_{i=1}^n A_i = A_1 + \cdots + A_n$ of domains $A_1, \ldots, A_n$ is given by the set $\{(a : i) \mid i = 1, \ldots, n,\ a \in A_i\} \cup \{\perp_{\sum_{i=1}^n A_i}\}$, the order being defined by

$$(a : i) \sqsubseteq_{\sum_{i=1}^n A_i} (b : j) \iff i = j \text{ and } a \sqsubseteq_{A_i} b.$$

Separated sums can be visualised by the following picture:



In particular, a separated sum with only one summand A is the lifting $A_\perp$. Separated sums satisfy the universal property

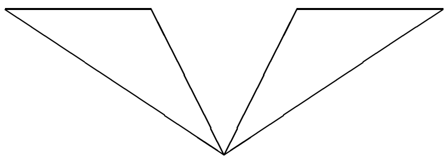
$$\begin{tikzcd}
A_1 \arrow[ddrr, "\mathsf{in}_1"] \arrow[ddddrr, "f_1"'] & A_2 \arrow[ddr, "\mathsf{in}_2"] \arrow[ddddr, "f_2"'] & \cdots & \cdots & A_n \arrow[ddll, "\mathsf{in}_n"'] \arrow[ddddll, "f_n"] \\
\\
& & \sum_{i=1}^n A_i \arrow[dd, dotted, "\exists! [f_1, \ldots, f_n]"] & & \\
\\
& & C & &
\end{tikzcd}$$

where, for all $i \in \{1, \ldots, n\}$, the injection $\mathsf{in}_i\colon A_i \to \sum_{i=1}^n A_i$ is defined by $\mathsf{in}_i(a) = (a : i)$. This means that, for each n morphisms $f_i\colon A_i \to C$ $(i = 1, \ldots, n)$, there is exactly one strict morphism $[f_1, \dots, f_n]\colon \sum_{i=1}^{n} A_i \to C$ such that the diagram commutes. (Note that there may be several non-strict maps making the diagram commute.) It is easy to check that the injections in_i and the mediating morphism $[f_1, \dots, f_n]$ are all continuous. Besides, the separated sum is an n-ary functor on DOM, the according morphism for $g_i\colon A_i \to B_i$ $(i = 1, \dots, n)$ is given by $\sum_{i=1}^{n} g_i := [\mathsf{in}_1 \circ g_1, \dots, \mathsf{in}_n \circ g_n]\colon \sum_{i=1}^{n} A_i \to \sum_{i=1}^{n} B_i$. The functoriality of the separated sum follows from the universal property as exhibited in the diagram.

For the elements of separated sums, we usually write $\mathsf{in}_i\, a$ for $(a : i)$.

Coalesced Sums

The *coalesced sum* $A \oplus B$ of domains A and B is the quotient of $A + B$ where $\mathsf{in}_1(\perp_A)$, $\mathsf{in}_2(\perp_B)$ and $\perp_{A+B}$ are identified:



Coalesced sums satisfy a universal property, too, they are categorical sums in the subcategory $DOM_\perp$ of DOM:

$$\begin{tikzcd} A \arrow[dr, "\widetilde{\mathsf{inl}}"] \arrow[ddr, "f"'] & & B \arrow[dl, "\widetilde{\mathsf{inr}}"'] \arrow[ddl, "g"] \\ & A \oplus B \arrow[d, dotted, "\exists!\,\widetilde{[f,g]}"'] & \\ & C & \end{tikzcd}$$

where $\widetilde{\mathsf{inl}}$, $\widetilde{\mathsf{inr}}$ are strict injections defined analogously to the separated case. Hence, the elements of $A \oplus B$ can be denoted by $\widetilde{\mathsf{inl}}\, a$, $\widetilde{\mathsf{inr}}\, b$ and $\perp_{A \oplus B}$ $(a \in A,\ b \in B)$. In this diagram, the strictness of f and g is crucial. Otherwise, the mediating morphism

$$\widetilde{[f,g]} \;=\; \lambda x{:}A{\oplus}B. \begin{cases} f(a) & \text{if } x = \widetilde{\mathsf{inl}}\, a \neq \perp \\ g(b) & \text{if } x = \widetilde{\mathsf{inr}}\, b \neq \perp \\ \perp & \text{if } x = \perp \end{cases}$$

does not necessarily make the diagram commute. As already suggested, the operation $\oplus\colon DOM_\perp \times DOM_\perp \to DOM_\perp$ is a functor on the subcategory $DOM_\perp$ where the action on morphisms $f\colon A \circ\!\!\to A'$ and $g\colon B \to B'$ is given by

$$f \oplus g \;:=\; \widetilde{[\widetilde{\mathsf{inl}} \circ f, \widetilde{\mathsf{inr}} \circ g]}.$$

It is worth noticing that $\oplus$ is not a functor on DOM. If we define $f \oplus g$ in the obvious way by setting $(f \oplus g)(\widetilde{\mathsf{inl}}\,a) = \widetilde{\mathsf{inl}}\,f(a)$, $(f \oplus g)(\widetilde{\mathsf{inr}}\,b) = \widetilde{\mathsf{inr}}\,g(b)$ $(a, b \neq \bot)$ and $(f \oplus g)(\bot) = \bot$ for all functions $f\colon A \to A'$, $g\colon B \to B'$, then $\oplus$ turns out to be not functorial. Let $c_{\top_\mathbf{2}} = \lambda x{:}\mathbf{2}.\top_\mathbf{2}\colon \mathbf{2} \to \mathbf{2}$ denote the constant $\top$-function and $c_{\bot_\mathbf{2}} = \lambda x{:}\mathbf{2}.\bot_\mathbf{2}\colon \mathbf{2} \to \mathbf{2}$ the constant $\bot$-function. Then $(c_{\top_\mathbf{2}} \circ c_{\bot_\mathbf{2}}) \oplus (\mathsf{id}_\mathbf{2} \circ \mathsf{id}_\mathbf{2}) = (c_{\top_\mathbf{2}} \oplus \mathsf{id}_\mathbf{2})$ maps $\mathsf{inl}\,\top_\mathbf{2}$ to $\mathsf{inl}\,\top_\mathbf{2}$ whereas $(c_{\top_\mathbf{2}} \oplus \mathsf{id}_\mathbf{2}) \circ (c_{\bot_\mathbf{2}} \oplus \mathsf{id}_\mathbf{2})(\mathsf{inl}\,\top_\mathbf{2}) = (c_{\top_\mathbf{2}} \oplus \mathsf{id}_\mathbf{2})(\bot_{\mathbf{2}\oplus\mathbf{2}}) = \bot_{\mathbf{2}\oplus\mathbf{2}}$ holds.

Since the coalesced sum operator is associative, we can easily construct arbitrary finitary coalesced sums from binary ones:

$$\bigoplus_{i=1}^{n} A_i \;=\; A_1 \oplus A_2 \oplus \cdots \oplus A_n \;:=\; \Big(\cdots\big((A_1 \oplus A_2) \oplus A_3\big) \oplus \cdots \oplus A_{n-1}\Big) \oplus A_n$$

Smash Products

In semantics for call-by-value programming languages, a pair of two programs is interpreted as bottom if already one its components diverges. Accordingly, it is occasionally desirable to have some sort of product in which elements of the form $\langle\bot,\bot\rangle$, $\langle a,\bot\rangle$ and $\langle\bot,b\rangle$ are identified. The *smash product* $A \otimes B$ of domains A, B does the job, it is the subset of those elements $\langle a, b\rangle \in A \times B$ that satisfy the condition $a = \bot_A \Leftrightarrow b = \bot_B$. The quotienting map is given by $\mathsf{smash}\colon A \times B \to A \otimes B$ and we write $(a, b) = \mathsf{smash}\big(\langle a, b\rangle\big)$. Smash products satisfy the following universal property:

$$\begin{tikzcd}
A \times B \arrow[r, "\mathsf{smash}"] \arrow[dr, "\forall f \text{ bistrict}"'] & A \otimes B \arrow[d, dotted, "\exists! \widetilde{f}"] \\
 & C
\end{tikzcd}$$

where f is called *bistrict* if $f\big(\langle\bot_A, b\rangle\big) = f\big(\langle a, \bot_B\rangle\big) = \bot_C$ holds for all $a \in A$, $b \in B$. It is easy to check that $\widetilde{f}$ is continuous, strict and unique. As $\otimes$ is associative and the Sierpiński domain $\mathbf{2}$ is a neutral element, smash products satisfy the categorical requirements for *tensor products*. We write (g, h) for the mediating morphism $\mathsf{smash} \circ \langle g, h\rangle$ for functions $g\colon D \multimap A$ and $h\colon D \multimap B$. In particular, the smash product construction $\otimes$ is a functor on the category $DOM_\bot$, however, it is not functorial on DOM. For example, the function $(c_{\top_\mathbf{2}} \circ c_{\bot_\mathbf{2}}) \otimes (\mathsf{id}_\mathbf{2} \circ \mathsf{id}_\mathbf{2}) = c_{\top_\mathbf{2}} \otimes \mathsf{id}_\mathbf{2} = \mathsf{id}_{\mathbf{2}\otimes\mathbf{2}}$ is different from $(c_{\top_\mathbf{2}} \otimes \mathsf{id}_\mathbf{2}) \circ (c_{\bot_\mathbf{2}} \otimes \mathsf{id}_\mathbf{2}) = \mathsf{id}_{\mathbf{2}\otimes\mathbf{2}} \circ c_{\bot_{\mathbf{2}\otimes\mathbf{2}}} = c_{\bot_{\mathbf{2}\otimes\mathbf{2}}}$.

Smash products in DOM have also projections $\widetilde{\mathsf{pr}_1}\colon A \otimes B \multimap A$ and $\widetilde{\mathsf{pr}_2}\colon A \otimes B \multimap B$. We prove that $\widetilde{\mathsf{pr}_1}$ is continuous, the proof for $\widetilde{\mathsf{pr}_2}$ goes analogously. Firstly, $\widetilde{\mathsf{pr}_1}$ is obviously monotone. Now assume that $\big((a_i, b_i)\big)_{i\in I}$ is a directed family of elements in $A \otimes B$ with $\bigsqcup^{\uparrow}{}_{i\in I}(a_i, b_i) \neq \bot_{A\otimes B}$ (otherwise it is trivial). Then there is an element $i_0 \in I$ with $a_j \neq \bot_A$, $b_j \neq \bot_B$ for all $j \in I$ with $j \geq i_0$. This implies $\widetilde{\mathsf{pr}_1}\big(\bigsqcup^{\uparrow}{}_{i\in I}(a_i, b_i)\big) = \widetilde{\mathsf{pr}_1}\big(\bigsqcup^{\uparrow}{}_{j\geq i_0} a_i, \bigsqcup^{\uparrow}{}_{j\geq i_0} b_i\big) = \bigsqcup^{\uparrow}{}_{j\geq i_0} a_i = \bigsqcup^{\uparrow}{}_{i\in I} \widetilde{\mathsf{pr}_1}(a_i, b_i)$.

Strict Functions, Symmetric Monoidal Closed Categories

As for cartesian products and exponentiation in DOM, there is a relationship between smash products and strict exponentiation in $DOM_\perp$. The category $DOM_\perp$ is *symmetric monoidal closed* with respect to smash products, i. e. the smash product is symmetric and the functor $\cdot \otimes C$ has a right adjoint for every object C. This is given by the functor $[C \circ\!\!\to \cdot]$. In other words, the category $DOM_\perp$ satisfies the universal property

$$\begin{tikzcd}
{[C \circ\!\!\to B]} & {[C \circ\!\!\to B] \otimes C} \arrow[r, "\widetilde{\mathsf{eval}}"] & B \\
A \arrow[u, dotted, "\exists!\ \widetilde{\mathsf{curry}}\, f"] & A \otimes C \arrow[u, "\widetilde{\mathsf{curry}}\, f \otimes \mathsf{id}_C"] \arrow[ur, "\forall f"'] &
\end{tikzcd}$$

where $\widetilde{\mathsf{eval}}$ and $\widetilde{\mathsf{curry}}$ are strict variants of the evaluation map and currying. It is easy to verify that all functions involved are continuous and strict. The operator $\circ\!\!\to$, where $f \circ\!\!\to g$ is defined analogously to g^f, is a functor on the subcategory $DOM_\perp$, however, as the coalesced sum and the smash product operator, it is not functorial on the whole category DOM: $\big((c_{\top_\mathbf{2}} \circ\!\!\to \mathsf{id}_\mathbf{2}) \circ (c_{\perp_\mathbf{2}} \circ\!\!\to \mathsf{id}_\mathbf{2})\big)(\mathsf{id}_\mathbf{2}) = (c_{\top_\mathbf{2}} \circ\!\!\to \mathsf{id}_\mathbf{2})(c_{\perp_\mathbf{2}}) = c_{\perp_\mathbf{2}}$, but $\big((c_{\top_\mathbf{2}} \circ c_{\perp_\mathbf{2}}) \circ\!\!\to (\mathsf{id}_\mathbf{2} \circ \mathsf{id}_\mathbf{2})\big)(\mathsf{id}_\mathbf{2}) = (c_{\top_\mathbf{2}} \circ\!\!\to \mathsf{id}_\mathbf{2})(\mathsf{id}_\mathbf{2}) = \mathsf{id}_\mathbf{2}$.

Connections between Finitary Constructions

Note that the correspondences $(A \times B)_\perp \cong A_\perp \otimes B_\perp$, $\sum_{i=1}^n A_i \cong (A_1)_\perp \oplus \cdots \oplus (A_n)_\perp$ and $A \to B \cong A_\perp \circ\!\!\to B$ hold in DOM.

2.4 Recursive Domain Equations

The constructions considered above are all finitary, in particular, they map finite domains to finite ones. However, the category $DOM_\perp$ also provides a nice framework to solve *domain equations* of the form $X \cong F(X, X)$ where $F\colon DOM_\perp^{op} \times DOM_\perp \to DOM_\perp$ is a mixed variant functor. This is essential for interpreting infinitely constructed recursive types. The following theorem goes back to M. Smyth and G. Plotkin, see [SP82]:

PROPOSITION 2.1. *Let $F\colon DOM_\perp^{op} \times DOM_\perp \to DOM_\perp$ be a* locally continuous *functor on $DOM_\perp$ (i. e. it is continuous on the morphism sets). Furthermore, let $e_{n+k,n}\colon F^{(n)}(\mathbf{1},\mathbf{1}) \circ\!\!\to F^{(n+k)}(\mathbf{1},\mathbf{1})$ and $p_{n,n+k}\colon F^{(n+k)}(\mathbf{1},\mathbf{1}) \circ\!\!\to F^{(n)}(\mathbf{1},\mathbf{1})$ be defined by*

$$\begin{aligned}
e_{1,0} &:= \lambda x{:}\mathbf{1}.\perp_{F(\mathbf{1},\mathbf{1})}, & p_{0,1} &:= \lambda x{:}F(\mathbf{1},\mathbf{1}).\perp_\mathbf{1}, \\
e_{n+1,n} &:= F(p_{n-1,n}, e_{n,n-1}), & p_{n,n+1} &:= F(e_{n,n-1}, p_{n-1,n}), \\
e_{n+k,n} &:= e_{n+k,n+k-1} \circ \cdots \circ e_{n+1,n}, & p_{n,n+k} &:= p_{n,n+1} \circ \cdots \circ p_{n+k-1,n+k}
\end{aligned}$$

where

$$F^{(0)}(\mathbf{1},\mathbf{1}) := \mathbf{1} \qquad \textit{and} \qquad F^{(n+1)}(\mathbf{1},\mathbf{1}) := F\big(F^{(n)}(\mathbf{1},\mathbf{1}), F^{(n)}(\mathbf{1},\mathbf{1})\big).$$

Then

$$\mathsf{fix}(F) \;:=\; \left\{ \langle x_0, x_1, x_2, \dots \rangle \in \prod_{n\in\mathbb{N}} F^{(n)}(\mathbf{1},\mathbf{1}) \;\middle|\; p_{n,n+k}(x_{n+k}) = x_n \textit{ for all } k \in \mathbb{N} \right\}$$

is a limit for the diagram of all domains $F^{(n)}(\mathbf{1},\mathbf{1})$ and projections $p_{n,n+k}$ as well as a colimit for the domains $F^{(n)}(\mathbf{1},\mathbf{1})$ with embeddings $e_{n+k,n}$. (Therefore, we call $\mathsf{fix}(F)$ a bilimit.*) The embeddings $e_n\colon F^{(n)}(\mathbf{1},\mathbf{1}) \hookrightarrow \mathsf{fix}(F)$ and projections $p_n\colon \mathsf{fix}(F) \hookrightarrow F^{(n)}(\mathbf{1},\mathbf{1})$ are given by*

$$\big(e_n(x)\big)_m = \begin{cases} p_{m,n}(x) & \textit{if } n \geq m \\ e_{m,n}(x) & \textit{if } n < m \end{cases}, \qquad\qquad p_n\big(\langle x_0, x_1, x_2, \dots \rangle\big) = x_n .$$

Furthermore, $F\big(\mathsf{fix}(F), \mathsf{fix}(F)\big) \;\cong\; \mathsf{fix}(F)$ holds and, for all domains A satisfying $F(A,A) \cong A$, there is a unique embedding projection pair *$e\colon \mathsf{fix}(F) \hookrightarrow A$, $p\colon A \hookrightarrow \mathsf{fix}(F)$ with $p \circ e = \mathsf{id}_{\mathsf{fix}(F)}$ and $e \circ p \sqsubseteq \mathsf{id}_A$. The isomorphisms $\mathsf{fold}\colon F\big(\mathsf{fix}(F), \mathsf{fix}(F)\big) \hookrightarrow \mathsf{fix}(F)$ and $\mathsf{unfold}\colon \mathsf{fix}(F) \hookrightarrow F\big(\mathsf{fix}(F), \mathsf{fix}(F)\big)$ are given by*

$$\mathsf{fold} \;=\; \bigsqcup_{n\in\mathbb{N}}^{\uparrow} e_{n+1} \circ F(e_n, p_n), \qquad\qquad \mathsf{unfold} \;=\; \bigsqcup_{n\in\mathbb{N}}^{\uparrow} F(p_n, e_n) \circ p_{n+1} .$$

The embeddings $e_n\colon F^{(n)}(\mathbf{1},\mathbf{1}) \hookrightarrow \mathsf{fix}(F)$ and projections $p_n\colon \mathsf{fix}(F) \hookrightarrow F^{(n)}(\mathbf{1},\mathbf{1})$ satisfy the equations

$$\begin{aligned} e_0 &\;=\; \lambda x{:}\mathbf{1}.\bot_{\mathsf{fix}(F)}, & \qquad p_0 &\;=\; \lambda x{:}\mathsf{fix}(F).\bot_{\mathbf{1}}, \\ e_{n+1} &\;=\; \mathsf{fold} \circ F(p_n, e_n), & \qquad p_{n+1} &\;=\; F(e_n, p_n) \circ \mathsf{unfold}. \end{aligned}$$

Moreover, the solution $\mathsf{fix}(F)$ of the domain equation $F(A,A) \cong A$ from above is invariant in the following sense. If $\Phi\colon [\mathsf{fix}(F) \hookrightarrow \mathsf{fix}(F)] \to [\mathsf{fix}(F) \hookrightarrow \mathsf{fix}(F)]$ is a functional defined by

$$\Phi(f) \;=\; \mathsf{fold} \circ F(f,f) \circ \mathsf{unfold},$$

then $\mathsf{id}_{\mathsf{fix}(F)} = \bigsqcup^{\uparrow}_{n\in\mathbb{N}} \Phi^n(\bot_{[\mathsf{fix}(F) \hookrightarrow \mathsf{fix}(F)]})$ holds.

Proof. We only give a sketch of the proof. It is not difficult to check that $\mathsf{fix}(F)$ as defined in the proposition is a limit for the indirect sequence of the domains $F^{(n)}(\mathbf{1},\mathbf{1})$ with the projections $p_{n,n+k}$. It is also easy to verify by induction that every pair $(e_{n+1,n}, p_{n,n+1})$ is an embedding projection pair, i. e. $p_{n,n+1} \circ e_{n+1,n} = \mathsf{id}_{F^{(n)}(\mathbf{1},\mathbf{1})}$ and $e_{n+1,n} \circ p_{n,n+1} \sqsubseteq \mathsf{id}_{F^{(n+1)}(\mathbf{1},\mathbf{1})}$ hold. A straightforward calculation shows that $\mathsf{fix}(F)$ with the morphisms $e_n\colon F^{(n)}(\mathbf{1},\mathbf{1}) \hookrightarrow \mathsf{fix}(F)$ as defined in the proposition is a colimit for the expanding sequence

$$\mathbf{1} \xrightarrow{e_{1,0}} F(\mathbf{1},\mathbf{1}) \xrightarrow{e_{2,1}} F^{(2)}(\mathbf{1},\mathbf{1}) \xrightarrow{e_{3,2}} \cdots$$

and that the morphisms e_n and p_n form embedding projections pairs themselves, see e. g. [AJ94]. Besides, we get:

$$\bigsqcup_{n\in\mathbb{N}}^{\uparrow} e_n \circ p_n = \mathsf{id}_{\mathsf{fix}(F)} \qquad (\star)$$

Due to the definition of the morphisms $e_{n+k,n}$ and $p_{n,n+k}$, the domain $F\big(\mathsf{fix}(F), \mathsf{fix}(F)\big)$ is also a limit for the diagram

$$\mathbf{1} \xleftarrow{p_{0,1}} F(\mathbf{1},\mathbf{1}) \xleftarrow{p_{1,2}} F^{(2)}(\mathbf{1},\mathbf{1}) \xleftarrow{p_{2,3}} \cdots$$

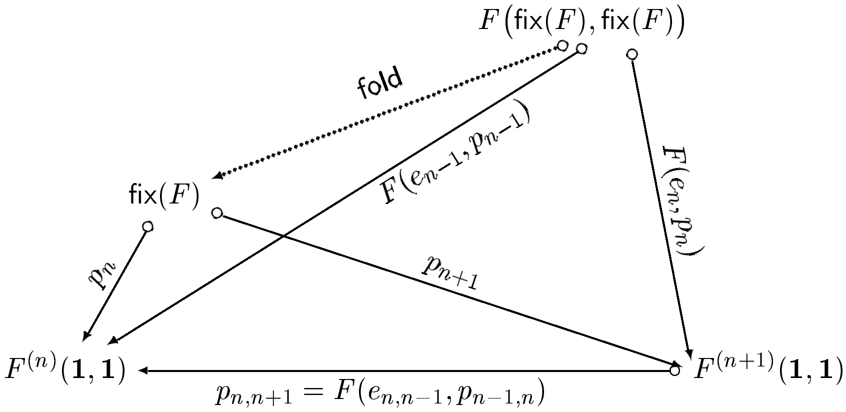
and a colimit for the diagram

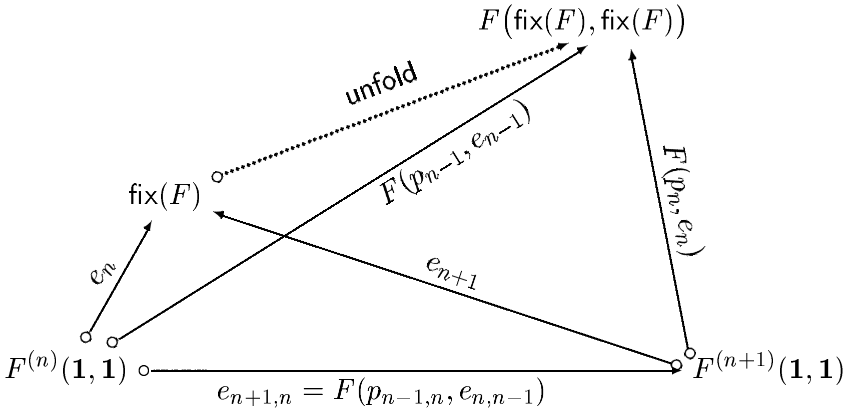
$$\mathbf{1} \xrightarrow{e_{1,0}} F(\mathbf{1},\mathbf{1}) \xrightarrow{e_{2,1}} F^{(2)}(\mathbf{1},\mathbf{1}) \xrightarrow{e_{3,2}} \cdots,$$

this means:

$$p_{n,n+1} \circ F(e_n, p_n) = F(e_{n-1}, p_{n-1}), \qquad F(p_n, e_n) \circ e_{n+1,n} = F(p_{n-1}, e_{n-1}).$$

Because of the universal property of $\mathsf{fix}(F)$, there are uniquely determined morphisms $\mathsf{fold}\colon F\big(\mathsf{fix}(F), \mathsf{fix}(F)\big) \multimap \mathsf{fix}(F)$ and $\mathsf{unfold}\colon \mathsf{fix}(F) \multimap F\big(\mathsf{fix}(F), \mathsf{fix}(F)\big)$ making the following diagrams commute:





Thus, we have

$$p_{n+1} \circ \mathsf{fold} = F(e_n, p_n), \qquad \mathsf{unfold} \circ e_{n+1} = F(p_n, e_n) \qquad (\star\star)$$

for every $n \in \mathbb{N}$. Using $(\star)$ leads to

$$\mathsf{fold} = \bigsqcup_{n\in\mathbb{N}}^{\uparrow} e_{n+1} \circ p_{n+1} \circ \mathsf{fold} = \bigsqcup_{n\in\mathbb{N}}^{\uparrow} e_{n+1} \circ F(e_n, p_n)$$

as well as

$$\mathsf{unfold} \;=\; \mathsf{unfold} \circ \bigsqcup_{n\in\mathbb{N}}^{\uparrow} e_{n+1} \circ p_{n+1} \;=\; \bigsqcup_{n\in\mathbb{N}}^{\uparrow} \mathsf{unfold} \circ e_{n+1} \circ p_{n+1} \;=\; \bigsqcup_{n\in\mathbb{N}}^{\uparrow} F(p_n, e_n) \circ p_{n+1}.$$

As the functor F is assumed to be locally continuous, we get

$$\begin{aligned}
\mathsf{unfold} \circ \mathsf{fold} &= \bigsqcup_{n\in\mathbb{N}}^{\uparrow} F(p_n, e_n) \circ p_{n+1} \circ e_{n+1} \circ F(e_n, p_n) \\
&= \bigsqcup_{n\in\mathbb{N}}^{\uparrow} F(e_n \circ p_n, e_n \circ p_n) \\
&= F\left(\bigsqcup_{n\in\mathbb{N}}^{\uparrow} e_n \circ p_n, \bigsqcup_{n\in\mathbb{N}}^{\uparrow} e_n \circ p_n\right) \\
&= F(\mathsf{id}_{\mathsf{fix}(F)}, \mathsf{id}_{\mathsf{fix}(F)}) \\
&= \mathsf{id}_{F(\mathsf{fix}(F), \mathsf{fix}(F))}.
\end{aligned}$$

It is easy to verify that $\mathsf{fold} \circ \mathsf{unfold} = \mathsf{id}_{\mathsf{fix}(F)}$ holds as well. Hence, $\mathsf{fix}(F)$ and $F\big(\mathsf{fix}(F), \mathsf{fix}(F)\big)$ are isomorphic.

Another consequence of $(\star\star)$ is

$$e_{n+1} \;=\; \mathsf{fold} \circ F(p_n, e_n), \qquad\qquad p_{n+1} \;=\; F(e_n, p_n) \circ \mathsf{unfold}$$

as required in the proposition, the equations $e_0 := \lambda x{:}\mathbf{1}.\bot_{\mathsf{fix}(F)}$ and $p_0 := \lambda x{:}\mathsf{fix}(F).\bot_{\mathbf{1}}$ hold trivially. This yields

$$\begin{aligned}
e_{n+1} \circ p_{n+1} &= \mathsf{fold} \circ F(p_n, e_n) \circ F(e_n, p_n) \circ \mathsf{unfold} \\
&= \mathsf{fold} \circ F(e_n \circ p_n, e_n \circ p_n) \circ \mathsf{unfold} \\
&= \Phi(e_n \circ p_n)
\end{aligned}$$

for all natural numbers $n \in \mathbb{N}$ which implies $e_{n+1} \circ p_{n+1} = \Phi^n(e_0 \circ p_0) = \Phi^n(\bot)$. Finally, $\mathsf{id}_{\mathsf{fix}(F)} = \bigsqcup^{\uparrow}_{n\in\mathbb{N}} e_{n+1} \circ p_{n+1} = \bigsqcup^{\uparrow}_{n\in\mathbb{N}} \Phi^n(\bot)$ follows. ☐

As shown by P. Freyd in [Fre91], the fixed point as constructed in Proposition 2.1 is initial in the following sense:

PROPOSITION 2.2. *Let $F\colon DOM_\bot^{op} \times DOM_\bot \to DOM_\bot$ be a locally continuous functor and let $D := \mathsf{fix}(F)$ be its canonical fixed point as constructed in Proposition 2.1. For every pair of domains A, B and every pair of morphisms $f\colon A \multimap F(B, A)$ and $g\colon F(A, B) \multimap B$ in $DOM_\bot$ there are unique morphisms $h\colon A \multimap D$ and $k\colon D \multimap B$ making the following diagrams commute:*

$$\begin{tikzcd}
D \arrow[r, "\mathsf{unfold}"] & F(D,D) \\
A \arrow[u, "h"] \arrow[r, "f"'] & F(B,A) \arrow[u, "{F(k,h)}"']
\end{tikzcd}
\quad \textit{and} \quad
\begin{tikzcd}
F(D,D) \arrow[r, "\mathsf{fold}"] \arrow[d, "{F(h,k)}"'] & D \arrow[d, "k"] \\
F(A,B) \arrow[r, "g"'] & B
\end{tikzcd}$$

Proof. Since we are considering locally continuous functors rather than arbitrary ones as in [Fre91], we can give a proof that is more elementary than the original one. It goes back to A. Pitts, see [Pit96].

The morphisms $h\colon A \circ\!\!\rightarrow D$ and $k\colon D \circ\!\!\rightarrow B$ are given by the fixed point of the function $\varphi\colon ([A \circ\!\!\rightarrow D] \times [D \circ\!\!\rightarrow B]) \rightarrow ([A \circ\!\!\rightarrow D] \times [D \circ\!\!\rightarrow B])$ with

$$\varphi(h,k) \;=\; \big\langle \mathsf{fold} \circ F(k,h) \circ f, g \circ F(h,k) \circ \mathsf{unfold} \big\rangle$$

(this function is continuous since F is locally continuous). For proving the uniqueness of h and k we need the following observation, which is often referred to as *Plotkin's Axiom*:

> Let A, B be domains. If $d\colon A \circ\!\!\rightarrow B$ is a strict continuous function and $a\colon A \rightarrow A$ and $b\colon B \rightarrow B$ are continuous functions with $b \circ d = d \circ a$, then $\mathsf{fix}(b) = d\big(\mathsf{fix}(a)\big)$ holds.

This result is not difficult to prove: $\mathsf{fix}(b) = \bigsqcup^{\uparrow}{}_{n\in\mathbb{N}}\, b^n(\bot) = \bigsqcup^{\uparrow}{}_{n\in\mathbb{N}}\, b^n \circ d(\bot) = \bigsqcup^{\uparrow}{}_{n\in\mathbb{N}}\, d \circ a^n(\bot) = d\big(\bigsqcup^{\uparrow}{}_{n\in\mathbb{N}}\, a^n(\bot)\big) = d\big(\mathsf{fix}(a)\big)$. Now suppose $h'\colon A \circ\!\!\rightarrow D$ and $k'\colon D \circ\!\!\rightarrow B$ are two other morphisms such that the (appropriately modified) diagrams in the proposition commute. Consider the function $d\colon [D \circ\!\!\rightarrow D] \rightarrow ([A \circ\!\!\rightarrow D] \times [D \circ\!\!\rightarrow B])$ defined by $d(e) = \langle e \circ h', k' \circ e\rangle$ which is continuous and strict. We show that d also makes the following diagram commute:

$$\begin{tikzcd}
{[D \circ\!\!\rightarrow D]} \arrow[r, "d"] \arrow[d, "\Phi"'] & {[A \circ\!\!\rightarrow D] \times [D \circ\!\!\rightarrow B]} \arrow[d, "\varphi"] \\
{[D \circ\!\!\rightarrow D]} \arrow[r, "d"'] & {[A \circ\!\!\rightarrow D] \times [D \circ\!\!\rightarrow B]}
\end{tikzcd}$$

For all $e \in [D \circ\!\!\rightarrow D]$ we get

$$\begin{aligned}
(\varphi \circ d)(e) &= \varphi\langle e \circ h', k' \circ e\rangle \\
&= \big\langle \mathsf{fold} \circ F(k' \circ e, e \circ h') \circ f, g \circ F(e \circ h', k' \circ e) \circ \mathsf{unfold}\big\rangle \\
&= \big\langle \mathsf{fold} \circ F(e,e) \circ F(k',h') \circ f, g \circ F(h',k') \circ F(e,e) \circ \mathsf{unfold}\big\rangle \\
&= \big\langle \mathsf{fold} \circ F(e,e) \circ \mathsf{unfold} \circ h', k' \circ \mathsf{fold} \circ F(e,e) \circ \mathsf{unfold}\big\rangle \\
&= \big\langle \Phi(e) \circ h', k' \circ \Phi(e)\big\rangle \\
&= d\big(\Phi(e)\big) \\
&= d \circ \Phi(e),
\end{aligned}$$

which proves $\varphi \circ d = d \circ \Phi$. Plotkin's Axiom yields $\mathsf{fix}(\varphi) = d\big(\mathsf{fix}(\Phi)\big)$. Since Proposition 2.1 implies $\mathsf{fix}(\Phi) = \mathsf{id}_D$, we finally obtain $\langle h,k\rangle = \mathsf{fix}(\varphi) = d\big(\mathsf{fix}(\Phi)\big) = d(\mathsf{id}_D) = \langle h',k'\rangle$. $\square$

In order to compute fixed points of functors with parameters, we need to generalise the previous result. Such functors will occur later when we study the denotation of recursive types like $\mu\alpha.\mu\beta.\tau(\alpha,\beta)$. A proof sketch can be found in [Pit96].

PROPOSITION 2.3. *Let* $G\colon DOM_\perp^{op} \times DOM_\perp \times DOM_\perp^{op} \times DOM_\perp \to DOM_\perp$ *be a bivariant locally continuous functor. Then there is another locally continuous functor* $F\colon DOM_\perp^{op} \times DOM_\perp \to DOM_\perp$ *such that*

$$G\big(D^-, D^+, F(D^+, D^-), F(D^-, D^+)\big) \;\cong\; F(D^-, D^+)$$

holds for every two objects D^- *and* D^+ *in* DOM. *The isomorphisms are given by two functions* fold_{D^-,D^+} *and* $\mathsf{unfold}_{D^-,D^+}$. *(The indexes are often omitted when they are clear from the context.) Besides, the domains* $F(D^+, D^-)$ *and* $F(D^-, D^+)$ *satisfy the following property: the pair* $(\mathsf{id}_{F(D^+,D^-)}, \mathsf{id}_{F(D^-,D^+)})$ *is the least fixed point of the function* Φ *on* $\big(F(D^+, D^-) \circ\!\!\to F(D^+, D^-)\big) \times \big(F(D^-, D^+) \circ\!\!\to F(D^-, D^+)\big)$ *given by* $\Phi(e^-, e^+) = \big(\Phi^-(e^-, e^+), \Phi^+(e^-, e^+)\big)$ *with*

$$\begin{aligned}
\Phi^-(e^-, e^+) &:= \mathsf{fold}_{D^+,D^-} \circ G(\mathsf{id}_{D^+}, \mathsf{id}_{D^-}, e^+, e^-) \circ \mathsf{unfold}_{D^+,D^-},\\
\Phi^+(e^-, e^+) &:= \mathsf{fold}_{D^-,D^+} \circ G(\mathsf{id}_{D^-}, \mathsf{id}_{D^+}, e^-, e^+) \circ \mathsf{unfold}_{D^-,D^+}.
\end{aligned}$$

Moreover, for all morphisms $f\colon A \circ\!\!\to G(D^-, D^+, B, A)$ *and* $g\colon G(D^+, D^-, A, B) \circ\!\!\to B$, *there are unique morphisms* $h\colon A \circ\!\!\to F(D^-, D^+)$ *and* $k\colon F(D^+, D^-) \circ\!\!\to B$ *making the following diagrams commute:*

$$\begin{tikzcd}
F(D^-, D^+) \arrow[r, "\mathsf{unfold}"] & G\big(D^-, D^+, F(D^+, D^-), F(D^-, D^+)\big) \\
A \arrow[u, "h"] \arrow[r, "f"'] & G(D^-, D^+, B, A) \arrow[u, "{G(\mathsf{id}, \mathsf{id}, k, h)}"']
\end{tikzcd}$$

and

$$\begin{tikzcd}
G\big(D^+, D^-, F(D^-, D^+), F(D^+, D^-)\big) \arrow[r, "\mathsf{fold}"] \arrow[d, "{G(\mathsf{id}, \mathsf{id}, h, k)}"'] & F(D^+, D^-) \arrow[d, "k"] \\
G(D^+, D^-, A, B) \arrow[r, "g"'] & B
\end{tikzcd}$$

□

The action on morphisms of the functor $F\colon DOM_\perp^{op} \times DOM_\perp \to DOM_\perp$ can also be obtained from Proposition 2.3. Given morphisms $u^-\colon D_2^- \circ\!\!\to D_1^-$ and $u^+\colon D_1^+ \circ\!\!\to D_2^+$, we obtain $F(u^-, u^+)$ and $F(u^+, u^-)$ as h and k by setting $D^- = D_2^-$, $D^+ = D_2^+$, $A = F(D_1^-, D_1^+)$, $B = F(D_1^+, D_1^-)$, $f = G(u^-, u^+, \mathsf{id}, \mathsf{id}) \circ \mathsf{unfold}$ and $g = \mathsf{fold} \circ G(u^+, u^-, \mathsf{id}, \mathsf{id})$.

Relational Structures

For the proof of *computational adequacy* (Proposition 5.6 and Theorem 5.7), we will use the concept of *relational structures*. This idea goes back to P. O'Hearn and R. Tennent [OT93] and was modified by A. Pitts [Pit96]:

DEFINITION 2.4. *A* relational structure $\mathcal{R}$ *on a category* $\mathcal{C}$ *consists of a set* $\mathcal{R}(A)$ *(called* $\mathcal{R}$*-relation on* A*), for each object* A *in* $\mathcal{C}$*, and a binary relation* $f\colon R \subset_{\mathcal{R}} S$ *(or just* $f\colon R \subset S$*) for each morphism* $f\colon A \to B$ *in* $\mathcal{C}$ *and every two* $\mathcal{R}$*-relations* $R \in \mathcal{R}(A)$ *and* $S \in \mathcal{R}(B)$*, satisfying the following two properties:*

- $\mathsf{id}_A\colon R \subset R$ *holds for all* $R \in \mathcal{R}(A)$*,*
- *if* f *and* g *are composable, then* $f\colon R \subset S$ *and* $g\colon S \subset T$ *imply* $g \circ f\colon R \subset T$*.*

A relation $R \in \mathcal{R}(A)$ *is called* ω-admissible *if, for all other* $\mathcal{R}$*-relations* $S \in \mathcal{R}(B)$*, the set* $\{f\colon A \hookrightarrow B \mid f\colon R \subset S\}$ *contains* $\perp$ *and is closed under suprema of* ω*-chains. The subset of all* ω*-admissible* $\mathcal{R}$*-relations in* $\mathcal{R}(A)$ *is denoted by* $\mathcal{R}_{adm}(A)$*.*

Proposition 2.3 can now be extended to relational structures. For the proof, see again [Pit96].

PROPOSITION 2.5. *Let* $G\colon DOM_\perp^{op} \times DOM_\perp \times DOM_\perp^{op} \times DOM_\perp \to DOM_\perp$ *and* $F\colon DOM_\perp^{op} \times DOM_\perp \to DOM_\perp$ *be locally continuous functors with*

$$G\big(D^-, D^+, F(D^+, D^-), F(D^-, D^+)\big) \;\cong\; F(D^-, D^+)$$

for all objects D^- *and* D^+ *in* $DOM_\perp$*. Besides, let* $\mathcal{R}$ *be a relational structure on* $DOM_\perp$ *and* $F(R^-, R^+) \in \mathcal{R}\big(F(D^-, D^+)\big)$ *with* $R^- \in D^-$*,* $R^+ \in D^+$ *be a family of* ω*-admissible* $\mathcal{R}$*-relations with*

$$\mathsf{fold}\colon G\big(R^-, R^+, F(R^+, R^-), F(R^-, R^+)\big) \subset F(R^-, R^+)$$

and

$$\mathsf{unfold}\colon F(R^-, R^+) \subset G\big(G(R^-, R^+, F(R^+, R^-), F(R^-, R^+)\big).$$

If

$$f\colon P \subset G(R^-, R^+, Q, P) \quad \textit{and} \quad g\colon G(R^+, R^-, P, Q) \subset Q$$

hold for all $f\colon A \hookrightarrow G(D^-, D^+, B, A)$*,* $g\colon G(D^+, D^-, A, B) \hookrightarrow B$*,* $R^- \in \mathcal{R}(D^-)$*,* $R^+ \in \mathcal{R}(D^+)$*,* $P \in \mathcal{R}(A)$ *and* $Q \in \mathcal{R}_{adm}(B)$*, then the uniquely determined morphisms* $h\colon A \hookrightarrow F(D^-, D^+)$ *and* $k\colon F(D^-, D^+) \hookrightarrow B$ *satisfy*

$$h\colon P \subset F(R^-, R^+) \quad \textit{and} \quad k\colon F(R^+, R^-) \subset Q.$$

□

2.5 Subcategories of Domains

Algebraic Domains

Most domains used in Denotational Semantics are (ω)-*algebraic* which means that every element is uniquely determined by (countably many) in some sense finite approximations. More precisely:

DEFINITION 2.6. *Let D be a domain. An element $c \in D$ is called* compact *if it satisfies the property*

$$c \sqsubseteq \bigsqcup\nolimits^{\uparrow} P \quad \Longrightarrow \quad \exists p \in P.\ c \sqsubseteq p \qquad \text{for all directed subsets } P \subseteq D.$$

The set of all compact elements of D is denoted by $K(D)$. Moreover, D is called algebraic *if, for every element $d \in D$, the set $K(d) := \{c \in K(D) \mid c \sqsubseteq d\}$ is directed and $d = \bigsqcup^{\uparrow} K(d)$ holds. We call D* ω-algebraic *if it is algebraic and the set of all compact elements of D is countable.*

SFP-Domains

Unfortunately, the subcategory of all algebraic domains is not closed under exponentiation, i.e. the function space between two algebraic domains does not need to be algebraic. However, there is a restricted class of domains considered in denotational semantics that is closed under exponentiation. Its objects are defined in the following way:

DEFINITION 2.7. *A domain D is called* SFP-domain *(for sequence of finite projections) if there is an increasing sequence of projections $\pi_n \colon D \to D$ (i.e. $\pi_n \circ \pi_n = \pi_n \sqsubseteq \mathsf{id}_D$ holds for all $n \in \mathbb{N}$) with finite image such that $\mathsf{id}_D = \bigsqcup^{\uparrow}{}_{n \in \mathbb{N}} \pi_n$ holds.*

It is not too difficult to verify that SFP-domains are stable under all previously defined operators including the bilimit construction. For example, if π_n^A and π_n^B $(n \in \mathbb{N})$ are the projections of the increasing sequences for SFP-domains A and B, then a sequence $\left(\pi_n^{[A \to B]}\right)_{n \in \mathbb{N}}$ witnessing the SFP-property is given by

$$\pi_n^{[A \to B]} \ := \ \lambda f{:}[A \to B].\lambda a{:}A.\pi_n^B \circ f \circ \pi_n^A(a).$$

The proof for the other operators is similar.

The class of all SFP-domains is indeed a subclass of algebraic domains:

LEMMA 2.8. *Every SFP-domain is ω-algebraic.*

Proof. Let d be an element in an SFL-domain D and $(\pi_n)_{n \in \mathbb{N}}$ an increasing sequence of projections with finite image. We show that every element $\pi_n(d)$ (for all $n \in \mathbb{N}$) is compact. If P is a directed set with $\pi_n(d) \sqsubseteq \bigsqcup^{\uparrow} P$, then $\pi_n(d) = \pi_n\big(\pi_n(d)\big) \sqsubseteq \pi_n(\bigsqcup^{\uparrow} P) = \bigsqcup^{\uparrow} \pi_n(P)$ follows from the continuity of π_n. Since $\pi_n(P)$ is finite, there is an $p \in P$ with $\bigsqcup^{\uparrow} \pi_n(P) = \pi_n(p) \sqsubseteq p$ and, hence, $\pi_n(d)$ is compact. On the other hand, because of $\mathsf{id}_D = \bigsqcup^{\uparrow}{}_{n \in \mathbb{N}} \pi_n$, every compact element in D is of the form $\pi_n(d)$ for some $d \in D$ and $n \in \mathbb{N}$. Furthermore, the set $K(D) = \{\pi_n(d) \mid d \in D,\ n \in \mathbb{N}\}$ is countable as it is a union of finite sets. It is easy to prove that, for all $d \in D$, the set $K(d)$ is directed, thus, the claim follows from $\mathsf{id}_D = \bigsqcup^{\uparrow}{}_{n \in \mathbb{N}} \pi_n$. □

The category of SFP-domains plays a special role in the theory of (algebraic) domains. As shown by M. Smyth, it is a maximal cartesian closed full subcategory of the category of algebraic domains [Smy83].

Scott Domains

As a motivation for the next class of domains consider two programs P_1 and P_2 of type **nat** $\to$ **nat** that are incomparable, but compatible, e.g. imagine P_1 returns output only for input $1, \dots, 10$, P_2 only for $5, \dots, 15$ and their results agree on the numbers $5, \dots, 10$. Both programs can be considered as approximations to a partial function $f\colon \mathbb{N}_\perp \to \mathbb{N}_\perp$ with domain $\{1, \dots, 15\}$. In semantics, it is often assumed that under these circumstances there is a least expressive program P approximating f and subsuming P_1 and P_2, which, in this case, computes the correct result for input $1, \dots, 15$ and diverges otherwise.

DEFINITION 2.9. *A subset M of a partially ordered set P is called* bounded *if it has an upper bound in P, i. e. there is an element $p \in P$ with $m \sqsubseteq p$ for all $m \in M$. A domain D is called* bounded complete *if each bounded subset M has a supremum $\bigsqcup M$ in D. An ω-algebraic bounded complete domain is called* Scott domain.

In order to verify that a given domain D is bounded complete, it is not necessary to check whether all bounded suprema exist:

PROPOSITION 2.10. *Let D be a domain, then the following are equivalent:*

1. *D is bounded complete.*
2. *Every two bounded elements in D have a supremum.*
3. *Every nonempty subset in D has an infimum.*

Proof. (1) $\Rightarrow$ (3): Let D be a bounded complete domain and P be a nonempty subset of D. The set of all lower bounds of P is trivially bounded by the elements of P, therefore, it has a supremum. This is certainly the greatest lower bound of P.

(3) $\Rightarrow$ (2): Let D be closed under nonempty infima and $\{d, e\} \in D$ a bounded subset. The set $\mathsf{ub}\{d, e\}$ of all upper bounds of d and e is nonempty and, therefore, has an infimum. This is the supremum $d \sqcup e$.

(2) $\Rightarrow$ (1): If P is a bounded subset of D, then the existence of all binary bounded suprema in D yields the existence of suprema of all finite subsets of P. The set of all these finite suprema is directed, hence, it has a supremum, too. This is the supremum of P. $\square$

There is a nice characterisation for Scott domains:

PROPOSITION 2.11. *Let D be a domain, then the following are equivalent:*

1. *D is a Scott domain.*
2. *D is a bounded complete SFP-domain.*
3. *D is an SFP-domain and has binary infima.*

Proof. (1) ⇒ (2): Let D be a Scott domain, we need to construct an increasing sequence of finite projections $\pi_n \colon D \to D$ ($n \in \mathbb{N}$) with $\mathsf{id}_D := \bigsqcup^{\uparrow}_{n\in\mathbb{N}} \pi_n$. Since D is a ω-algebraic by Lemma 2.8, there is a countable set $K(D) = \{c_1, c_2, \dots\}$ of compact elements generating every element $d \in D$. Let $(\pi_n)_{n\in\mathbb{N}}$ be defined by

$$\pi_n(d) \;:=\; \bigsqcup\{c_i \in K(D) \mid c_i \sqsubseteq d,\; i \leq n\}.$$

This supremum exists as, for all $d \in D$, the set of all smaller compact elements is bounded by d. It is also finite since every set $\pi_n(D)$ is finite. As D is algebraic, $\bigsqcup^{\uparrow}_{n\in\mathbb{N}} \pi_n(d) = \bigsqcup^{\uparrow}_{n\in\mathbb{N}} \bigsqcup\{c_i \in K(D) \mid c_i \sqsubseteq d, i \leq n\} = \bigsqcup\{c \in K(D) \mid c \sqsubseteq d\} = d$ follows, which implies $\bigsqcup^{\uparrow}_{n\in\mathbb{N}} \pi_n = \mathsf{id}_D$.

(2) ⇒ (3): This follows directly from Proposition 2.10.

(3) ⇒ (1): Let D be an SFP-domain with binary infima. Due to Lemma 2.8 and Proposition 2.10, it remains to verify that D is closed under nonempty infima. Suppose $P \neq \varnothing$ is a subset of D. As D is an SFP-domain, there is an increasing sequence of finite projections $\pi_n \colon D \to D$ with $\bigsqcup^{\uparrow}_{n\in\mathbb{N}} \pi_n = \mathsf{id}_D$. For all $n \in \mathbb{N}$, the set $\pi_n(P)$ is finite, hence its infimum $\bigsqcap \pi_n(P)$ exists by assumption. We prove that $\bigsqcup^{\uparrow}_{n\in\mathbb{N}} \big(\bigsqcap \pi_n(P)\big)$ is the infimum of P. For all $p \in P$ and $n \in \mathbb{N}$, $\bigsqcap \pi_n(P) \sqsubseteq \pi_n(p) \sqsubseteq p$ holds which implies that $\bigsqcup^{\uparrow} \big(\bigsqcap \pi_n(P)\big)$ is a lower bound of P. If $d \in D$ is below all elements of P, then $\pi_n(d) \sqsubseteq \pi_n(p)$ is true for all $p \in P$ and $n \in \mathbb{N}$, thus, $\pi_n(d) \sqsubseteq \bigsqcap \pi_n(P)$ follows. This implies $d = \bigsqcup^{\uparrow}_{n\in\mathbb{N}} \pi_n(d) \sqsubseteq \bigsqcup^{\uparrow}_{n\in\mathbb{N}} \big(\bigsqcap \pi_n(P)\big)$, hence, $\bigsqcup^{\uparrow}_{n\in\mathbb{N}} \big(\bigsqcap \pi_n(P)\big)$ is the infimum of P. □

Thus, every two elements in a Scott domain have an infimum. This is computed continuously:

LEMMA 2.12. *If D is Scott domain, then the infimum operation $\sqcap \colon D \times D \to D$ is continuous.*

Proof. It is clear that the operation $\sqcap$ is monotone. We show

$$d \sqcap \bigsqcup^{\uparrow} P \;=\; \bigsqcup^{\uparrow}(d \sqcup P) \;:=\; \bigsqcup^{\uparrow}\{d \sqcap p \mid p \in P\}$$

for every directed subset $P \in D$ and all elements $d \in P$. One inequality is easily proved: from the monotonicity of $\sqcap$, it follows that $d \sqcap \bigsqcup^{\uparrow} P$ is an upper bound of all elements $d \sqcap p$ with $p \in P$, this entails $d \sqcap \bigsqcup^{\uparrow} P \sqsupseteq \bigsqcup^{\uparrow}(d \sqcup P)$.

By Lemma 2.8, the domain D is algebraic, hence, in order to prove $d \sqcap \bigsqcup^{\uparrow} P \sqsubseteq \bigsqcup^{\uparrow}(d \sqcup P)$, it suffices to show $c \sqsubseteq \bigsqcup^{\uparrow}(d \sqcap P)$ for all compact elements $c \in D$ with $c \sqsubseteq d \sqcap \bigsqcup^{\uparrow} P$. Given such a compact element $c \sqsubseteq d \sqcap \bigsqcup^{\uparrow} P$, then $c \sqsubseteq d$ and $c \sqsubseteq \bigsqcup^{\uparrow} P$ holds. Since c is compact, there is an element $p_0 \in P$ with $c \sqsubseteq p_0$. This implies $c \sqsubseteq d \sqcap p_0 \sqsubseteq \bigsqcup^{\uparrow}(d \sqcap P)$.

As the infimum operation $\sqcap \colon D \times D \to D$ is commutative, it is and continuous in both argument and therefore continuous. □

We have already seen that the subcategory of SFP-domains is closed under the above finitary and recursive constructions. This is also true for the category of Scott domains:

COROLLARY 2.13. *The subcategory of Scott domains is closed under cartesian products, smash products, liftings, coalesced sums, smash sums, exponentials, strict function spaces and bilimits.*

Proof. It is straightforward to verify that the category of bounded complete domains is closed under the above finitary and recursive constructions. Due to Proposition 2.11, the claim follows from the above observation that this also holds for the category of SFP-domains. $\square$

2.6 Monads and Algebras

The last part of this section introduces some more categorical concepts.

Natural Transformations

Let $F, G\colon \mathcal{C} \to \mathcal{D}$ be functors. A *natural transformation* $\eta\colon F \dot{\to} G$ is collection of morphisms $\eta_C\colon FC \to GC$ in $\mathcal{D}$ (for all $C \in \mathcal{C}$) such that, for all morphisms $f\colon C \to C'$ in $\mathcal{C}$, the following diagram commutes:

$$
\begin{tikzcd}
FC \arrow[r, "\eta_C"] \arrow[d, "Ff"'] & GC \arrow[d, "Gf"] \\
FC' \arrow[r, "\eta_{C'}"'] & GC'
\end{tikzcd}
$$

Monads

A monad $T = \langle T, \eta, \mu \rangle$ in a category $\mathcal{C}$ is a functor $T\colon \mathcal{C} \to \mathcal{C}$ together with two natural transformations

$$\eta\colon \mathsf{Id}_{\mathcal{C}} \dot{\to} T, \qquad \mu\colon T^2 \dot{\to} T$$

which make the following diagrams commute:

$$
\begin{tikzcd}
T^3 \arrow[r, "T\mu"] \arrow[d, "\mu T"'] & T^2 \arrow[d, "\mu"] \\
T^2 \arrow[r, "\mu"'] & T
\end{tikzcd}
\qquad
\begin{tikzcd}
\mathsf{Id}\,T \arrow[r, "\eta T"] \arrow[dr, equal] & T^2 \arrow[d, "\mu"] & T\mathsf{Id} \arrow[l, "T\eta"'] \arrow[dl, equal] \\
 & T &
\end{tikzcd}
$$

Here, $T\mu\colon T^3 \dot{\to} T^2$ stands for the natural transformation with components $(T\mu)_A = T(\mu_A)\colon T^3A \to T^2A$ while $\mu T\colon T^3 \dot{\to} T^2$ has components $(\mu T)_A = \mu_{TA}$ (analogously for $T\eta$ and ηT, respectively).

For instance, the lifting functor $\cdot_\perp\colon \mathit{DOM} \to \mathit{DOM}$ with the two natural transformations $\mathsf{up} = (\mathsf{up}_A)_{A\in\mathsf{Obj}(\mathit{DOM})}$ and $(\mathsf{down}_{A_\perp})_{A\in\mathsf{Obj}(\mathit{DOM})}$ is a monad in DOM. Another example is the strict lifting monad on $\mathit{DOM}_\perp$ or $\mathit{DOM}_{\mathit{tot}}$ which is more important for the contents of this thesis. It is equivalent to the lifting monad on the category of *predomains* (i. e. directed complete partially ordered sets, not necessarily having a least element) with partial continuous functions as morphisms. The embedding η is given by the strictification $\mathsf{str}(\mathsf{up})$ of up while the second transformation has $(\mathsf{down}_A)_\perp\colon (A_\perp)_\perp \mathrel{\circ\!\!\rightarrow} A_\perp$ (which is different from $\mathsf{down}_{A_\perp}$) as components.

Algebras

Let $T = \langle T, \eta, \mu\rangle$ be a monad in a category $\mathcal{C}$ and A be an object in $\mathcal{C}$. We call a function $a\colon TA \to A$ a T-*algebra* or *algebra* if it makes the following diagrams commute:

$$\begin{tikzcd}
A \arrow[r, "\eta_A"] \arrow[dr, equal] & TA \arrow[d, "a"] \\
 & A
\end{tikzcd}
\qquad
\begin{tikzcd}
T^2A \arrow[r, "Ta"] \arrow[d, "\mu_A"'] & TA \arrow[d, "a"] \\
TA \arrow[r, "a"'] & A
\end{tikzcd}$$

The aim of this paragraph is to reconstruct the category $\mathit{DOM}_\perp$ as a category of algebras in $\mathit{DOM}_\perp$. Consider the strict lifting monad $\big\langle \cdot_\perp, \mathsf{str}(\mathsf{up}), \big((\mathsf{down}_A)_\perp\big)_{A\in\mathsf{Obj}(\mathit{DOM}_\perp)}\big\rangle$, we firstly examine what the algebras with respect to this monad look like. The first condition in the definition of algebras yields $a(\mathsf{up}\,x) = x$ for all $x \in A \setminus \{\perp_A\}$. The element $\mathsf{up}\perp_A$ is either mapped to a least element $*_A$ in $A\setminus\{\perp_A\}$ or to $\perp_A$. Hence, for the category $\mathit{DOM}_{\mathit{tot}}$ with total functions the former must be the case. This implies that every codomain A of a total algebra $a\colon A_\perp \mathrel{\circ\!\!\rightarrow}_t A$ must have a unique atom $*_A$. This being the case, the validity of the second condition in the definition of algebras is already guaranteed. On the other hand, whenever a domain A has a unique atom, then $a\colon A_\perp \mathrel{\circ\!\!\rightarrow}_t A$ as just described is indeed an algebra. This characterises algebras in the category $\mathit{DOM}_{\mathit{tot}}$.

If T is a monad in a category $\mathcal{C}$, then the collection of all T-algebras forms a category $\mathcal{C}^T$, the *Eilenberg-Moore category* of $\mathcal{C}$. The morphisms are commuting diagrams of the form

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

for all objects A, B in the original category $\mathcal{C}$.

Let us have a closer look at the category $(\mathit{DOM}_{\mathit{tot}})^\perp$. We have already seen that

$A \setminus \{\perp_A\}$ has a least element $*_A$ for every algebra $a\colon A_\perp \circ\!\!\rightarrow_t A$. A morphism

$$
\begin{tikzcd}
A_\perp \arrow[r, "a", "t"' near end] \arrow[d, "f_\perp"', "t" near end] & A \arrow[d, "f", "t"' near end] \\
B_\perp \arrow[r, "b"', "t" near end] & B
\end{tikzcd}
$$

in the category $(DOM_{tot})^\perp$ preserves this element $*_A$: $f(*_A) = f \circ a(\mathsf{up} \perp_A) = b \circ f_\perp(\mathsf{up} \perp_A) = b\big(\mathsf{up}\left(f(\perp_A)\right)\big) = b(\mathsf{up} \perp_B) = *_B$ (the least element in $B \setminus \{\perp_B\}$). Besides, $f_\perp(*_{A_\perp}) = f_\perp(\mathsf{up} \perp_A) = \mathsf{up} \perp_B = *_B$ follows from the strictness of f. On the other hand, the condition $f(*_B) = *_B$ is already sufficient to make the defining diagram for morphisms commute. Thus, we have found a complete characterisation for morphisms in $(DOM_{tot})^\perp$.

Now we are almost their. The category $DOM_\perp$ can be reconstructed as a category of algebras in DOM_{tot} in the following way:

PROPOSITION 2.14. *The categories $DOM_\perp$ and $(DOM_{tot})^\perp$ are equivalent.*

Proof. The equivalence of $(DOM_{tot})^\perp$ and $DOM_\perp$ is proved by constructing a functor $F\colon (DOM_{tot})^\perp \to DOM_\perp$ that is full, faithful and (up to isomorphisms) surjective on objects. Let F be defined by

$$F(a) \;:=\; A \setminus \{\perp_A\}$$

for all algebras $a\colon A_\perp \circ\!\!\rightarrow_t A$. A morphism

$$
\begin{tikzcd}
A_\perp \arrow[r, "a", "t"' near end] \arrow[d, "f_\perp"', "t" near end] & A \arrow[d, "f", "t"' near end] & & A \setminus \{\perp_A\} \arrow[d, "f^\downarrow"] \\
B_\perp \arrow[r, "b"', "t" near end] & B & \text{is mapped to} & B \setminus \{\perp_B\}
\end{tikzcd}
$$

with $f^\downarrow\colon x \mapsto f(x)$. As shown above, for every algebra $a\colon A_\perp \circ\!\!\rightarrow_t A$, the set $A \setminus \{\perp_A\}$ has a unique atom, therefore, it is indeed a domain. Now we check that F maps morphisms from $(DOM_{tot})^\perp$ to morphisms in $DOM_\perp$. Given a morphism

$$
\begin{tikzcd}
A_\perp \arrow[r, "a", "t"' near end] \arrow[d, "f_\perp"', "t" near end] & A \arrow[d, "f", "t"' near end] \\
B_\perp \arrow[r, "b"', "t" near end] & B
\end{tikzcd}
$$

in $(DOM_{tot})^{\perp}$, then $f^{\downarrow}\colon A\setminus\{\perp_A\} \circ\!\!\to B\setminus\{\perp_B\}$ is obviously continuous. Moreover, because of $f^{\downarrow}(*_A) = *_B$ (as shown above), it is also strict. Since F obviously preserves identities and compositions, it is a functor.

Next we verify that F is (up to isomorphisms) surjective on objects. Given an object A in $DOM_{\perp}$, then $(\mathsf{down}_A)_{\perp}\colon (A_{\perp})_{\perp} \circ\!\!\to_t A_{\perp}$ is an object in $(DOM_{tot})^{\perp}$ with $F\big((\mathsf{down}_A)_{\perp}\big) = A$.

For verifying that the functor F is also full, suppose that there is a morphism $f\colon A\setminus\{\perp_A\} \circ\!\!\to B\setminus\{\perp_B\}$ in $DOM_{\perp}$. Define $g := f_{\perp}\colon A \circ\!\!\to_t B$. This describes a morphism in $(DOM_{tot})^{\perp}$ since g is continuous, strict, total and preserves $*_A$. Moreover, F maps this morphism to f, thus, F is full.

Finally, we show that the functor F is also faithful. A morphism from $a\colon A_{\perp} \circ\!\!\to_t A$ to $b\colon B_{\perp} \circ\!\!\to_t B$ in $(DOM_{tot})^{\perp}$ is already uniquely determined by the function $f\colon A \circ\!\!\to_t B$. Suppose that $f, g\colon A \circ\!\!\to_t B$ are two maps such that the corresponding morphisms $f^{\downarrow}, g^{\downarrow}\colon A\setminus\{\perp_A\} \circ\!\!\to B\setminus\{\perp_B\}$ are equal in $DOM_{\perp}$. Then $f = (f^{\downarrow})_{\perp} = (g^{\downarrow})_{\perp} = g$ follows trivially, which implies that F is faithful. □

3 The Category *SD* of Sequential Domains

In the original Scott model, there are continuous functions such as the parallel-or that are not definable in a sequential functional language like PCF. In the category presented in this work, these functions do not exist. The idea is to require the morphisms to preserve a special kind of relations. This technique goes back to previous works [OR95b, RS97], however, the model here can be used for a wider variety of languages, and, from a categorical point, satisfies more desirable properties.

3.1 Characterising Sequentiality

Sequentiality Systems

We first need a preliminary definition which is a simplification of Riecke's and Sandholm's concept of *computational theories* [RS97]. Condition (P3) does not occur in their approach, but it is needed to deal with liftings.

DEFINITION 3.1. *Let w (for world) be a finite set. We call a partition on a subset of w a* partial partition *on w. A set of partial partitions S on w is called a* sequentiality system *on w if it is closed under the following closure conditions:*

(P1) if $w \neq \varnothing$ then $\{w\} \in S$,

(P2) if $v \in P \in S$ then $P \setminus \{v\} \in S$ (dropping),

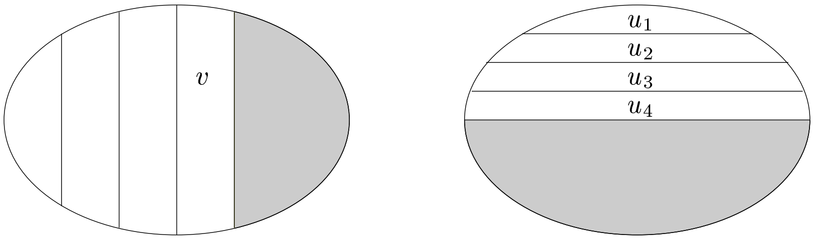
(P3) if $u, v \in P \in S$ then $(P \setminus \{u, v\}) \cup \{u \cup v\} \in S$ (joining),

(P4) if $P_1, P_2 \in S$, $v \in P_1$ then $(P_1 \setminus \{v\}) \cup \{u \cap v \mid u \in P_2,\ u \cap v \neq \varnothing\} \in S$ (refinement).

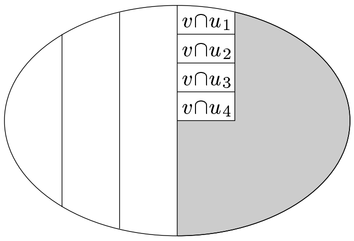
It sometimes happens that, for some predicate Φ, the set $v = \{x \in w \mid \Phi(x)\}$ turns out to be empty and, therefore, the statement $\{v\} \in S$ for a sequentiality system S would contradict the definition of partial partitions. In order to get rid of this problem, we write P^- for the partial partition $P \setminus \{\varnothing\}$. In particular, the last set in condition (P4) can be rewritten as $(P_1 \setminus \{v\}) \cup \{u \cap v \mid u \in P_2\}^-$.

The above closure conditions are formulated in an abstract way, allowing for general reasoning. To give some intuition, imagine that a sequentiality system S on a finite set w describes a certain class of algorithms, for instance, algorithms that are in some sense sequential. Concretely, the statement $\{v_1, \dots, v_n\} \in S$ suggests that there is a program that, given an $x \in w$, computes the index of a class v_i if x is in v_i and diverges in case of $x \notin v_1 \cup \dots \cup v_n$. Although it may be intuitively clear what sequentiality means for ground types, one can certainly argue about the meaning of sequentiality for higher types. Sequentiality systems as presented here do not fix a notion of sequentiality, but provide an abstract pattern, requiring some reasonable conditions for algorithms in general and sequential ones in particular.

Let us focus at the closure conditions in detail. The rather trivial condition (P1) claims that there is always a program that takes as an element $x \in w$ as input and returns 'yes'. Condition (P2) and (P3) say that, if there are programs that can compute to which of the classes $v_1, \ldots, v_n$ an element $x \in w$ belongs to, then there also exist programs calculating in which of the classes $v_2, \ldots, v_n$ and $v_1 \cup v_2, v_3, \ldots, v_n$, respectively, it is. Finally, condition (P4) means that, if there is a program computing in which of the classes in $P = \{v_1, \ldots, v_m, v\} \in S$ an element is and if there is another program that calculates to which of the classes $\{u_1, \ldots, u_n\}$ elements of w belong to, then there is a program telling in which of the classes $v_1, \ldots, v_m, v \cap u_1, \ldots, v \cap u_n$ an element $x \in w$ can be found. This is illustrated in the following diagram:



$P_1 \in S$ $\qquad\qquad\qquad$ $P_2 \in S$



$$\Longrightarrow \quad \big(P_1 \setminus \{v\}\big) \cup \{u \cap v \mid u \in P_2,\ u \cap v \neq \varnothing\} \ \in \ S$$

Note that condition (P4) claims more than that S is closed under intersections.

From my point of view, every collection of algorithms that can be understood as 'the class of all sequential functions' respects these closure conditions. So I think condition (P1)–(P4) are reasonable.

Sequential Function Systems

Let $\overline{n}_\perp$ be the flat domain with n maximal elements $1, \ldots, n$ and a least element $\perp$. A sequentiality system S on w can also be presented as a family $(S_n)_{n \in \mathbb{N}}$ of sets of functions from w to $\overline{n}_\perp$, naming the equivalence classes of a partial partition in S.

DEFINITION 3.2. *Let S_n be a nonempty set of functions $\varphi \colon w \to \overline{n}_\perp$ for all elements $n \in \mathbb{N}$. The family $(S_n)_{n \in \mathbb{N}}$ is called* sequential function system *on w if it satisfies the following conditions:*

(S1) $\varphi \colon w \to \overline{n}_\perp \in S_n$, $\mu \colon \overline{n}_\perp \to \overline{m}_\perp$ *monotone* $\implies$ $\mu \circ \varphi \in S_m$,

(S2) $\varphi \in S_2$, $\psi_1 \in S_{m_1}$, $\psi_2 \in S_{m_2}$ $\implies$ $\theta \colon w \to \overline{(m_1 + m_2)}_\perp \in S_{m_1 + m_2}$,

where θ is defined by

$$\theta(x) \;=\; \begin{cases} \psi_1(x) & \text{if } \varphi(x) = 1 \\ m_1 + \psi_2(x) & \text{if } \varphi(x) = 2 \\ \perp & \text{if } \varphi(x) = \perp \end{cases} .$$

Both definitions are indeed equivalent:

LEMMA 3.3. *Let w be a finite set. For every sequentiality system S on w, there is sequential function system $(S_n)_{n\in\mathbb{N}}$ on w and, conversely, for every sequential function system $(S_n)_{n\in\mathbb{N}}$ on w, there is a sequentiality system S on w such that, for all $n \in \mathbb{N}$, all partial partitions in S are enumerated by functions $\varphi\colon w \to \overline{n}_\perp$ in some S_n and all such functions enumerate a partial partition in S.*

Proof. Let S be a sequentiality system on w. The corresponding sequential function system $(S_n)_{n\in\mathbb{N}}$ consists of all functions $\varphi\colon w \to \overline{n}$ that enumerate a partial partition $P \in S$, i.e. $\varphi(x) = \perp$ holds if, and only if, $x \notin \bigcup P$ holds and $\varphi(x) = \varphi(y) \neq \perp$ is true if, and only if, there is a $v \in P$ with $x \in v$ and $y \in v$.

For verifying condition (S1), we assume that $\varphi \in S_n$ is an enumeration of a partial partition $P \in S$. Moreover, let $\mu\colon \overline{n}_\perp \to \overline{m}_\perp$ be a monotone map. If $\mu(\perp) = \perp$ holds, then $\mu \circ \varphi$ is a renumbering of P where some classes might be thrown together and some might be dropped. Since S is closed under (P2) and (P3), we can find a partial partition in S which is enumerated by $\mu \circ \varphi$. On the other hand, if $\mu(\perp) \neq \perp$ is true, then, because of the monotonicity of μ, $\mu \circ \varphi$ is a constant function enumerating the singleton partition $\{w\}$, which is in S because of condition (P1).

For the second condition, assume $\varphi \in S_2$, $\psi_1 \in S_{m_1}$ and $\psi_2 \in S_{m_2}$. The corresponding partitions are denoted by $\{v_1, v_2\}^-$, $\{u_1^1, \ldots, u_{m_1}^1\}$ and $\{u_1^2, \ldots, u_{m_2}^2\}$. Because of condition (P4) (applied twice) the partial partition $\{v_1 \cap u_i^1 \mid i = 1, \ldots, m_1\}^- \cup \{v_2 \cap u_i^2 \mid i = 1, \ldots, m_2\}^-$ is also in S. Since it is enumerated by θ, this shows that $(S_n)_{n\in\mathbb{N}}$ is closed under (S2).

For the converse direction, let $\varphi \in S_n$ be an element of a sequential function system $(S_n)_{n\in\mathbb{N}}$ on w. Then the set $\{\varphi^{-1}(i) \mid i = 1, \ldots, n\}$ is a partial partition on w with not more than n disjoint classes. We prove that the set S of all those partial partitions is a sequentiality system.

Since S_n is nonempty (for all $n \in \mathbb{N}$), the singleton partition w is enumerated by any function $(\lambda x{:}\overline{n}_\perp.1) \circ \varphi$ where φ is an element of S_n. Hence, condition (P1) holds. Given a partial partition $\{v_1, \ldots, v_n\} \in S$ with an enumerating function $\varphi\colon w \to n$ in S_n, then

$$\left(\lambda x{:}\overline{n}_\perp . \begin{cases} x & \text{if } x \neq n \\ \perp & \text{if } x = n \end{cases}\right) \circ \varphi \qquad \text{and} \qquad \left(\lambda x{:}\overline{n}_\perp . \begin{cases} x & \text{if } x \neq n \\ x-1 & \text{if } x = n \end{cases}\right) \circ \varphi$$

correspond to $\{v_1, \ldots, v_{n-1}\}$ and $\{v_1, \ldots, v_{n-1} \cup v_n\}$. Thus, conditions (P2) and (P3) hold for S.

For the verification of (P4) let $\{v_1, \ldots, v_n\}$ and $\{u_1, \ldots, u_n\}$ be partial partitions in S, enumerated by functions ψ_1 and ψ_2. If we take φ as an enumeration for $\{v_1 \cup \cdots \cup v_{n-1}, v_n\}$, then θ as defined in (S2) corresponds to $\{v_1, \ldots, v_{n-1}\} \cup \{v_n \cap u_i \mid i = 1, \ldots, m\}^- \in S$. Hence, (P4) holds which completes the proof. $\square$

Although the first description is more abstract and might look more suggestive from a mathematical point of view, it is sometimes more convenient to use the latter one. In particular, it is easy to see that the family of all intersections of relations

$$R^w_{v_1,v_2} = \{\varphi\colon w \to \overline{n}_\perp \mid (\forall x{\in}v_1.\ \varphi(x) \neq \perp) \implies \forall x, y{\in}v_2.\ \varphi(x) = \varphi(y)\},$$

($v_1 \subseteq v_2 \subseteq w$) as originally defined by K. Sieber [Sie92] form a sequential function system, i. e. it satisfies conditions (S1) and (S2). The only constraint to Sieber's original definition is the finiteness of the codomain of the elements φ: Sieber uses $\mathbb{N}_\perp$ and we restrict ourselves to finite approximations $\overline{n}_\perp$. This does not affect the standard proof of por not being a morphism: identifying the flat Boolean domain with $\overline{2}_\perp$, true with 1 and false with 2 yields $(1, \perp, \perp), (\perp, 1, \perp) \in S_2 := R^{\{1,2,3\}}_{\{1,2\},\{1,2,3\}}$, however, $\mathsf{por}\big((1, \perp, \perp), (\perp, 1, \perp)\big) = (1, 1, \perp)$ is not related by S_2.

3.2 Kripke Logical Relations

In order to characterise definability for the simply typed λ-calculus, A. Jung and J. Tiuryn introduced *Kripke relations* with varying arity [JT93]. They were also used by P. O'Hearn and J. Riecke to build a domain theoretic fully abstract model for PCF [OR95b]. We will extend this technique to study families of sequentiality systems over different sets w that are stable under re-indexing. The idea is to define a (countable) category *SPIC* (for sequentiality partition index category) whose subcategories can be used as index categories. Objects $\mathsf{w} = (w, S)$ consist of a finite subset $w \subseteq \mathbb{N}$ and a sequentiality system S on w. A morphism in *SPIC* is a function $\mu\colon (v, S^{\mathsf{v}}) \to (w, S^{\mathsf{w}})$ which is stable under re-indexing, i. e. it satisfies the *Kripke monotonicity condition*:

$$P \in S^{\mathsf{w}} \quad \implies \quad \{\mu^{-1}(u) \mid u \in P\} \in S^{\mathsf{v}}$$

or, equivalently, $\varphi \in S^{\mathsf{w}}_n \implies \varphi \circ \mu \in S^{\mathsf{v}}_n$ for sequential function systems.

The notion of sequentiality systems induces an extension of *logical relations* on arbitrary domains. Given a (finite) set w, a *w-ary relation* R on A is a set of functions $r\colon w \to A$. Recall that $r \restriction v$ stands for the function given by $(r \restriction v)(x) = r(x)$ for $x \in v$ and $(r \restriction v)(x) = \perp$ for $x \notin v$.

DEFINITION 3.4. *Let A be a domain and $\mathsf{w} = (w, S)$ be an object in SPIC. A w-ary relation R on A is called* w-logical relation *on A if it satisfies the following conditions:*

(R1) $\forall D \subseteq R.\ D$ *is directed* $\implies \bigsqcup^{\uparrow} D \in R$ (admissibility),

(R2) $\forall a{\in}A.\ \lambda x{:}w.a \in R$ (reflexivity),

(R3) $r \in R$, $\{v\} \in S \implies r \restriction v \in R$ (restriction),

(R4) $\{v_1, \ldots, v_n\} \in S$, $r \restriction v_i \in R$ *(for $i = 1, \ldots, n$)* $\implies r \restriction (v_1 \cup \cdots \cup v_n) \in R$ (gluing).

Conditions (R3) and (R4) can also be formulated differently, they are equivalent to:

(R3') $P \in S,\ p\colon P \to R \implies \mathsf{glue}(P,p) := \bigsqcup_{v\in P} p(v) \restriction v \in R.$

The operation glue can be understood as a case analysis:

$$\mathsf{glue}(P,p)(x) \;=\; \begin{cases} \big(p(v)\big)(x) & \text{if } x \in v \\ \bot & \text{if } x \notin \bigcup P \end{cases}$$

In the following, we sometimes use the more complicated looking condition (R3') instead of (R3) and (R4) as this turns out to be handy in proofs.

The extension to Kripke relations is the following:

DEFINITION 3.5. *Let $\mathcal{C}$ be a subcategory of SPIC and assume that, for all objects w in $\mathcal{C}$, R^{w} is a w-logical relation on a domain A. The family $\mathcal{R} = (R^{\mathsf{w}})_{\mathsf{w}\in\mathsf{Obj}(\mathcal{C})}$ is called* $\mathcal{C}$-Kripke logical relation *on A if it is* $\mathcal{C}$-Kripke monotone*:*

$$r \in R^{\mathsf{w}},\ \mu\colon \mathsf{v} \xrightarrow{\mathcal{C}} \mathsf{w} \qquad\Longrightarrow\qquad r\circ\mu \in R^{\mathsf{v}}.$$

The set of all Kripke relations itself satisfies a closure condition as well:

LEMMA 3.6. *Let $\mathcal{C}$ be a subcategory of SPIC and A be a domain. The set $\mathcal{R}$ of all $\mathcal{C}$-Kripke logical relations on A forms a* closure system*, i. e. it is closed under (possibly empty) intersections.*

Proof. Let $(R_i)_{i\in I}$ be a family of $\mathcal{C}$-Kripke logical relations on A. Then the intersection $R := \bigcap_{i\in I} R_i$ is given componentwise by $R^{\mathsf{w}} = \bigcap_{i\in I} R_i^{\mathsf{w}}$. It is easy to check that R^{w} satisfies conditions (R1)–(R4) for every $\mathsf{w} \in \mathsf{Obj}(\mathcal{C})$ if all R_i^{w} do so as well. Hence, it remains to show that R is $\mathcal{C}$-Kripke monotone. Let v,w be objects in $\mathcal{C}$, $\mu\colon \mathsf{v} \xrightarrow{\mathcal{C}} \mathsf{w}$ be a morphism and assume $r \in R^{\mathsf{w}}$. Since r is in all relations R_i^{w}, $r\circ\mu \in R_i^{\mathsf{v}}$ follows for all $i \in I$. Thus, $r\circ\mu$ is indeed an element of R^{v}, which implies that R is a $\mathcal{C}$-Kripke logical relation on A. $\square$

The least $\mathcal{C}$-Kripke logical relation $\mathsf{cl}(R) := \bigcap\{T \in \mathcal{R} \mid T^{\mathsf{w}} \supseteq R^{\mathsf{w}} \text{ for all } \mathsf{w} \in \mathsf{Obj}(\mathcal{C})\}$ that contains a family $R = (R^{\mathsf{w}})_{\mathsf{w}\in\mathsf{Obj}(\mathcal{C})}$ of w-ary relations is called the *closure* of R. If R is already $\mathcal{C}$-Kripke monotone, its closure $\mathsf{cl}(R)$ can be presented in a more constructive way. Let us fix an object $\mathsf{w} = (w,S)$ in $\mathcal{C}$ and consider the functions $\rho_1^{\mathsf{w}}, \rho_2^{\mathsf{w}}\ \rho_{3'}^{\mathsf{w}}, : \mathcal{P}(A^w) \to \mathcal{P}(A^w)$ defined by

$$\begin{aligned} \rho_1^{\mathsf{w}}(R^{\mathsf{w}}) &= \bigcap\big\{Q \subseteq [w \to A] \mid Q \supseteq R^{\mathsf{w}},\ Q \text{ admissible}\big\}, \\ \rho_2^{\mathsf{w}}(R^{\mathsf{w}}) &= R^{\mathsf{w}} \cup \{\lambda x{:}w.a \mid a \in A\}, \\ \rho_{3'}^{\mathsf{w}}(R^{\mathsf{w}}) &= \big\{\mathsf{glue}(P,p) \mid P \in S,\ p\colon P \to R^{\mathsf{w}}\big\} \end{aligned}$$

in accordance to conditions (R1), (R2) and (R3'). The function ρ_1^{w} can also be described as follows (see [AJ94], Exercise 2.3.9(33)):

$$\rho_1^{\mathsf{w}}(R^{\mathsf{w}}) = s_{2^{|R^{\mathsf{w}}|}}(R^{\mathsf{w}})$$

where the s_κ are defined by $s_0(R^{\mathsf{w}}) = R^{\mathsf{w}}$, $s_{\alpha+1}(R^{\mathsf{w}}) = \big\{ \bigsqcup^{\uparrow} P \mid P \subseteq s_\alpha(R^{\mathsf{w}}),\ P \text{ directed}\big\}$ and $s_\alpha(R^{\mathsf{w}}) = \bigcup_{\beta<\alpha} s_\beta(R^{\mathsf{w}})$ if α is a limit ordinal.

LEMMA 3.7. *Let $\mathcal{C}$ be a subcategory of SPIC, A be a domain and $R = (R^{\mathsf{w}})_{\mathsf{w} \in \mathsf{Obj}(\mathcal{C})}$ be a family of w-ary relations on A that is $\mathcal{C}$-Kripke monotone. Then the smallest $\mathcal{C}$-Kripke logical relation $\mathsf{cl}(R)$ on A that contains R is given by*

$$\big(\mathsf{cl}(R)\big)^{\mathsf{w}} \;:=\; \rho_1^{\mathsf{w}} \circ \rho_{3'}^{\mathsf{w}} \circ \rho_2^{\mathsf{w}}(R^{\mathsf{w}})$$

for all objects $\mathsf{w} = (w, S)$ in SPIC.

Proof. Let w be a world in an index category $\mathcal{C}$, we first show that $\big(\mathsf{cl}(R)\big)^{\mathsf{w}}$ is closed under conditions (R1), (R2) and (R3'). It is straightforward to verify that all functions $\rho_1^{\mathsf{w}}, \rho_2^{\mathsf{w}}, \rho_{3'}^{\mathsf{w}}$ are idempotent. Moreover, it is obvious that $\big(\mathsf{cl}(R)\big)^{\mathsf{w}}$ is closed under conditions (R1) and (R2). In order to show that it is also closed under (R3'), let $P = \{v_1, \dots, v_n\} \in S$ be a partial partition. Moreover, suppose that $R^P = \{p \colon P \to R\}$ is closed under gluing in the sense that, for all $p \in R^P$, $\mathsf{glue}(P, r) \in R$ is true. We prove that $\rho_1^{\mathsf{w}}(R)$ is closed under gluing as well. It suffices to show that the set

$$M \;:=\; \big\{p \in R^P \mid \mathsf{glue}(P, p) \in \rho_1^{\mathsf{w}}(R)\big\}$$

is closed under directed suprema, then $\big(\rho_1^{\mathsf{w}}(R)\big)^P \subseteq M$ follows as $\big(\rho_1^{\mathsf{w}}(R)\big)^P$ is the intersection of all admissible sets containing R^P. Let $D \subseteq M$ be a directed subset. For every $d \in D$, $\mathsf{glue}(P, d)$ is in R^P as R^P is closed under gluing. Therefore, $\mathsf{glue}(P, \bigsqcup^{\uparrow} D) = \bigsqcup^{\uparrow}{}_{d \in D}\, \mathsf{glue}(P, d) \in \big(\rho_1^{\mathsf{w}}(R)\big)^P$ follows. The finishes the proof of $\big(\rho_1^{\mathsf{w}}(R)\big)^P \subseteq M$ which implies that $\rho_1^{\mathsf{w}}(R)$ is closed under condition (R3').

Next, we show that the family of functions $\rho_1 \circ \rho_{3'} \circ \rho_2 = (\rho_1^{\mathsf{w}} \circ \rho_{3'}^{\mathsf{w}} \circ \rho_2^{\mathsf{w}})_{\mathsf{w} \in \mathsf{Obj}(\mathcal{C})}$ preserves stability under re-indexing with morphisms of $\mathcal{C}$. Let $(R^{\mathsf{w}})_{\mathsf{w} \in \mathsf{Obj}(\mathcal{C})}$ be a family of w-ary relations that is stable under re-indexing, r an element of $\rho_i^{\mathsf{w}}(R^{\mathsf{w}})$, and $\mu \colon (v, S^{\mathsf{v}}) \xrightarrow{\mathcal{C}} (w, S^{\mathsf{w}})$ a morphism in $\mathcal{C}$, we prove $r \circ \mu \in \rho_i^{\mathsf{v}}(R^{\mathsf{v}})$ for $i = 1, 2, 3'$. This entails that $\rho_1 \circ \rho_{3'} \circ \rho_2(R) := \big(\rho_1^{\mathsf{w}} \circ \rho_{3'}^{\mathsf{w}} \circ \rho_2^{\mathsf{w}}(R^{\mathsf{w}})\big)_{\mathsf{w} \in \mathsf{Obj}(\mathcal{C})}$ is $\mathcal{C}$-Kripke monotone and, therefore, it is the smallest $\mathcal{C}$-Kripke logical relation that contains R.

(R1): We prove that $M := \{r \in R_{\mathcal{C}}^{\mathsf{w}} \mid r \circ \mu \in \rho_1^{\mathsf{v}}(D_{\mathcal{C}}^{\mathsf{v}})\}$ is closed under directed suprema. Let $D \subseteq M$ be a directed subset. Then $\bigsqcup^{\uparrow} D \circ \mu = \bigsqcup^{\uparrow}{}_{d \in D}(d \circ \mu) \in \rho_1^{\mathsf{v}}(R_{\mathcal{C}}^{\mathsf{v}})$ follows as all functions $d \circ \mu$ are in $R_{\mathcal{C}}^{\mathsf{v}}$. This shows that $\big(\rho_1^{\mathsf{w}}(R^{\mathsf{w}})\big)_{\mathsf{w} \in \mathsf{Obj}(\mathcal{C})}$ is $\mathcal{C}$-Kripke monotone.

(R2): Trivial.

(R3'): Let $P \in S^{\mathsf{w}}$ be a partial partition with $r = \mathsf{glue}(P, p)$. Then $r \circ \mu = \mathsf{glue}(P, p) \circ \mu = \mathsf{glue}\big(\{\mu^{-1}(u) \mid u \in P\}, p \circ \mu\big) \in \rho_3^{\mathsf{v}}(R^{\mathsf{v}})$ follows as $\{\mu^{-1}(u) \mid u \in P\}$ is a partial partition in S^{v}.

This finishes the proof of the claimed constructive presentation of $\mathsf{cl}(R)$. $\square$

The following lemma shows that the operator ρ_1 (mapping a set to the least superset that is closed under directed suprema) does not interfere with the property of a function to preserve a relation. It gives a handy tool that we will use several times in the proof of the main theorem of this section (Theorem 3.10).

LEMMA 3.8. *Let $f\colon A \to B$ be a continuous function and Q and R arbitrary w-ary relations on A and B, respectively. If $f \circ q \in R$ holds for all $q \in Q$, then $f \circ q \in \rho_1^{\mathsf{w}}(R)$ is true for all $q \in \rho_1^{\mathsf{w}}(Q)$.*

Proof. We have to show that the set $M := \{q \in Q \mid f \circ q \in \rho_1^{\mathsf{w}}(R)\}$ is closed under directed suprema. Consider a subset $D \subseteq M$, then $f \circ \bigsqcup^{\uparrow} D = \bigsqcup^{\uparrow}{}_{d \in D}(f \circ d) \in \rho_1^{\mathsf{w}}(R)$ follows because $f \circ d \in R$ holds by assumption. This implies $\rho_1^{\mathsf{w}}(Q) \subseteq M$ which completes the proof. $\square$

The lemma does not hold if ρ_1^{w} is replaced by $\rho_{3'}^{\mathsf{w}}$, it is already wrong for the closure operator ρ_3^{w} where $\rho_3^{\mathsf{w}}(R)$ is the smallest relation that is closed under restrictions to classes v with $\{v\} \in S$. For instance, consider $\mathsf{w} = (w, S)$ with $w = \{1, 2\}$ and $S := \{\varnothing, \{\{1\}\}, \{\{1, 2\}\}\}$. It is easy to check that S satisfies conditions (P1)–(P4). Furthermore, define $A := \mathbf{2}$, $B := \mathbf{2}_\perp$, $Q := \{\langle \top_{\mathbf{2}}, \top_{\mathbf{2}} \rangle\}$ and $R := \{\langle x, y\rangle \mid x = \perp_{\mathbf{2}_\perp} \text{ or } y = \perp_{\mathbf{2}_\perp} \text{ or } x = y\}$. Choosing $f := \mathsf{up}_{\mathbf{2}}\colon \mathbf{2} \to \mathbf{2}_\perp$, we get $\mathsf{up}_{\mathbf{2}} \circ \langle \top_{\mathbf{2}}, \top_{\mathbf{2}}\rangle = \langle \mathsf{up}_{\mathbf{2}}\, \top_{\mathbf{2}}, \mathsf{up}_{\mathbf{2}}\, \top_{\mathbf{2}}\rangle \in R$ for the only element $\langle \top_{\mathbf{2}}, \top_{\mathbf{2}}\rangle$ in Q, but $\mathsf{up}_{\mathbf{2}} \circ \big(\langle \top_{\mathbf{2}}, \top_{\mathbf{2}}\rangle \restriction \{1\}\big) = \langle \mathsf{up}_{\mathbf{2}}\, \top_{\mathbf{2}}, \mathsf{up}_{\mathbf{2}} \perp_{\mathbf{2}}\rangle$ fails to be related by $R = \rho_3^{\mathsf{w}}(R)$.

Nevertheless, it can be easily shown that the analogous claim of Lemma 3.8 is true for all closure operators $\rho_1, \rho_2, \rho_{3'}$ if we require the function f to be strict.

3.3 Sequential Domains

The category of sequential domains can now be defined as follows:

DEFINITION 3.9. *The category SD is given by a collection of objects and morphisms which are defined as follows:*

Objects: *Objects consist of a domain A and a family $\mathcal{Q}$ of $\mathcal{C}$-Kripke logical relations $Q_{\mathcal{C}}$ on A where $\mathcal{C}$ ranges over all subcategories of SPIC.*

Morphisms: *A morphism $f\colon (A, \mathcal{Q}) \to (B, \mathcal{R})$ is a continuous function which is* uniform, *i. e. it satisfies the condition*

$$q \in Q_{\mathcal{C}}^{\mathsf{w}} \quad \Longrightarrow \quad f \circ q \in R_{\mathcal{C}}^{\mathsf{w}}$$

for all subcategories $\mathcal{C}$ of SPIC and all objects $\mathsf{w} \in \mathsf{Obj}(\mathcal{C})$.

The full subcategory of those objects in *SD* where all families of Kripke relations consist of diagonals (i. e. relations of the form $\{(a, \dots, a) \mid a \in A\}$) is equivalent to the category *DOM* as all continuous functions are then automatically uniform. Likewise, the subcategory in which the w-ary relations on every object A are all given by the total relation A^w is equivalent to *DOM* as well.

As shown in the following theorem, the category *SD* is cartesian closed and has various kinds of products, sums and function spaces satisfying the universal domain theoretic properties as exhibited in Section 2. Hence, it contains models for call-by-name languages whose operational semantics is defined in accordance with domain theoretic structures. Such a language, augmented by a couple of call-by-value features, will be examined in the next section.

THEOREM 3.10. *The category SD is cpo-enriched, well-pointed and cartesian closed. It is also closed under smash products, liftings, separated sums and coalesced sums satisfying the universal properties as explained in the previous section. Moreover, the smash products together with the strict function spaces endow the subcategory $SD_\perp$ with a symmetric monoidal closed structure.*

If $Q_\mathcal{C}$, $R_\mathcal{C}$ and $(Q_i)_\mathcal{C}$ are $\mathcal{C}$-Kripke relations on the domains A, B and A_i, respectively, the relations on $A \times B$, $A \otimes B$ etc. are defined by

$$\mathbf{1}^{\mathsf{w}}_\mathcal{C} := \{\lambda x{:}w.\top\},$$

$$(Q \times R)^{\mathsf{w}}_\mathcal{C} := \big\{\langle q, r\rangle \colon w \to A \times B \mid q \in Q^{\mathsf{w}}_\mathcal{C}, r \in R^{\mathsf{w}}_\mathcal{C}\big\},$$

$$(Q \otimes R)^{\mathsf{w}}_\mathcal{C} := \rho^{\mathsf{w}}_1\big(\{(q, r) \colon w \to A \otimes B \mid q \in Q^{\mathsf{w}}_\mathcal{C}, r \in R^{\mathsf{w}}_\mathcal{C}\}\big)$$

(recall that $\rho^{\mathsf{w}}_1(P^{\mathsf{w}}_\mathcal{C})$ is the smallest admissible relation containing $P^{\mathsf{w}}_\mathcal{C}$),

$$(Q_\perp)^{\mathsf{w}}_\mathcal{C} := \big\{q \colon w \to A_\perp \,\big|\, \big\{\{x \in w \mid q(x) \in A\}\big\}^- \in S, \mathsf{down} \circ q \in Q^{\mathsf{w}}_\mathcal{C}\big\},$$

$$\Big(\sum\nolimits_{i=1}^n Q_i\Big)^{\mathsf{w}}_\mathcal{C} := \Big\{p \colon w \to \sum\nolimits_{i=1}^n A_i \,\Big|\, \varphi \in S_n,\, q_1 \in (Q_1)^{\mathsf{w}}_\mathcal{C} \dots q_n \in (Q_n)^{\mathsf{w}}_\mathcal{C},\, p(x) = \begin{cases} \mathsf{in}_i \circ q_i(x) & \text{if } \varphi(x) = i \\ \perp & \text{if } \varphi(x) = \perp \end{cases}\Big\},$$

$$(Q \oplus R)^{\mathsf{w}}_\mathcal{C} := \Big\{p \,\Big|\, \varphi \in S_2,\, q \in Q^{\mathsf{w}}_\mathcal{C},\, r \in R^{\mathsf{w}}_\mathcal{C},\, p(x) = \begin{cases} \widetilde{\mathsf{inl}} \circ q(x) & \text{if } \varphi(x) = 1 \\ \widetilde{\mathsf{inr}} \circ r(x) & \text{if } \varphi(x) = 2 \\ \perp & \text{if } \varphi(x) = \perp \end{cases}\Big\},$$

$$[Q \to R]^{\mathsf{w}}_\mathcal{C} := \big\{p \colon w \to [A \to B] \,\big|\, \forall \mu \colon \mathsf{v} \xrightarrow{\mathcal{C}} \mathsf{w}. \forall q {\in} Q^{\mathsf{v}}_\mathcal{C}.\ \lambda x{:}v.\big(p(\mu x)\big)(qx) \in R^{\mathsf{v}}_\mathcal{C}\big\},$$

$$[Q \multimap R]^{\mathsf{w}}_\mathcal{C} := \big\{p \colon w \to [A \multimap B] \,\big|\, \forall \mu \colon \mathsf{v} \xrightarrow{\mathcal{C}} \mathsf{w}. \forall q {\in} Q^{\mathsf{v}}_\mathcal{C}.\ \lambda x{:}v.\big(p(\mu x)\big)(qx) \in R^{\mathsf{v}}_\mathcal{C}\big\}$$

for every subcategory $\mathcal{C}$ of the index category SPIC and every object $\mathsf{w} = (w, S) \in \mathsf{Obj}(\mathcal{C})$.

We write $\mathbf{1}$ for the singleton domain as well as for the sequential domain with its uniquely determined family of Kripke relations. Consequently, the relations on the Sierpiński sequential domain $\mathbf{2} := \mathbf{1}_\perp$ are given by $\mathbf{2}^{\mathsf{w}}_\mathcal{C} = (\mathbf{1}_\perp)^{\mathsf{w}}_\mathcal{C} = S_1$. Note that there are objects in SD with $\{\perp, \top\}$ as underlying domain that are different from $\mathbf{2}$.

Proof. Firstly, we show that SD is cpo-enriched. Let $(f_i)_{i \in I}$ be a directed set of morphisms in $SD\big((A, \mathcal{Q}), (B, \mathcal{R})\big)$. Then the domain theoretic supremum $\bigsqcup^{\uparrow}_{i \in I} f_i$ is uniform since $(\bigsqcup^{\uparrow}_{i \in I} f_i) \circ q = \bigsqcup^{\uparrow}_{i \in I}(f_i \circ q) \in R^{\mathsf{w}}_\mathcal{C}$ holds for all $q \in Q^{\mathsf{w}}_\mathcal{C}$ because of condition (R1). This also guarantees that the composition of morphisms in SD is continuous. Hence, the category SD is cpo-enriched.

Given an object $(A, \mathcal{Q})$ in SD, then all morphisms $e \colon \mathbf{1} \to (A, \mathcal{Q})$ are uniform because of condition (R2). This implies that SD is indeed well-pointed.

Next, we verify that the conditions (R1)–(R4) remain true while applying a type constructor to objects.

For smash products, it is easy to check that $\{(q, r) \mid q \in Q^{\mathsf{w}}_\mathcal{C},\ r \in R^{\mathsf{w}}_\mathcal{C}\}$ already satisfies (R2)–(R4) if $Q^{\mathsf{w}}_\mathcal{C}$ and $R^{\mathsf{w}}_\mathcal{C}$ satisfy them. Hence, $(Q \otimes R)^{\mathsf{w}}_\mathcal{C}$ satisfies all four conditions (R1)–(R4) because of Lemma 3.7. Many of the different cases for the other constructions are straightforward and therefore omitted. Proofs for strict function spaces are basically the same as for total ones.

(R1) for $\mathcal{Q}_\perp$: Let $(q_i)_{i\in I}$ a directed family in $(Q_\perp)^{\mathsf{w}}_{\mathcal{C}}$. For every $x \in w$ with $\bigsqcup^{\uparrow}{}_{i\in I}\, q_i(x) \neq \perp$ there is an $i \in I$ such that $q_i(x) \neq \perp$. Since w is finite, there is a finite subset $J \subseteq I$ with $\{x \in w \mid \bigsqcup^{\uparrow}{}_{i\in I}\, q_i(x) \neq \perp\} = \{x \in w \mid \bigsqcup^{\uparrow}{}_{i\in J}\, q_i(x) \neq \perp\}$. The family $(q_i)_{i\in I}$ is directed, so there is an index $i_0 \in I$ such that q_{i_0} is an upper bound for $\{q_i \mid i \in J\}$. This entails $\big\{\{x \in w \mid \bigsqcup^{\uparrow}{}_{i\in I}\, q_i(x) \neq \perp\}\big\}^{-} = \big\{\{x \in w \mid q_{i_0}(x) \neq \perp\}\big\}^{-} \in S$, verifying the first part of the definition of $(Q_\perp)^{\mathsf{w}}_{\mathcal{C}}$. The second part is straightforward.

(R1) for $\mathcal{Q} \oplus \mathcal{R}$: Let $(p_i)_{i\in I}$ be a directed family in $(Q \oplus R)^{\mathsf{w}}_{\mathcal{C}}$. Then there are $\varphi_i \in S_2$, $q_i \in Q^{\mathsf{w}}_{\mathcal{C}}$ and $r_i \in R^{\mathsf{w}}_{\mathcal{C}}$ with $p_i(x) = \begin{cases} \widetilde{\mathsf{inl}} \circ q_i(x) & \text{if } \varphi_i(x) = 1 \\ \widetilde{\mathsf{inr}} \circ r_i(x) & \text{if } \varphi_i(x) = 2 \\ \perp & \text{if } \varphi_i(x) = \perp \end{cases}$ for all $i \in I$. Consider those elements $x \in w$ such that there exists an index $i_x \in I$ with $\varphi_j(x) = \varphi_{i_x}(x)$ for all $j \geq i_x$ (the case that there is no such x is trivial). Since the family $(q_i)_{i\in I}$ is directed and since w is finite, there is an upper bound i_0 of all those i_x $(x \in w)$. We show that $\{v_1, v_2\}^{-}$ with $v_1 = \bigcap_{j\geq i_0} \varphi_j^{-1}(\{1\})$ and $v_2 = \bigcap_{j\geq i_0} \varphi_j^{-1}(\{2\})$ is a partial partition in S by verifying that $\{u_1, u_2\}, \{u_1', u_2'\} \in S$ implies $\{u_1 \cap u_1', u_2 \cap u_2'\}^{-} \in S$. Because of condition (P2), $\{u_1', u_2'\} \in S$ yields $\{u_1'\} \in S$ and $\{u_2'\} \in S$. Condition (P4) applied twice leads to $\{u_1 \cap u_1', u_2\}^{-} \in S$ and $\{u_1 \cap u_1', u_2 \cap u_2'\}^{-} \in S$. The sets v_1 and v_2 are finite intersections of sets $\{u_i, u_i'\}$, thus $\{v_1, v_2\}^{-}$ is a partial partition in S that corresponds to a function $\varphi \in S_2$ with $\varphi(x) = \begin{cases} 1 & \text{if } \varphi_j(x) = 1 \text{ for all } j \geq i_0 \\ 2 & \text{if } \varphi_i(x) = 2 \text{ for all } j \geq i_0 \\ \perp & \text{otherwise} \end{cases}$.
It is now easy to check that the directed supremum $p := \bigsqcup^{\uparrow}{}_{i\in I}\, p_i$ is given by setting $q := q_{i_0} \restriction v_1$ and $r := r_{i_0} \restriction v_2$. Because of condition (R3), q and r are related by $Q^{\mathsf{w}}_{\mathcal{C}}$ and $R^{\mathsf{w}}_{\mathcal{C}}$, respectively.

(R3) for $\sum_{i=1}^{n} \mathcal{Q}_i$: For every function $p\colon w \to \sum_{i=1}^{n} \mathcal{Q}_i$ with $p \in (\sum_{i=1}^{n} \mathcal{Q}_i)^{\mathsf{w}}_{\mathcal{C}}$, there are $q_i \in (\mathcal{Q}_i)^{\mathsf{w}}_{\mathcal{C}}$ (for $i = 1, \ldots, n$) with $p(x) = \begin{cases} \mathsf{in}_i \circ q_i(x) & \text{if } \varphi(x) = i \\ \perp & \text{if } \varphi(x) = \perp \end{cases}$. Assume $\{v\} \in S$, then the restriction $p \restriction v$ is given by $(p \restriction v)(x) = \begin{cases} \mathsf{in}_i \circ (q_i \restriction v)(x) & \text{if } (\varphi \restriction v)(x) = i \\ \perp & \text{if } (\varphi \restriction v)(x) = \perp \end{cases}$.
Because of $\varphi \restriction v \in S_n$ (apply (P4) with $\{v\}$ to all classes of the corresponding partial partition of φ) and $q_i \restriction v \in (Q_i)^{\mathsf{w}}_{\mathcal{C}}$ (holds by induction hypothesis), this proves $p \restriction v \in (\sum_{i=1}^{n} Q_i)^{\mathsf{w}}_{\mathcal{C}}$.

(R3) for $[\mathcal{Q} \to \mathcal{R}]$: Assume $p \in [Q \to R]^{\mathsf{w}}_{\mathcal{C}}, \{v\} \in S$, $\mu\colon \mathsf{w}' \xrightarrow{\mathcal{C}} \mathsf{w}$ with $\mathsf{w}' = (w', S^{\mathsf{w}'})$ and $q \in Q^{\mathsf{w}'}_{\mathcal{C}}$. Since μ is $\mathcal{C}$-Kripke monotone, $\{v\} \in S$ implies $\{\mu^{-1}(v)\} \in S^{\mathsf{w}'}$. Using the assumption $p \in [Q \to R]^{\mathsf{w}}_{\mathcal{C}}$ leads to $\lambda\!\!\lambda x{:}w'.\big(p \circ \mu(x)\big)(qx) \restriction \mu^{-1}(v) \in R^{\mathsf{w}'}_{\mathcal{C}}$ by applying the induction hypothesis. This completes the proof because of $\lambda\!\!\lambda x{:}w'.\big(p \circ \mu(x)\big)(qx) \restriction \mu^{-1}(v) = \lambda\!\!\lambda x{:}w'.\big((p \restriction v) \circ \mu(x)\big)(qx)$.

(R4) for $\mathcal{Q}_\perp$: Assume $\{v_1, \ldots, v_n\} \in S$ and $q \restriction v_i \in (Q_\perp)^{\mathsf{w}}_{\mathcal{C}}$ for all $i = 1, \ldots, n$. By induction hypothesis, $\big\{v_i \cap \{x \in w \mid q(x) \neq \perp\}\big\}^{-} = \big\{\{x \in w \mid (q \restriction v_i)(x) \neq \perp\}\big\}^{-} \in S$ follows. Thus, $\big\{v_1 \cap \{x \in w \mid (q \restriction v_1)(x) \neq \bot\}, \dots, v_n \cap \{x \in w \mid (q \restriction v_n)(x) \neq \bot\}\big\}^- = \big\{v_1 \cap \{x \in w \mid q(x) \neq \bot\}, \dots, v_n \cap \{x \in w \mid q(x) \neq \bot\}\big\}^- \in S$ can be shown by applying condition (P4) n-times. This leads to $\big\{\{x \in w \mid \big(q \restriction (v_1 \cup \dots \cup v_n)\big)(x) \neq \bot\}\big\}^- = \big\{\bigcup_{i=1}^{n} \big(v_i \cap \{x \in w \mid q(x) \neq \bot\}\big)\big\}^- \in S$ due to condition (P3). It is straightforward to verify $\mathsf{down} \circ \big(q \restriction (v_1 \cup \dots \cup v_n)\big) \in Q^{\mathsf{w}}_{\mathcal{C}}$.

(R4) for $\sum_{i=1}^{n} \mathcal{Q}_i$: Let $P \in S$ be a partial partition on w and suppose $p \restriction v \in (\sum_{i=1}^{n} Q_i)^{\mathsf{w}}_{\mathcal{C}}$ for all $v \in P$. By definition of $(\sum_{i=1}^{n} Q_i)^{\mathsf{w}}_{\mathcal{C}}$ there are functions $q_i^v \in (Q_i)^{\mathsf{w}}_{\mathcal{C}}$ ($i = 1, \dots, n$, $v \in P$) and $\varphi_v \in S_n$ with $(p \restriction v)(x) = \begin{cases} \mathsf{in}_i \circ q_i^v(x) & \text{if } \varphi_v(x) = i \\ \bot & \text{if } \varphi_v(x) = \bot \end{cases}$. The function $p \restriction \bigcup P$ is then given by $(p \restriction \bigcup P)(x) = \begin{cases} \mathsf{in}_i \circ q_i(x) & \text{if } \varphi(x) = i \\ \bot & \text{if } \varphi(x) = \bot \end{cases}$ where φ and q_i are defined by $\varphi(x) := \begin{cases} \varphi_v(x) & \text{if } x \in v \\ \bot & \text{if } x \notin \bigcup P \end{cases}$ and $q_i(x) := \begin{cases} q_i^v(x) & \text{if } x \in v \\ \bot & \text{if } x \notin \bigcup P \end{cases}$. Because of $\varphi \in S_n$ (apply (S2)) and $q_i \in (Q_i)^{\mathsf{w}}_{\mathcal{C}}$ (follows from $q_i \restriction v = q_i^v \restriction v$, $q_i = q_i \restriction \bigcup P$ and the induction hypothesis), this yields $p \restriction \bigcup P \in (\sum_{i=1}^{n} Q_i)^{\mathsf{w}}_{\mathcal{C}}$.

(R4) for $[\mathcal{Q} \to \mathcal{R}]$: Assume $P \in S$ is a partial partition with $p \restriction u \in [Q \to R]^{\mathsf{w}}_{\mathcal{C}}$ for $u \in P$. Let $\mu \colon \mathsf{v} \xrightarrow{\mathcal{C}} \mathsf{w}$ be a morphism in $\mathcal{C}$ and suppose $q \in Q^{\mathsf{w}}_{\mathcal{C}}$. Then $\lambda x{:}v.\big(p(\mu(x))\big)\big(q(x)\big) \restriction u = \lambda x{:}v.\big((p \restriction u)(\mu(x))\big)\big(q(x)\big) \in R^{\mathsf{v}}_{\mathcal{C}}$ follows by definition of $[A \to B]^{\mathsf{w}}_{\mathcal{C}}$. Because μ is $\mathcal{C}$-Kripke monotone, $\{\mu^{-1}(u) \mid u \in P\}$ and therefore also $\{\mu^{-1}(\bigcup P)\} = \{\mu^{-1}(u) \mid u \in P\}$ are partial partitions in S. This implies $\lambda x{:}v.\big((p \restriction \mu^{-1}(\bigcup P))(\mu(x))\big)\big(q(x)\big) = \lambda x{:}v.\big(p(\mu(x))\big)\big(q(x)\big) \restriction \mu^{-1}(\bigcup P) \in R^{\mathsf{v}}_{\mathcal{C}}$ by induction hypothesis.

It is an easy induction to verify that the relations in the theorem preserve Kripke monotonicity. Two examples are given to illustrate the technique, the other cases follow the same principle. Assume that $\mu\colon (v, S^{\mathsf{v}}) \to (w, S^{\mathsf{w}})$ is a morphism in $\mathcal{C}$.

$\mathcal{Q} \oplus \mathcal{R}$: Assume that p is related by $(Q \oplus R)^{\mathsf{w}}_{\mathcal{C}}$, then there are functions $\varphi \in S_2^{\mathsf{w}}$, $q \in Q^{\mathsf{w}}_{\mathcal{C}}$ and $r \in R^{\mathsf{w}}_{\mathcal{C}}$ with $p(x) = \begin{cases} \widetilde{\mathsf{inl}} \circ q(x) & \text{if } \varphi(x) = 1 \\ \widetilde{\mathsf{inr}} \circ r(x) & \text{if } \varphi(x) = 2 \\ \bot & \text{if } \varphi(x) = \bot \end{cases}$. Thus, $\varphi \circ \mu \in S_2^{\mathsf{v}}$, $q \circ \mu \in Q^{\mathsf{v}}_{\mathcal{C}}$ and $r \circ \mu \in R^{\mathsf{v}}_{\mathcal{C}}$ follow by definition and by induction hypothesis, respectively. This implies $p \circ \mu \in (Q \oplus R)^{\mathsf{v}}_{\mathcal{C}}$.

$[\mathcal{Q} \to \mathcal{R}]$: Suppose $p \in [Q \to R]^{\mathsf{w}}_{\mathcal{C}}$. We have to show $p \circ \mu \in [Q \to R]^{\mathsf{v}}_{\mathcal{C}}$. Given $q \in Q^{\mathsf{v}'}_{\mathcal{C}}$ and $\nu\colon \mathsf{v}' \xrightarrow{\mathcal{C}} \mathsf{v}$, then $\lambda x{:}v'.\big(p \circ \mu \circ \nu(x)\big)(qx) \in R^{\mathsf{v}'}_{\mathcal{C}}$ follows from the assumption $p \in [Q \to R]^{\mathsf{w}}_{\mathcal{C}}$.

We now show that SD is cartesian closed. Because DOM is already cartesian closed, it suffices to verify that the operators $\times$ and $\to$ are functors on SD and that the projections pr_1, pr_2, the pairing $\langle \cdot, \cdot \rangle$, the abstraction $d \mapsto \lambda x.d$, the evaluation map eval, and the currying $\mathsf{curry}\, f$ of a morphism f are uniform. The nontrivial cases are:

Abstraction: Assume $a \in A$ and let $f\colon (A, \mathcal{Q}) \times (C, \mathcal{P}) \to (B, \mathcal{R})$ be a morphism in *SD*. We need to prove that $(\mathsf{curry}\, f)(a)\colon (C, \mathcal{P}) \to (B, \mathcal{R})$ is uniform. Given $p \in P^{\mathsf{w}}_{\mathcal{C}}$, then $(\mathsf{curry}\, f)(a) \circ p = \lambda c{:}C.f\big(\langle a, c\rangle\big) \circ p = \lambda x{:}w.f\big(\langle a, p(x)\rangle\big) = f \circ \langle \lambda x{:}A.a, p\rangle \in R^{\mathsf{w}}_{\mathcal{C}}$ follows, which shows the uniformity of $(\mathsf{curry}\, f)(a)$.

Evaluation: Suppose $\langle q, p\rangle \in \big([P \to R] \times P\big)^{\mathsf{w}}_{\mathcal{C}}$. Then $\mathsf{eval} \circ \langle q, p\rangle = \lambda x{:}w.\big(q(x)\big)\big(p(x)\big) \in R^{\mathsf{w}}_{\mathcal{C}}$ follows from the definition of the relation on products and function spaces.

Currying: Let $f\colon (A, \mathcal{Q}) \times (C, \mathcal{P}) \to (B, \mathcal{R})$ be a morphism in *SD*, we need to verify that $\mathsf{curry}\, f\colon (A, \mathcal{Q}) \to ([C \to B], [\mathcal{P} \to \mathcal{R}])$ is uniform. Given $q \in Q^{\mathsf{w}}_{\mathcal{C}}$, $\mu\colon \mathsf{v} \xrightarrow{\mathcal{C}} \mathsf{w}$ and $p \in P^{\mathsf{v}}_{\mathcal{C}}$, then $\lambda x{:}v.\big((\mathsf{curry}\, f) \circ q\big(\mu(x)\big)\big)\big(p(x)\big) = f \circ \langle q \circ \mu, p\rangle \in R^{\mathsf{v}}_{\mathcal{C}}$ follows. This shows $(\mathsf{curry}\, f) \circ q \in [P \to R]^{w}_{\mathcal{C}}$.

Given two strict morphisms $f_1\colon (A_1, \mathcal{Q}_1) \circ\!\!\to (B_1, \mathcal{R}_1)$ and $f_2\colon (A_2, \mathcal{Q}_2) \circ\!\!\to (B_2, \mathcal{R}_2)$, we show that $f_1 \otimes f_2\colon (A_1 \otimes A_2, \mathcal{Q}_1 \otimes \mathcal{Q}_2) \circ\!\!\to (B_1 \otimes B_2, \mathcal{R}_1 \otimes \mathcal{R}_2)$ is uniform. Because of Lemma 3.8, it suffices to prove $(f_1 \otimes f_2) \circ q \in (R_1 \otimes R_2)^{\mathsf{w}}_{\mathcal{C}}$ for all $q\colon w \to A_1 \otimes A_2$ such that there are $q_1 \in (Q_1)^{\mathsf{w}}_{\mathcal{C}}$ and $q_2 \in (Q_2)^{\mathsf{w}}_{\mathcal{C}}$ with $q = (q_1, q_2)$. We get $(f_1 \otimes f_2) \circ q = (f_1 \otimes f_2) \circ (q_1, q_2) = (f_1 \circ q_1, f_2 \circ q_2)$ since f_1 and f_2 are strict. Hence, $(f_1 \otimes f_2) \circ q \in (R_1 \otimes R_2)^{\mathsf{w}}_{\mathcal{C}}$ follows by the uniformity of f_1 and f_2. Therefore, since the smash product is functorial on $DOM_\perp$, it is a functor on $SD_\perp$ as well.

Thus, for proving that *SD* has smash products, it remains to show that the functions $\mathsf{smash}\colon (A \times B, \mathcal{Q} \times \mathcal{R}) \to (A \otimes B, \mathcal{Q} \otimes \mathcal{R})$ and $\widetilde{f}\colon (A \otimes B, \mathcal{Q} \otimes \mathcal{R}) \circ\!\!\to (C, \mathcal{P})$ are uniform for every bistrict morphism $f\colon (A, \mathcal{Q}) \times (B, \mathcal{R}) \circ\!\!\to (C, \mathcal{P})$. The former is trivial, we focus on the latter.

Let $f\colon (A, \mathcal{Q}) \times (B, \mathcal{R}) \circ\!\!\to (C, \mathcal{P})$ be a bistrict morphism in *SD*, then $\widetilde{f} \circ (q, r) = \widetilde{f} \circ \mathsf{smash} \circ \langle q, r\rangle = f \circ \langle q, r\rangle \in P^{\mathsf{w}}_{\mathcal{C}}$ follows for all $q \in Q^{\mathsf{w}}_{\mathcal{C}}$ and $r \in R^{\mathsf{w}}_{\mathcal{C}}$. Because of Lemma 3.8, this entails that $\widetilde{f}$ is uniform.

The functions $\mathsf{up}\colon (A, \mathcal{Q}) \to (A_\perp, \mathcal{Q}_\perp)$ and $\mathsf{down}\colon (A_\perp, \mathcal{Q}_\perp) \to (A, \mathcal{Q})$ are obviously uniform and, if $f\colon (A, \mathcal{Q}) \to (B, \mathcal{R})$ is uniform, then so is $f_\perp\colon (A_\perp, \mathcal{Q}_\perp) \circ\!\!\to (B_\perp, \mathcal{R}_\perp)$. Thus, the category *SD* has liftings.

For proving that *SD* has separated sums satisfying the universal property as explained in the previous section it remains to prove that, for all $i = 1, \ldots n$ and for all morphisms $f_i\colon (A_i, \mathcal{Q}_i) \to (C, \mathcal{P})$ in *SD*, the functions $\mathsf{in}_i\colon (A_i, \mathcal{Q}_i) \to (\sum_{i=1}^n A_i, \sum_{i=1}^n \mathcal{Q}_i)$ and $[f_1, \ldots, f_n]\colon (\sum_{i=1}^n A_i, \sum_{i=1}^n \mathcal{Q}_i) \to (C, \mathcal{P})$ are uniform. The proof for this is analogous to the one for coalesced sums, see below. It also entails that separated sums on *SD* are given by a functor.

Let $f_1\colon (A_1, \mathcal{Q}_1) \circ\!\!\to (B_1, \mathcal{R}_1)$ and $f_2\colon (A_2, \mathcal{Q}_2) \circ\!\!\to (B_2, \mathcal{R}_2)$ be two morphisms in $SD_\perp$, we verify that $f_1 \oplus f_2\colon (A_1 \oplus A_2, \mathcal{Q}_1 \oplus \mathcal{Q}_2) \circ\!\!\to (B_1 \oplus B_2, \mathcal{R}_1 \oplus \mathcal{R}_2)$ is uniform. Suppose $p\colon w \to A_1 \oplus A_2$ is a function such that there are maps $q_i \in (Q_i)^{\mathsf{w}}_{\mathcal{C}}$ $(i = 1, 2)$ and $\varphi \in S_2$ with $p(x) = \begin{cases} \widetilde{\mathsf{inl}} \circ q_1(x) & \text{if } \varphi(x) = 1 \\ \widetilde{\mathsf{inr}} \circ q_2(x) & \text{if } \varphi(x) = 2 \\ \perp & \text{if } \varphi(x) = \perp \end{cases}$. Then $(f_1 \oplus f_2) \circ p = \begin{cases} \widetilde{\mathsf{inl}} \circ f_1 \circ q_1(x) & \text{if } \varphi(x) = 1 \\ \widetilde{\mathsf{inr}} \circ f_2 \circ q_2(x) & \text{if } \varphi(x) = 2 \\ \perp & \text{if } \varphi(x) = \perp \end{cases}$ follows, which implies $(f_1 \oplus f_2) \circ p \in (R_1 \oplus R_2)^{\mathsf{w}}_{\mathcal{C}}$. This shows the uniformness of $f_1 \oplus f_2$. Hence, since $\oplus$ is a functor on $DOM_\perp$, it is also functorial on $SD_\perp$.

To complete the proof that the category SD has coalesced sums it remains to verify that $\widetilde{\mathsf{inl}}\colon (A,\mathcal{Q}) \circ\!\!\rightarrow (A\oplus B, \mathcal{Q}\oplus\mathcal{R})$ and $\widetilde{\mathsf{inr}}\colon (B,\mathcal{R}) \circ\!\!\rightarrow (A\oplus B, \mathcal{Q}\oplus\mathcal{R})$ as well as $\widetilde{[f,g]}\colon (A\oplus B, \mathcal{Q}\oplus\mathcal{R}) \circ\!\!\rightarrow (C,\mathcal{P})$ for strict morphisms $f\colon (A,\mathcal{Q}) \circ\!\!\rightarrow (C,\mathcal{P})$ and $g\colon (B,\mathcal{R}) \circ\!\!\rightarrow (C,\mathcal{P})$ are uniform. The first two cases are trivial by instantiating the constant functions $\lambda x{:}w.1$ and $\lambda x{:}w.2$, respectively, for φ in the definition of $(Q\oplus R)^{\mathsf{w}}_{\mathcal{C}}$. For the $\widetilde{[\cdot,\cdot]}$-case suppose that $p\colon w\to A\oplus B$ is a map such that there exist functions $q\in Q^{\mathsf{w}}_{\mathcal{C}}$, $r\in R^{\mathsf{w}}_{\mathcal{C}}$ and $\varphi\in S_2$ with $p(x)=\begin{cases}\widetilde{\mathsf{inl}}\circ q(x) & \text{if } \varphi(x)=1\\ \widetilde{\mathsf{inr}}\circ r(x) & \text{if } \varphi(x)=2\\ \bot & \text{if } \varphi(x)=\bot\end{cases}$. Then $\widetilde{[f,g]}\circ p(x)=\begin{cases} f\circ q(x) & \text{if } \varphi(x)=1\\ f\circ r(x) & \text{if } \varphi(x)=2\\ \bot & \text{if } \varphi(x)=\bot\end{cases}$ follows. Because of $f\circ q, g\circ r\in P^{\mathsf{w}}_{\mathcal{C}}$, applying condition (R4) leads to $\widetilde{[f,g]}\circ p\in P^{\mathsf{w}}_{\mathcal{C}}$.

Finally, we check that the subcategory $SD_\bot$ is symmetric monoidal closed. The definition of the smash product functor is obviously symmetric. Given two strict morphisms $f_1\colon (B_1,\mathcal{R}_1)\circ\!\!\rightarrow (A_1,\mathcal{Q}_1)$ and $f_2\colon (A_2,\mathcal{Q}_2)\circ\!\!\rightarrow (B_2,\mathcal{R}_2)$, it is easy to prove that $f_1\circ\!\!\rightarrow f_2\colon ([A_1\circ\!\!\rightarrow A_2],[\mathcal{Q}_1\circ\!\!\rightarrow \mathcal{Q}_2])\circ\!\!\rightarrow ([B_1\circ\!\!\rightarrow B_2],[\mathcal{R}_1\circ\!\!\rightarrow \mathcal{R}_2])$ is uniform. Since the strict function space operator is a functor on $DOM_\bot$, it therefore is also functorial on $SD_\bot$. The uniformness of the strict evaluation function $\widetilde{\mathsf{eval}}$ follows directly from the universal property for smash products:

$$
\begin{tikzcd}
\big([C\circ\!\!\rightarrow B]\times C, [\mathcal{P}\circ\!\!\rightarrow\mathcal{R}]\times\mathcal{P}\big) \arrow[r, "\mathsf{smash}"] \arrow[dr, "\mathsf{eval}"'] & \big([C\circ\!\!\rightarrow B]\otimes C, [\mathcal{P}\circ\!\!\rightarrow\mathcal{R}]\otimes\mathcal{P}\big) \arrow[d, dotted, "\exists!\widetilde{\mathsf{eval}}"] \\
 & (B,\mathcal{R})
\end{tikzcd}
$$

Besides, the $\otimes$-operator in SD is clearly associative. Thus, the only thing which remains to be checked is that the Sierpiński domain $\mathbf{2}$ with its relations as defined above is neutral for the smash product operator.

Let $\mathcal{C}$ be a subcategory of $SPIC$, w be an object in $\mathcal{C}$ and $(A,\mathcal{Q})$ be an object in SD. Firstly, we prove $(Q\otimes\mathbf{2})^{\mathsf{w}}_{\mathcal{C}}\cong Q^{\mathsf{w}}_{\mathcal{C}}$. Clearly, the underlying domains $A\otimes\mathbf{2}$ and A are isomorphic. If q is a function related by $Q^{\mathsf{w}}_{\mathcal{C}}$, then the corresponding pair $(q,\lambda x{:}\mathbf{2}.\top_{\mathbf{2}})$ is related by $(Q\otimes\mathbf{2})^{\mathsf{w}}_{\mathcal{C}}$.

On the other hand, given $r\in (Q\otimes\mathbf{2})^{\mathsf{w}}_{\mathcal{C}}$, there are functions $q_i\in Q^{\mathsf{w}}_{\mathcal{C}}$ and $p_i\in \mathbf{2}^{\mathsf{w}}_{\mathcal{C}}$ with

$$
\begin{aligned}
r &= \lambda x{:}w.\bigsqcup_{i\in I}^{\uparrow}(q_i,p_i)(x)\\
&= \lambda x{:}w.\begin{cases}\Big(\bigsqcup^{\uparrow}_{i\in I}\big(q_i\restriction\{x\in w\mid p_i(x)=\top_{\mathbf{2}}\}\big)(x),\top_{\mathbf{2}}\Big) & \text{if } \exists i{\in}I.\ (q_i,p_i)(x)\neq\bot\\ \bot_{A\otimes\mathbf{2}} & \text{otherwise}\end{cases}\\
&= \bigsqcup_{i\in I}^{\uparrow}\big(q_i\restriction\{x\in w\mid p_i(x)=\top_{\mathbf{2}}\},\lambda x{:}w.\top_{\mathbf{2}}\big).
\end{aligned}
$$

Thus, the function $\bigsqcup^{\uparrow}_{i\in I}\left(q_i \restriction \{x \in w \mid p_i(x) = \top_{\mathbf{2}}\}\right) : w \to A$ that corresponds to r is related by $Q^{\mathsf{w}}_{\mathcal{C}}$ because $Q^{\mathsf{w}}_{\mathcal{C}}$ satisfies conditions (R1) and (R3). This completes the proof of Theorem 3.10. □

As in *DOM*, coalesced sum, smash product and strict function space operators are not functors on *SD*, but, as shown in the proof of Theorem 3.10, they are functorial on the subcategory $SD_\perp$. Moreover, since all embeddings and projections that occur in the construction of bilimits in $DOM_\perp$ are uniform (and therefore morphisms in $SD_\perp$), infinite constructions work on $SD_\perp$ in the same way as they do on $DOM_\perp$:

PROPOSITION 3.11. *Propositions 2.1, 2.2, 2.3 and 2.5 still hold if the category* $DOM_\perp$ *is replaced by* $SD_\perp$. □

3.4 The Subcategory of SSB-Objects

Some objects in *SD* allow to chop off the bottom element. This is formalised in the following definition:

DEFINITION 3.12. *An object* $(A, \mathcal{Q})$ *of the category SD is called* SSB-object *(for sequentially separable bottom) if there is a morphism* $\chi_A : (A, \mathcal{Q}) \to \mathbf{2}$ *in SD such that* $\chi_A(a) = \perp$ *holds if, and only if,* $a = \perp$ *holds. The full subcategory of all SSB-objects in SD is called SSB. As for DOM, we write* $SSB_\perp$ *and* SSB_{tot} *for the subcategory with the SSB-objects and strict functions and strict total functions, respectively, as morphisms.*

A couple of properties that are not valid in general are satisfied for SSB-objects. For example, the domain of definition of any relation yields a partial partition in each sequentiality system.

LEMMA 3.13. *If* $(A, \mathcal{Q})$ *is an SSB-object,* $\mathsf{w} = (w, S)$ *an object in an index category* $\mathcal{C}$ *and* $q : w \to A$ *a function with* $q \in Q^{\mathsf{w}}_{\mathcal{C}}$*, then* $\left\{\{x \in w \mid q(x) \neq \perp_A\}\right\}^{-} \in S$ *holds.*

Proof. Since $(A, \mathcal{Q})$ is an SSB-object, there is a morphism $\chi_A : (A, \mathcal{Q}) \to \mathbf{2}$ in *SD* with $\left\{\{x \in w \mid q(x) \neq \perp_A\}\right\}^{-} = \left\{\{x \in w \mid \chi_A \circ q(x) \neq \perp_{\mathbf{2}}\}\right\}^{-} \in S$ by definition of $\mathbf{2}^{\mathsf{w}}_{\mathcal{C}}$. □

If the domain of a morphism f is an SSB-object, the strictification $\mathsf{str}(f)$ of f is a morphism in *SD* as well:

LEMMA 3.14. *Let* $(A, \mathcal{Q})$ *be an SSB-object and* $f : (A, \mathcal{Q}) \to (B, \mathcal{R})$ *a morphism in SD. Then the strictification*

$$\mathsf{str}(f) \;=\; \lambda a{:}A. \begin{cases} f(a) & \text{if } a \neq \perp_A \\ \perp_B & \text{if } a = \perp_A \end{cases}$$

is a morphism in SD, too.

Proof. The continuity of $\mathsf{str}(f)$ follows from the continuity of f. It remains to show that $\mathsf{str}(f)$ is uniform. Given $q \in Q^{\mathsf{w}}_{\mathcal{C}}$ for an index category $\mathcal{C}$ and a world $\mathsf{w} = (w, S) \in \mathsf{Obj}(\mathcal{C})$, then $\mathsf{str}(f) \circ q = (f \restriction \{a \in A \mid a \neq \perp_A\}) \circ q = (f \circ q) \restriction \{x \in w \mid q(x) \neq \perp_A\}$ follows. Since $(A, \mathcal{Q})$ is an SSB-object, $\left\{\{x \in w \mid q(x) \neq \perp_A\}\right\}^{-} \in S$ follows from Lemma 3.13. Hence, condition (R3) implies $\mathsf{str}(f) \circ q \in R^{\mathsf{w}}_{\mathcal{C}}$. □

The question arises which constructions on SD preserve the SSB property.

PROPOSITION 3.15. *The following statements hold in SD:*

1. *If $(A, \mathcal{Q})$ is an arbitrary object, then $(A_\perp, \mathcal{Q}_\perp)$ is an SSB-object.*
2. *If $(A, \mathcal{Q})$ and $(B, \mathcal{R})$ are SSB-objects, then so are $(A \otimes B, \mathcal{Q} \otimes \mathcal{R})$ and $(A \oplus B, \mathcal{Q} \oplus \mathcal{R})$.*
3. *If $(A_1, \mathcal{Q}_1), \ldots, (A_n, \mathcal{Q}_n)$ are arbitrary objects, then $(\sum_{i=1}^n A_i, \sum_{i=1}^n \mathcal{Q}_i)$ is an SSB-object.*

Proof.

1. The morphism $\chi_{A_\perp} \colon (A_\perp, \mathcal{Q}_\perp) \to \mathbf{2}$ is given by $(\lambda x{:}A.\bot_{\mathbf{1}})_\perp$, which is uniform by Theorem 3.10.
2. The characteristic functions for the objects $(A \otimes B, \mathcal{Q} \otimes \mathcal{R})$ and $(A \oplus B, \mathcal{Q} \oplus \mathcal{R})$ in SD are given by $\chi_{A \otimes B} = (\lambda x{:}A{\otimes}B.\top_{\mathbf{2}}) \restriction \big(\chi_A^{-1}(\{\top_{\mathbf{2}}\}) \cap \chi_B^{-1}(\{\top_{\mathbf{2}}\})\big)$ and $\chi_{A \oplus B} = (\lambda x{:}A{\oplus}B.\top_{\mathbf{2}}) \restriction \big(\chi_A^{-1}(\{\top_{\mathbf{2}}\}) \cup \chi_B^{-1}(\{\top_{\mathbf{2}}\})\big)$. Besides, since $\chi_A^{-1}(\{\top_{\mathbf{2}}\})$ and $\chi_B^{-1}(\{\top_{\mathbf{2}}\})$ as well as their union and their intersection can be considered as the only class of a singleton partial partition in S, the claim follows from condition (R3).
3. Suppose $q \in (\sum_{i=1}^n \mathcal{Q}_i)^{\mathsf{w}}_{\mathcal{C}}$, then there is a function $\varphi \in S_n$ with $q(x) = \bot$ if, and only if, $\varphi(x)$ equals $\bot$. Therefore, the characteristic function for $(\sum_{i=1}^n A_i, \sum_{i=1}^n \mathcal{Q}_i)$ is given by $\chi_{\sum_{i=1}^n A_i}, = (\lambda x{:}\sum_{i=1}^n A_i.\top_{\mathbf{2}}) \restriction \big(\varphi^{-1}(\{1\}) \cup \cdots \cup \varphi^{-1}(\{n\})\big)$. It is a morphism in SD because of condition (R3). $\square$

Note that the cartesian product of SSB-objects does not necessarily satisfy the SSB property. For example, the sequential domain $\mathbf{2}\times\mathbf{2}$ with its induced family of relations is not an SSB-object. This can be shown by using Sieber's relation from page 38 again. Consider the relation $S_1 := R^{\{1,2,3\}}_{\{1,2\},\{1,2,3\}}$ on the domain $\mathbf{2}$, then $\big(\langle \top_{\mathbf{2}}, \bot_{\mathbf{2}}\rangle, \langle \bot_{\mathbf{2}}, \top_{\mathbf{2}}\rangle, \langle \bot_{\mathbf{2}}, \bot_{\mathbf{2}}\rangle\big)$ is related by $S_1 \times S_1$, but $\big(\chi_{\mathbf{2}\times\mathbf{2}}(\top_{\mathbf{2}}, \bot_{\mathbf{2}}), \chi_{\mathbf{2}\times\mathbf{2}}(\bot_{\mathbf{2}}, \top_{\mathbf{2}}), \chi_{\mathbf{2}\times\mathbf{2}}(\bot_{\mathbf{2}}, \bot_{\mathbf{2}})\big) = (\top_{\mathbf{2}}, \top_{\mathbf{2}}, \bot_{\mathbf{2}})$ is not in S_1. Hence, the function $\chi_{\mathbf{2}\times\mathbf{2}} \colon \mathbf{2}\times\mathbf{2} \to \mathbf{2}$ is not a morphism in SD, and, therefore, $\mathbf{2}\times\mathbf{2}$ is not an SSB-object.

However, the subclass of SSB-objects in SD is closed under infinite constructions:

PROPOSITION 3.16. *Let $F \colon SD_\perp^{op} \times SD_\perp \to SD_\perp$ be a locally continuous functor mapping SSB-objects to SSB-objects. Then its bilimit $\mathsf{fix}(F)$ as constructed in Proposition 2.1 is an SSB-object.*

Proof. Since all objects of the increasing sequence $\mathbf{1}, F(\mathbf{1},\mathbf{1}), F^{(2)}(\mathbf{1},\mathbf{1}), \ldots$ satisfy the SSB-property (where $F^{(n)}(\mathbf{1},\mathbf{1})$ is defined as in Proposition 2.1), they all have a bottom separating morphism $\chi_{F^{(n)}(\mathbf{1},\mathbf{1})} \colon F^{(n)}(\mathbf{1},\mathbf{1}) \to \mathbf{2}$. The characteristic function for the bilimit $\mathsf{fix}(F)$ is given by

$$\chi_{\mathsf{fix}(F)} = \bigsqcup_{n\in\mathbb{N}}^{\uparrow} \chi_{F^{(n)}(\mathbf{1},\mathbf{1})} \circ p_n \colon \mathsf{fix}(F) \to \mathbf{2}$$

(recall that $p_n \colon \mathsf{fix}(F) \to F^{(n)}(\mathbf{1},\mathbf{1})$ is the projection from the bilimit to its n-th component). It is straightforward to verify that the set of all $\chi_{F^{(n)}(\mathbf{1},\mathbf{1})} \circ p_n$ $(n \in \mathbb{N})$ is directed and that its supremum $\chi_{\mathsf{fix}(F)}$ maps an element $x \in \mathsf{fix}(F)$ to $\bot_{\mathbf{2}}$ if, and only if, $x = \bot_{\mathsf{fix}(F)}$ is true. Since SD is cpo-enriched, $\chi_{\mathsf{fix}(F)}$ is also a morphism in SD, and, hence, $\mathsf{fix}(F)$ is an SSB-object. $\square$

With the notion of SSB-types we are now able to show that, unlike the parallel-or function, left- and right-or are morphisms in SD. In order to verify the uniformity of the left-or lor, suppose $p = \langle p_1, p_2 \rangle \in (\mathbb{B}_\perp \times \mathbb{B}_\perp)^{\mathsf{w}}_{\mathcal{C}}$ for a world $\mathsf{w} = (w, S)$ in an index category $\mathcal{C}$ where $\mathbb{B}_\perp$ is the coalesced sum $\mathbf{2} \oplus \mathbf{2}$, it is an SSB-object by Proposition 3.15. Then $p_1, p_2 \in (\mathbb{B}_\perp)^{\mathsf{w}}_{\mathcal{C}}$ holds by Theorem 3.10, moreover, there are $\varphi_1, \varphi_2 \in S_2$, $q_1, q_2, r_1, r_2 \in \mathbf{2}^{\mathsf{w}}_{\mathcal{C}}$ with

$$p_1(x) \;=\; \begin{cases} \widetilde{\mathsf{inl}} \circ q_1(x) & \text{if } \varphi_1(x) = 1 \\ \widetilde{\mathsf{inr}} \circ r_1(x) & \text{if } \varphi_1(x) = 2 \\ \perp & \text{otherwise} \end{cases}, \qquad p_2(x) \;=\; \begin{cases} \widetilde{\mathsf{inl}} \circ q_2(x) & \text{if } \varphi_2(x) = 1 \\ \widetilde{\mathsf{inr}} \circ r_2(x) & \text{if } \varphi_2(x) = 2 \\ \perp & \text{otherwise} \end{cases}.$$

Due to Lemma 3.13, we assume without loss of generality that $\varphi_i(x)$ is bottom if, and only if, $p_i(x)$ is bottom (for $i = 1, 2$). Consider the partial partitions $\{v_1, u_1\}^-$, $\{v_2, u_2\}^- \in S$ corresponding to φ_1 and φ_2, respectively. Then $\{v_1,\, u_1 \cap v_2,\, u_1 \cap u_2\}^-$ and, therefore, $\{v_1 \cup (u_1 \cap v_2),\, u_1 \cap u_2\}^-$ are in S as well. Let $\varphi \in S_2$ be a function denoting the latter partial partition. Then

$$\mathsf{lor} \circ p(x) \;=\; \begin{cases} \widetilde{\mathsf{inl}}(\top) & \text{if } \varphi(x) = 1 \\ \widetilde{\mathsf{inr}}(\top) & \text{if } \varphi(x) = 2 \\ \perp & \text{otherwise} \end{cases}$$

holds, which implies $\mathsf{lor} \circ p \in (\mathbb{B}_\perp)^{\mathsf{w}}_{\mathcal{C}}$. Analogously, we can show $\mathsf{ror} \circ p \in (\mathbb{B}_\perp)^{\mathsf{w}}_{\mathcal{C}}$ for all $p \in (\mathbb{B}_\perp \times \mathbb{B}_\perp)^{\mathsf{w}}_{\mathcal{C}}$. Hence, the functions $\mathsf{lor}, \mathsf{ror} \colon (\mathbb{B}_\perp \times \mathbb{B}_\perp) \to \mathbb{B}_\perp$ are morphisms in SD.

SSB-objects play an important role in SD. For instance, the construction of smash products on SSB-objects does not require the closure operator ρ_1 as for arbitrary objects:

PROPOSITION 3.17. *Let $(A, \mathcal{Q}), (B, \mathcal{R})$ be SSB-objects and $\mathcal{C}$ be a subcategory of SPIC. Then the relations on the smash product of $(A, \mathcal{Q})$ and $(B, \mathcal{R})$ are given by:*

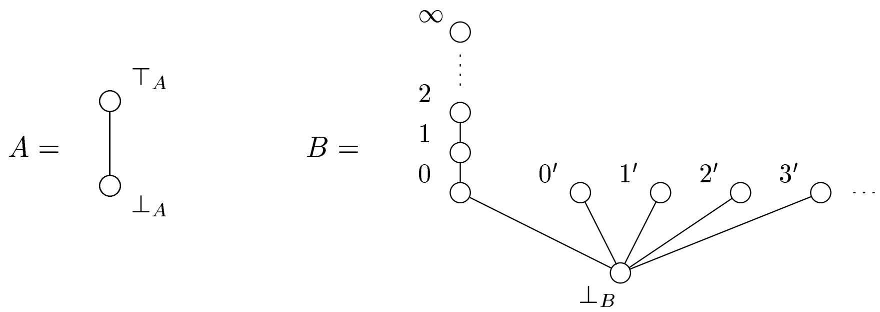
$$(Q \otimes R)^{\mathsf{w}}_{\mathcal{C}} \;=\; \big\{(q, r) \colon w \to A \otimes B \mid q \in Q^{\mathsf{w}}_{\mathcal{C}}, r \in R^{\mathsf{w}}_{\mathcal{C}}\big\},$$

for every object $\mathsf{w} = (w, S)$ in $\mathcal{C}$.

Proof. We prove that the relation in the proposition satisfies the admissibility condition (R1) if $Q^{\mathsf{w}}_{\mathcal{C}}$ and $R^{\mathsf{w}}_{\mathcal{C}}$ do so.

Let $\big((q_i, r_i) \colon w \to (A \otimes B)\big)_{i \in I}$ be a directed family of relations with $q_i \in Q^{\mathsf{w}}_{\mathcal{C}}$ and $r_i \in R^{\mathsf{w}}_{\mathcal{C}}$ for all $i \in I$. We have to show that $\bigsqcup^{\uparrow}{}_{i \in I}(q_i, r_i) \in (Q \otimes R)^{\mathsf{w}}_{\mathcal{C}}$ holds. Define a set $v := \{x \in w \mid \exists i {\in} I.\ (q_i, r_i)(x) \neq \perp\}$, we prove that $\{v\}^-$ is a partial partition in S. Since $(A, \mathcal{Q})$ and $(B, \mathcal{R})$ are SSB-objects, there are bottom separating morphisms $\chi_A \colon (A, \mathcal{Q}) \to \mathbf{2}$ and $\chi_B \colon (B, \mathcal{R}) \to \mathbf{2}$ with $\chi_A \circ q_i,\ \chi_B \circ r_i \in \mathbf{2}^{\mathsf{w}}_{\mathcal{C}}$. This implies $(\chi_A \circ q_i, \chi_B \circ r_i) \in (\mathbf{2} \otimes \mathbf{2})^{\mathsf{w}}_{\mathcal{C}} \cong \mathbf{2}^{\mathsf{w}}_{\mathcal{C}}$ for all $i \in I$. Because the set of all those pairs is directed and because w is finite, there is an $i_0 \in I$ with $\bigsqcup^{\uparrow}{}_{i \in I}(\chi_A \circ q_i, \chi_B \circ r_i) = (\chi_A \circ q_{i_0}, \chi_B \circ r_{i_0})$. Thus, $\{v\}^- = \big\{\{x \in w \mid (\chi_A \circ q_{i_0}, \chi_B \circ r_{i_0})(x) \neq \perp\}\big\}^- \in S$ follows. Applying condition (R3) for $Q^{\mathsf{w}}_{\mathcal{C}}$ and $R^{\mathsf{w}}_{\mathcal{C}}$ yields $q_i \restriction v \in Q^{\mathsf{w}}_{\mathcal{C}}$ and $r_i \restriction v \in R^{\mathsf{w}}_{\mathcal{C}}$. Because $\bigsqcup^{\uparrow}{}_{i \geq i_0} q_i \restriction v \in Q^{\mathsf{w}}_{\mathcal{C}}$ and $\bigsqcup^{\uparrow}{}_{i \geq i_0} r_i \restriction v \in R^{\mathsf{w}}_{\mathcal{C}}$ hold by induction hypothesis, this finally leads to $\big(\bigsqcup^{\uparrow}{}_{i \geq i_0} q_i \restriction v, \bigsqcup^{\uparrow}{}_{i \geq i_0} r_i \restriction v\big) \in \{(q, r) \colon w \to A \otimes B \mid q \in Q^{\mathsf{w}}_{\mathcal{C}}, r \in R^{\mathsf{w}}_{\mathcal{C}}\}$. It can be easily checked that this is the supremum $\bigsqcup^{\uparrow}{}_{i \in I}(q_i, r_i)$. $\square$

Unlike for SSB-objects, the relation $\mathcal{Q} \otimes \mathcal{R}$ on the smash product of arbitrary objects $(A, \mathcal{Q})$ and $(B, \mathcal{R})$ is generally not given by the componentwise constructed relation $P^{\mathsf{w}}_{\mathcal{C}} := \{(q,r)\colon w \to A \otimes B \mid q \in Q^{\mathsf{w}}_{\mathcal{C}}, r \in R^{\mathsf{w}}_{\mathcal{C}}\}$. The following counter-example is a modification of a family of relations used in [OR95a]. Let $\mathcal{C}$ be the subcategory of *SPIC* that consists of one single object $\mathsf{w} = (w, S)$ with $w = \{1, 2\}$ and $S = \big\{\varnothing, \{w\}\big\}$. The underlying domains of two *SD*-objects are defined by $A := \{\bot_A, \top_A\}$ and $B := (\mathbb{N}^\infty)_\bot \oplus \mathbb{N}_\bot$ (where $\mathbb{N}^\infty$ denotes the domain of natural numbers with the natural order plus a top element ∞ and $\mathbb{N}_\bot$ stands for the flat domain of natural numbers). To avoid confusion, the elements of $\mathbb{N}^\infty$ are denoted by $0, 1, 2, \dots, \infty$ and the ones of $\mathbb{N}_\bot$ by $0', 1', 2', \dots$.



Consider the following binary relations on A and B:

$$Q^{\mathsf{w}}_{\mathcal{C}} \;:=\; \{(\top_A, \bot_A)\} \cup \{(a, a) \mid a \in A\}$$

and

$$R^{\mathsf{w}}_{\mathcal{C}} \;:=\; \{(n, n') \mid n \in \mathbb{N}\} \cup \{(b, b) \mid b \in B\}.$$

They both satisfy conditions (R1)–(R4). Hence, since w is the only object in $\mathcal{C}$, they form $\mathcal{C}$-Kripke logical relations on A and B. By definition of the smash product and the relation $P^{\mathsf{w}}_{\mathcal{C}}$,

$$\big((\top_A, n), (\bot_A, n')\big) = \big((\top_A, n), (\bot_A, \bot_B)\big) \in P^{\mathsf{w}}_{\mathcal{C}}$$

is true for every natural number $n \in \mathbb{N}$. On the other hand, the directed supremum $\bigsqcup^{\uparrow}{}_{n \in \mathbb{N}} \big((\top_A, n), (\bot_A, n')\big) = \big((\top_A, \infty), (\bot_A, \bot_B)\big)$ is not related by $P^{\mathsf{w}}_{\mathcal{C}}$.

Note that $(A, \mathcal{Q})$ is not the Sierpiński sequential domain, since the families $Q_{\mathcal{C}}$ and $\mathbf{2}_{\mathcal{C}}$ of $\mathcal{C}$-Kripke relations are different. In particular, $(A, \mathcal{Q})$ is not even an SSB-object. This is a crucial presumption for the counter-example. Otherwise, there would be a morphism $\chi_A\colon (A, \mathcal{Q}) \to \mathbf{2}$ with $\big(\chi_A(\top_A), \chi_A(\bot_A)\big) = (\top_{\mathbf{2}}, \bot_{\mathbf{2}}) \in \mathbf{2}^{\mathsf{w}}_{\mathcal{C}}$. However, this would imply $\big\{\{1\}\big\} \in S$, which contradicts the definition of w.

Smash products on SSB-objects satisfy another interesting property:

PROPOSITION 3.18. *Let $(A, \mathcal{Q})$ and $(B, \mathcal{R})$ be SSB-objects in SD. Then the two projections $\widetilde{\mathsf{pr}_1}\colon (A \otimes B, \mathcal{Q} \otimes \mathcal{R}) \to (A, \mathcal{Q})$ and $\widetilde{\mathsf{pr}_2}\colon (A \otimes B, \mathcal{Q} \otimes \mathcal{R}) \to (B, \mathcal{R})$ exist in SD.*

Proof. We have to check that the projections given by $\widetilde{\mathsf{pr}_1}(a, b) = \begin{cases} a & \text{if } b \neq \bot \\ \bot & \text{if } b = \bot \end{cases}$ (and analogously for $\widetilde{\mathsf{pr}_2}$) are uniform. Because of Proposition 3.17, the elements of $(Q \otimes R)^{\mathsf{w}}_{\mathcal{C}}$ are given by pairs (q,r) with $q \in Q^{\mathsf{w}}_{\mathcal{C}}$ and $r \in R^{\mathsf{w}}_{\mathcal{C}}$. Hence, $\widetilde{\mathsf{pr}_1} \circ (q,r) = q \restriction \{x \in w \mid r(x) \neq \bot_B\}$ holds. Since B is an SSB-object, $\big\{\{x \in w \mid r(x) \neq \bot_B\}\big\}^{-} = \big\{\{x \in w \mid \chi_B \circ r(x) \neq \bot_{\mathbf{2}}\}\big\}^{-} \in S$ follows by Lemma 3.13. Thus, condition (R3) implies $\widetilde{\mathsf{pr}_1} \circ (q,r) \in Q^{\mathsf{w}}_{\mathcal{C}}$. The proof for $\widetilde{\mathsf{pr}_2}$ is analogous. $\square$

Projections for smash products of non-SSB-objects usually do not exist: because of $(A,\mathcal{Q}) \otimes \mathbf{2} \cong (A,\mathcal{Q})$, every projection $\widetilde{\mathsf{pr}_2}\colon (A,\mathcal{Q}) \otimes \mathbf{2} \to \mathbf{2}$ can be considered as a characteristic function $\chi_A\colon (A,\mathcal{Q}) \to \mathbf{2}$ (analogously for $\widetilde{\mathsf{pr}_1}$).

As a consequence of the previous results, we can find isomorphisms between some objects in *SD* in the same way as in *DOM*:

PROPOSITION 3.19. *In SD, the correspondences*

$$\begin{array}{rrcl} & \left(\sum_{i=1}^{n} A_i, \sum_{i=1}^{n} \mathcal{Q}_i\right) & \cong & \big((A_1)_\bot \oplus \cdots \oplus (A_n)_\bot, (\mathcal{Q}_1)_\bot \oplus \cdots \oplus (\mathcal{Q}_n)_\bot\big), \\ & \big((A \times B)_\bot, (\mathcal{Q} \times \mathcal{R})_\bot\big) & \cong & \big(A_\bot \otimes B_\bot, \mathcal{Q}_\bot \otimes \mathcal{R}_\bot\big) \\ \textit{and} & ([A \to B], [\mathcal{Q} \to \mathcal{R}]) & \cong & ([A_\bot \circ\!\!\to B], [\mathcal{Q}_\bot \circ\!\!\to \mathcal{R}]) \end{array}$$

hold.

Proof. The underlying domains are isomorphic in *DOM*, we have to check whether the relations on them are the same.

The isomorphisms on the underlying domains are given by:

$$\begin{array}{rcl} \iota_1 & = & [\widetilde{\mathsf{inl}}^{n-1} \circ \mathsf{up}, \widetilde{\mathsf{inr}} \circ \widetilde{\mathsf{inl}}^{n-2} \circ \mathsf{up}, \ldots, \widetilde{\mathsf{inr}} \circ \widetilde{\mathsf{inl}} \circ \mathsf{up}, \widetilde{\mathsf{inr}} \circ \mathsf{up}]\colon \sum_{i=1}^{n} A_i \to \bigoplus_{i=1}^{n} (A_i)_\bot, \\ \iota_1^{-1} & = & g_n\colon (A_1)_\bot \oplus \cdots \oplus (A_n)_\bot \to \sum_{i=1}^{n} A_i \\ & & \quad \text{with } g_2 := \widetilde{[f_1, f_2]},\ g_i := \widetilde{[g_{i-1}, f_i]},\ f_i := (\mathsf{in}_i \circ \mathsf{down}) \restriction \{x \in (A_i)_\bot \mid x \neq \bot\}; \\ \iota_2 & = & (\mathsf{up} \circ \mathsf{pr}_1 \circ \mathsf{down}, \mathsf{up} \circ \mathsf{pr}_2 \circ \mathsf{down}) \restriction \{x \mid x \neq \bot\}\colon (A \times B)_\bot \to A_\bot \otimes B_\bot, \\ \iota_2^{-1} & = & \big(\mathsf{up} \circ \langle \mathsf{down} \circ \widetilde{\mathsf{pr}_1}, \mathsf{down} \circ \widetilde{\mathsf{pr}_2}\rangle\big) \restriction \{x \in A_\bot \otimes B_\bot \mid x \neq \bot\}\colon A_\bot \otimes B_\bot \to (A \times B); \\ \iota_3 & = & (\cdot \circ \mathsf{down}) \restriction \{x \in A_\bot \mid x \neq \bot\}\colon [A \to B] \to [A_\bot \circ\!\!\to B], \\ \iota_3^{-1} & = & \cdot \circ \mathsf{up}\colon [A_\bot \circ\!\!\to B] \to [A \to B]. \end{array}$$

By Proposition 3.15, $\big((A\times B)_\bot, (\mathcal{Q}\times\mathcal{R})_\bot\big)$, $(A_\bot, \mathcal{Q}_\bot)$, $(B_\bot, \mathcal{R}_\bot)$ and $(A_\bot \otimes B_\bot, \mathcal{Q}_\bot \otimes \mathcal{R}_\bot)$ are all SSB-objects. Therefore, due to Lemma 3.14 and Proposition 3.18, all isomorphisms ι_1, ι_2, ι_3, ι_1^{-1}, ι_2^{-1} and ι_3^{-1} are also morphisms in *SD*. This completes the proof. $\square$

3.5 Closure Properties of *SD*

We now focus on subcategories of *SD* that are closed under the domain theoretic constructions as considered above. Unfortunately, it is not clear whether the uniform (strict) function space of two bounded complete objects is again bounded complete. The problem is that the suprema in $[A \to B]_{SD}$ are different from the ones in $[A \to B]_{DOM}$. For example, in DOM, the supremum $\mathsf{lor} \sqcup \mathsf{ror}$ of the two or functions is given by por, which does not exist in SD (for the bounded completeness it does not need to as lor and ror are not bounded in SD, in fact, $[\mathbb{B}_\perp \times \mathbb{B}_\perp \to \mathbb{B}_\perp]_{SD}$ is bounded complete). In order to take the Kripke logical relations into account, notions as 'SFP-domain' and 'Scott domain' need to be modified:

DEFINITION 3.20. *A sequential domain $(A, \mathcal{Q})$ is called* uniformly SFP-domain *if there is an increasing sequence of uniform projections $\pi_n \colon D \to D$ (i. e. $\pi_n \circ \pi_n = \pi_n \sqsubseteq \mathsf{id}_D$ holds for all $n \in \mathbb{N}$) with finite image such that $\mathsf{id}_D = \bigsqcup^{\uparrow}_{n \in \mathbb{N}} \pi_n$ holds. A uniformly SFP-domain is called* uniformly Scott domain *if it has a uniform binary infimum operation $\sqcap \colon (A, \mathcal{Q}) \times (A, \mathcal{Q}) \to (A, \mathcal{Q})$.*

Obviously, the underlying domain of a uniformly SFP-domain is an SFP-domain and, due to Proposition 2.11, the underlying domain of a uniformly Scott domain is a Scott domain.

The full subcategory of uniformly SFP-domains of SD is indeed cartesian closed and also closed under bilimits. All constructions except function spaces are the same as in DOM, so it remains to show that, for SFP-domains A and B, the set of all (strict) uniform functions from A to B is again a uniformly SFP-domain. This can be done in the same way as in DOM since the projections as defined on page 28 are clearly uniform.

Analogously to Corollary 2.13, the following proposition classifies the closure properties of the subcategory of uniformly Scott domains. Besides, it gives a handy tool to verify that the category of type interpretations of a programming language consists of (uniformly) Scott domains.

PROPOSITION 3.21. *The full subcategory of uniformly Scott domains of SD is closed under cartesian products, smash products, liftings, coalesced sums, smash sums, exponentials, strict function spaces and bilimits.*

Proof. Since the subcategory of uniformly SFP-domains in SD is closed under all constructions as required in the proposition, it remains to show that, for uniformly Scott domains $(A, \mathcal{Q})$, $(B, \mathcal{R})$, the objects $(A \times B, \mathcal{Q} \times \mathcal{R})$, $([A \to B], [\mathcal{Q} \to \mathcal{R}])$ etc. have a uniform binary infimum operation. We just look at some constructions, the other cases can be proved analogously. Note that the continuity of the infimum operation follows from Lemma 2.12.

Liftings: Let $(A, \mathcal{Q})$ be a uniformly Scott domain. Given $p, q \in (Q_\perp)^{\mathsf{w}}_{\mathcal{C}}$ for a world $\mathsf{w} = (w, S)$ in an index category $\mathcal{C}$, then the infimum $p \sqcap_{A_\perp} q$ exists as well. Moreover, $\big\{\{x \in w \mid p(x) \neq \perp\}\big\}^-, \big\{\{x \in w \mid q(x) \neq \perp\}\big\}^- \in S$ holds, this implies $\big\{\{x \mid (p \sqcap q)(x) \neq \perp\}\big\}^- = \big\{\{x \mid p(x) \neq \perp\} \cap \{x \mid q(x) \neq \perp\}\big\}^- \in S$ by applying condition (P4). Furthermore, $\mathsf{down} \circ p, \mathsf{down} \circ q \in Q^{\mathsf{w}}_{\mathcal{C}}$ is true, which entails $\mathsf{down} \circ (p \sqcap_{A_\perp} q) = (\mathsf{down} \circ p) \sqcap_A (\mathsf{down} \circ q) \in Q^{\mathsf{w}}_{\mathcal{C}}$ by assumption. This finally proves $p \sqcap_{A_\perp} q \in Q^{\mathsf{w}}_{\mathcal{C}}$.

Smash sums: Assume that $(A, \mathcal{Q})$ and $(B, \mathcal{R})$ are uniformly Scott domains. Moreover, suppose $p, q \in (Q \oplus R)^{\mathsf{w}}_{\mathcal{C}}$ for a world w in an index category $\mathcal{C}$. Due to Proposition 3.17, there are φ_p, $\varphi_q \in S_2$, $p_1, q_1 \in Q^{\mathsf{w}}_{\mathcal{C}}$ and $p_2, q_2 \in R^{\mathsf{w}}_{\mathcal{C}}$ with

$$p(x) = \begin{cases} \widetilde{\mathsf{inl}} \circ p_1(x) & \text{if } \varphi_p(x) = 1 \\ \widetilde{\mathsf{inr}} \circ p_2(x) & \text{if } \varphi_p(x) = 2 \\ \bot & \text{if } \varphi_p(x) = \bot \end{cases} \quad \text{and} \quad q(x) = \begin{cases} \widetilde{\mathsf{inl}} \circ q_1(x) & \text{if } \varphi_q(x) = 1 \\ \widetilde{\mathsf{inr}} \circ q_2(x) & \text{if } \varphi_q(x) = 2 \\ \bot & \text{if } \varphi_q(x) = \bot \end{cases}.$$

Furthermore, as A and B have uniform infimum operations, $p_1 \sqcap_A q_1 \in Q^{\mathsf{w}}_{\mathcal{C}}$ and $p_2 \sqcap_B q_2 \in R^{\mathsf{w}}_{\mathcal{C}}$ follow. Choosing $\varphi := \varphi_p \sqcap \varphi_q$ leads to

$$(p \sqcap_{[A \oplus B]} q)(x) = \begin{cases} \widetilde{\mathsf{inl}} \circ (p_1 \sqcap_A q_1)(x) & \text{if } \varphi(x) = 1 \\ \widetilde{\mathsf{inr}} \circ (p_2 \sqcap_B q_2)(x) & \text{if } \varphi(x) = 2 \\ \bot & \text{if } \varphi(x) = \bot, \end{cases}$$

it is easy to verify $\varphi \in S_2$. This proves $p \sqcap_{[A \oplus B]} q \in (Q \oplus R)^{\mathsf{w}}_{\mathcal{C}}$.

Function spaces: We first show that uniform function spaces have binary infima. Let $(A, \mathcal{Q})$ and $(B, \mathcal{R})$ be uniformly Scott domains and assume $f, g \in [A \to B]$. We show that $f \sqcap_{[A \to B]} g := \lambda a{:}A.f(a) \sqcap_B g(a)$ is uniform. For this, assume $q \in Q^{\mathsf{w}}_{\mathcal{C}}$ for a world w in an index category $\mathcal{C}$. Then $\langle f \circ q, g \circ q \rangle \in (R \times R)^{\mathsf{w}}_{\mathcal{C}}$ follows from the uniformity of f and g. Since $\sqcap_B : B \times B \to B$ is uniform by assumption, this implies

$$\begin{aligned} (f \sqcap_{[A \to B]} g) \circ q &= \lambda x{:}w.(f \sqcap_{[A \to B]} g)\big(q(x)\big) \\ &= \big(\lambda x{:}w.(f \circ q)(x)\big) \sqcap_B \big(\lambda x{:}w.(g \circ q)(x)\big) \\ &= (f \circ q) \sqcap_B (g \circ q) \in R^{\mathsf{w}}_{\mathcal{C}}, \end{aligned}$$

which entails the uniformity of $f \sqcap_{[A \to B]} g$.

Now we prove that this infimum operation $\sqcap_{[A \to B]} : [A \to B] \to [A \to B]$ is uniform. Suppose $p, r \in (Q \to R)^{\mathsf{w}}_{\mathcal{C}}$, $v \in \mathsf{Obj}(\mathcal{C})$, $\mu : \mathsf{v} \xrightarrow{\mathcal{C}} \mathsf{w}$ and $q \in Q^{\mathsf{v}}_{\mathcal{C}}$. Then

$$\lambda x{:}v.\Big(\big((p \sqcap r) \circ \mu\big)(x)\Big)\big(q(x)\big) = \Big(\lambda x{:}v.\big((p \circ \mu)(x)\big)\big(q(x)\big)\Big) \sqcap \Big(\lambda x{:}v.\big((r \circ \mu)(x)\big)\big(q(x)\big)\Big)$$

is in $R^{\mathsf{v}}_{\mathcal{C}}$ as $\sqcap_B$ is uniform.

Bilimits: Let $(A_0, \mathcal{Q}_0)$, $(A_1, \mathcal{Q}_1)$, $(A_2, \mathcal{Q}_2), \ldots$ be an increasing sequence of uniformly Scott domains. For a world w in an index category $\mathcal{C}$, the relations $Q^{\mathsf{w}}_{\mathcal{C}}$ of the bilimit $(A, \mathcal{Q})$ are computed componentwise, thus, A is a uniformly Scott domain as well.

$\square$

3.6 Reconstructing $SD_\bot$ as a Category of Algebras

The category $SD_\bot$ can be constructed from its SSB-objects. As A. Simpson pointed out, it is the Eilenberg-Moore category of algebras in SSB_{tot} with respect to the lifting monad:

PROPOSITION 3.22. *The categories $(SSB_{tot})^\bot$ and $SD_\bot$ are equivalent.*

Proof. The equivalence of the two categories $(SSB_{tot})^{\perp}$ and $SD_{\perp}$ is proved in the same way as Proposition 2.14. We construct a functor $F\colon (SSB_{tot})^{\perp} \to SD_{\perp}$ that is full, faithful and (up to isomorphisms) surjective on objects. Let F be defined by

$$F(a) := \bigl(A \setminus \{\perp_A\}, \mathcal{Q}'\bigr)$$

for all algebras $a\colon (A_\perp, \mathcal{Q}_\perp) \circ\!\!\to_t (A, \mathcal{Q})$ where $\mathcal{Q}'$ is given by

$$(Q')^{\mathsf{w}}_{\mathcal{C}} \;:=\; \bigl\{q\colon w \to A \setminus \{\perp_A\} \mid \mathsf{str}(\mathsf{up})_{A\setminus\{\perp_A\}} \circ q \in Q^{\mathsf{w}}_{\mathcal{C}}\bigr\}$$

for every subcategory $\mathcal{C}$ of *SPIC* and every object $\mathsf{w} \in \mathsf{Obj}(\mathcal{C})$. The action on morphisms is the same as in the proof of Proposition 2.14:

$$\begin{tikzcd}
(A_\perp, \mathcal{Q}_\perp) \arrow[r, "a"] \arrow[d, "f_\perp"'] & (A, \mathcal{Q}) \arrow[d, "f"] & & \bigl(A \setminus \{\perp_A\}, \mathcal{Q}'\bigr) \arrow[d, "f^{\downarrow}"] \\
(B_\perp, \mathcal{R}_\perp) \arrow[r, "b"'] & (B, \mathcal{R}) & \text{is mapped to} & \bigl(B \setminus \{\perp_B\}, \mathcal{R}'\bigr)
\end{tikzcd}$$

As in DOM_{tot}, the underlying domain A of the codomain $(A, \mathcal{Q})$ of an algebra $a\colon (A_\perp, \mathcal{Q}_\perp) \circ\!\!\to_t (A, \mathcal{Q})$ has a unique atom, hence, $A \setminus \perp_A$ is indeed a domain. It is straightforward to verify that $(Q')^{\mathsf{w}}_{\mathcal{C}}$ satisfies conditions (R1)–(R4) for all subcategories $\mathcal{C}$ of *SPIC* and all objects $\mathsf{w} \in \mathsf{Obj}(\mathcal{C})$. Note, however, that the strictness of $\mathsf{str}(\mathsf{up})$ in the definition of the relations is crucial for the verification of (R3) and (R4). It is easily checked that $Q_{\mathcal{C}}$ is $\mathcal{C}$-Kripke monotone, thus, $F(a)$ is an object in $SD_\perp$.

In order to verify that F maps morphisms in $(SSB_{tot})^{\perp}$ to morphisms in $SD_\perp$, we need to show that, for all morphisms $f\colon (A, \mathcal{Q}) \circ\!\!\to_t (B, \mathcal{R})$ in $(SSB_{tot})^{\perp}$, the function $f^{\downarrow}\colon (A \setminus \{\perp_A\}, \mathcal{Q}') \to (B \setminus \{\perp_B\}, \mathcal{R}')$ is uniform. Assume $q \in (Q')^{\mathsf{w}}_{\mathcal{C}}$. Since f is uniform, $\mathsf{str}(\mathsf{up})_{B\setminus\{\perp_B\}} \circ f^{\downarrow} \circ q = f \circ \mathsf{str}(\mathsf{up})_{A\setminus\{\perp_A\}} \circ q \in R^{\mathsf{w}}_{\mathcal{C}}$ follows. This proves that $f^{\downarrow}$ is uniform as well.

It is straightforward to prove that F is full, faithful and (up to isomorphism) surjective on objects, which finishes the proof. $\square$

It is widely accepted that the category of predomains with partial functions is in some sense the most natural category that occurs in Denotational Semantics, see e. g. [Plo85]. It is more primitive than the category of domains with a least element and provides a model for call-by-value languages such as FPC. Moreover, $DOM_\perp$ can be recovered as the category of algebras over predomains, this is essentially the claim of Proposition 2.14.

The category SSB_{tot}, which can be considered as a modified category of predomains, does not enjoy nice closure properties itself, however, it contrives a reconstruction of $SD_\perp$ as a category of algebras over a suitable lifting monad via Proposition 3.22. Nevertheless, SSB_{tot} is already defined as a subcategory of $SD_\perp$ and it is not clear how to present SSB_{tot} as a primitive category.

4 The General Sequential Language SFL

Before we focus on the functional language SFL, the main subject of this section, we start with some general considerations.

4.1 Some Remarks on Evaluation Strategies

There are two kinds of operational semantics, one is based on a *call-by-value* evaluation strategy, the other one follows the *call-by-name* paradigm. Furthermore, programming languages are sometimes characterised according to their *lazy* or *eager* behavior. Unfortunately, these concepts are understood by different people in different ways.

Call-by-Value versus Call-by-Name

The easiest way to illustrate the difference between 'call-by-value' and 'call-by-name' is to look at the evaluation rules for applications:

Call-by-value:

$$\frac{M \Downarrow \lambda x.M' \qquad N \Downarrow V_1 \qquad M'[V_1/x] \Downarrow V_2}{MN \Downarrow V_2}$$

Call-by-name:

$$\frac{M \Downarrow \lambda x.M' \qquad M'[N/x] \Downarrow V}{MN \Downarrow V}$$

In order to evaluate the application term MN, the call-by-value strategy as well as the call-by-name strategy start the computation by evaluating the first argument M. The difference of the two evaluation disciplines arises when this computation is finished and M is identified as a function $\lambda x.M'$. Next, according to the call-by-value paradigm, N has to be evaluated to a value V_1 before this value is plugged into M'. In a call-by-name language, however, N is treated as a black box until the evaluation of $M'[N/x]$ calls for the value of N. For instance, if M is a program defining a constant function and Ω a diverging program, then $M\Omega$ terminates according to a call-by-name strategy, but diverges with respect to a call-by-value strategy.

This observation has a consequence for the structure of the denotational semantics for a language. In order to interpret functional programs in a denotational model, one normally considers terms-in-contexts of the form

$$\Gamma \vdash M : \tau$$

where the *context* Γ is a list of all variables occurring freely in M. Such a term-in-context is interpreted as a morphism

$$[\![\Gamma \vdash M : \tau]\!] \colon [\![\Gamma]\!] \to [\![\tau]\!]$$

where $[\![\Gamma]\!]$ is the product of type denotations of the variables in Γ. This works fine for call-by-name languages. However, in a call-by-value language where programs that define functions are supposed to be evaluated strictly, interpretations $[\![\Gamma \vdash M : \tau]\!] \colon [\![\Gamma]\!] \to [\![\tau]\!]$ of terms-in-contexts ought to be strict functions as well, more precisely, they should be strict in all arguments. Hence, $[\![\Gamma]\!]$ must be the smash product in the model as $[\![\Gamma \vdash M : \tau]\!](\vec{x})$ is undefined whenever already one of its arguments is undefined (recall that an element $(a_1, \ldots, a_n) \in A_1 \otimes \cdots \otimes A_n$ is bottom if, and only if, one of the a_i's is bottom).

Usually, the operational semantics for an arbitrary programming language follows either the call-by-value or the call-by-name paradigm. Hence, it is normally clear whether terms-in-contexts are interpreted as strict functions with a smash product as domain of definition or as not necessarily strict functions with cartesian products as domains of definition. In this thesis, however, we are studying languages encompassing both paradigms of evaluation strategies. Concretely, although the language SFL presented in this section has call-by-value functions as well as call-by-name functions, it should be considered as a call-by-name language due to the interpretations of SFL terms-in-context. Accordingly, the language SFPL presented in Section 6 is treated as a call-by-value language although it also supports call-by-name type constructions.

Lazy versus Eager

When people working in the theory of programming languages talk about evaluation strategies, they often use the attributes *lazy* and *eager*. What do they mean? Unfortunately, there seem to be at least four different notions of laziness and eagerness:

- For some people, the expression 'eager' is a synonym for call-by-value and 'lazy' for call-by-name.

- Some other people use 'eager' for call-by-value, too, however, 'lazy' refers to a *call-by-need* strategy. This describes an efficient implementation technique where an application such as MN is evaluated call-by-name like, however, the term N would not be evaluated more than once as the result of its first computation is stored and can be reused.

- Thirdly, 'lazy' sometimes stands for *weak* evaluation strategies where programs such as $\lambda x.M$ are already normal forms, whereas an 'eager' strategy would continue evaluating the subterm M. In this sense, most 'real' programming languages are considered as lazy languages.

- The fourth notion is denotationally motivated. In 'lazy' languages one wants to distinguish between the programs $\lambda x.\Omega$ and Ω, hence, unlike in 'eager' languages, interpretations of functions spaces are lifted. Due to this notion, laziness is already forced in call-by-value languages where programs are considered to be equal if their observational behaviour is equal. For instance, the programs Ω and $\lambda x.\Omega$ can be observed to be different as $(\lambda x.*)\Omega$ (where $*$ is a constant) diverges whereas $(\lambda x.*)(\lambda x.\Omega)$ terminates.

As a consequence of this overloading of notation, we do not use these expressions at all.

4.2 The Syntax of SFL

The language SFL (sequential functional language) can be considered as a general meta-language for sequential computation, subsuming languages such as PCF and FPC. It is a call-by-name language that also supports a couple of call-by-value constructions. Furthermore, SFL is interesting not only from an operational point of view. As it contains type operators for all domain theoretic constructions occurring in Theorem 3.10 (although some of them are not defined on all types), it provides a suitable description of the category *SD* of sequential domains.

SFL is a language that is designed in accordance with the domain theoretic structure of a cartesian closed category of domains. In particular, product types should be interpreted as products in the category. Consequently, unlike in call-by-value languages, the terms Ω and $\langle\Omega,\Omega\rangle$ should not be observed to be different. However, it is essential that $\langle\Omega,\Omega\rangle$ evaluates to itself: since we have a call-by-name language, it must be possible to evaluate just one of components of a pair $\langle M,N\rangle$ without checking the other one as this may be a diverging term. Using a sequential evaluation strategy, we do not know whether to compute the left or the right component first. Thus, every evaluation of a pair of two terms must terminate, in particular, so must $\langle\Omega,\Omega\rangle$. This leaves a serious problem. On the one hand, the term Ω diverges, on the other hand, $\langle\Omega,\Omega\rangle$ evaluates to itself. However, as the $\perp$-element in a cartesian product $A\times B$ is the pair $\langle\perp_A,\perp_B\rangle$, the interpretations $[\![\Omega]\!]$ and $[\![\langle\Omega,\Omega\rangle]\!]$ are equal (they are both $\perp$). Let us consider for instance a call-by-value application

$$P \;:=\; (\widetilde{\lambda} x.*)\,\tilde{}\,M.$$

For $M=\Omega$, this term diverges, but for $M=\langle\Omega,\Omega\rangle$ it evaluates to $*$. In order to deal with this problem, we introduce two kinds of types. We restrict the arguments of call-by-value applications (and other call-by-value expressions) to elements of those types where termination coincides with non-$\perp$-interpretation. This does certainly not hold in general: a universal procedure for verifying that, say, a function f is different from the always diverging function requires the computation of the results $f(x)$ for different arguments x in parallel.

SFL Types

In the previous paragraph, it has been argued that, due to the requirement of sequentiality, it is impossible to define all SFL type constructions in the usual way without any constraints. As already suggested, we use two different kinds of types in order to deal with this problem. This idea is not new, a similar technique has been used before by A. Meyer and S. Cosmadakis [MC88]. With their type system, however, they have not been able to handle smash products.

In order to distinguish types of different kinds, we use different letters. We denote arbitrary types by τ, τ', τ_1, $\tau_2\ldots$. Those types for which it is sequentially semidecidable whether their terms are not interpreted as bottom are called *computational types*, *SSB-types* or just *σ-types*, they are denoted by σ, σ', σ_1, $\sigma_2\ldots$. Although semantically motivated, this condition can be formulated purely syntactically. Nevertheless, it turns out that computational types are interpreted as SSB-objects in SD. The grammar for the types is given by

$$\begin{array}{rcl} \sigma & ::= & \alpha \mid \tau_\perp \mid \sigma \otimes \sigma \mid \sum_{i=1}^{n} \tau \mid \sigma \oplus \sigma \mid \mu\alpha.\sigma \\ \tau & ::= & \sigma \mid \tau \times \tau \mid \tau \to \tau \mid \sigma \circ\!\!\to \tau \mid \mu\alpha.\tau \end{array}$$

where α ranges over the set of all type variables and n over the set of natural numbers. As usual, we use $\tau_1 \to \tau_2 \to \tau_3$ as an abbreviation for $\tau_1 \to (\tau_2 \to \tau_3)$.

We have already seen that the term $P = (\widetilde{\lambda} x.*)\tilde{\ } M$ from above causes problems if the subterm M cannot be observed to terminate. Hence, the argument type of a term defining a strict function has to be a computational type. We will later analyse in detail when we may take arbitrary types and when we have to restrict ourselves to compuptational types.

Note that $\tau_\perp$ is a special instance of the separated sum type $\sum_{i=1}^{1} \tau_i$. However, as the lifting operator is a very fundamental and important type former in SFL, we keep it in the definition of SFL type formers.

One may wonder how pure call-by-value languages such as FPC can fit into this approach. In the present notation, all FPC types need to be computational, however, function types as $\sigma \circ\!\!\to \tau$ are not. This problem is usually solved by lifting function types in call-by-value languages, i. e. the interpretation of of a type $\sigma \circ\!\!\to \tau$ is not the strict function space $[\![\![\sigma]\!] \circ\!\!\to [\![\tau]\!]]$, but $[\![\![\sigma]\!] \circ\!\!\to [\![\tau]\!]]_\perp$.

SFL Terms

The terms of SFL are given by the grammar

$$\begin{array}{rl} M ::= & x \mid \langle M, M\rangle \mid \mathbf{pr_1}\, M \mid \mathbf{pr_2}\, M \mid (M, M) \mid \widetilde{\mathbf{pr_1}}\, M \mid \widetilde{\mathbf{pr_2}}\, M \mid \\ & \mathbf{in}_i^{\tau,\dots,\tau} M \mid \mathbf{case}^{\tau,\dots,\tau,\tau} M \,\mathbf{of}\, \mathbf{in}_i\, x \Rightarrow M, \dots, M \mid \widetilde{\mathbf{inl}}^{\sigma,\sigma} M \mid \widetilde{\mathbf{inr}}^{\sigma,\sigma} M \mid \\ & \widetilde{\mathbf{case}}^{\sigma,\sigma,\tau} M \;\mathbf{of}\; \widetilde{\mathbf{inl}}\, x \Rightarrow M; \widetilde{\mathbf{inr}}\, x \Rightarrow M \mid \mathbf{up}\, M \mid \mathbf{down}\, M \mid \lambda x{:}\tau.M \mid \\ & M\,M \mid \widetilde{\lambda} x{:}\sigma.M \mid M\tilde{\ }M \mid \mathbf{fold}^{\mu\alpha.\tau} M \mid \mathbf{unfold}^{\mu\alpha.\tau} M \end{array}$$

where x ranges over the set of all variables. Type annotations are only used for type inference, they are often omitted when they are clear from the context. Term formers with tildes or round brackets correspond to types with circled connectives, their evaluation is in accordance with the call-by-value paradigm. Terms not containing any free variables are also called *programs*.

Typing Rules

For the typing rules, we need to consider terms with free variables. Thus, we look at terms-in-contexts of the form $\Gamma \vdash M : \tau$ where the *(type) context* Γ is set of pairs $x{:}\tau$ assigning each free variable x of M with a type.

We assume that all occurring types are *closed*, i. e. they do not contain type variables that are not bound by a μ-expression.

The typing rules are to be read in the following way: If the expressions above the line are well-typed SFL terms-in-contexts, then so is the one below the line.

$$\frac{}{\Gamma, x{:}\tau \vdash x : \tau}$$

$$\frac{\Gamma \vdash M : \tau_1 \qquad \Gamma \vdash N : \tau_2}{\Gamma \vdash \langle M, N \rangle : \tau_1 \times \tau_2} \qquad \frac{\Gamma \vdash M : \tau_1 \times \tau_2}{\Gamma \vdash \mathbf{pr_1} M : \tau_1} \qquad \frac{\Gamma \vdash M : \tau_1 \times \tau_2}{\Gamma \vdash \mathbf{pr_2} M : \tau_2}$$

$$\frac{\Gamma \vdash M : \sigma_1 \qquad \Gamma \vdash N : \sigma_2}{\Gamma \vdash (M, N) : \sigma_1 \otimes \sigma_2} \qquad \frac{\Gamma \vdash M : \sigma_1 \otimes \sigma_2}{\Gamma \vdash \widetilde{\mathbf{pr_1}}\, M : \sigma_1} \qquad \frac{\Gamma \vdash M : \sigma_1 \otimes \sigma_2}{\Gamma \vdash \widetilde{\mathbf{pr_2}}\, M : \sigma_2}$$

$$\frac{\Gamma \vdash M : \tau_i}{\Gamma \vdash \mathbf{in}_i^{\tau_1, \dots, \tau_n} M : \sum_{i=1}^{n} \tau_i} \quad \text{for all } n \in \mathbb{N},\ i = 1, \dots, n$$

$$\frac{\Gamma \vdash M : \sum_{i=1}^{n} \tau_i \qquad \Gamma, x_1{:}\tau_1 \vdash M_1 : \tau \qquad \cdots \qquad \Gamma, x_n{:}\tau_n \vdash M_n : \tau}{\Gamma \vdash \mathbf{case}^{\tau_1, \dots, \tau_n, \tau} M \ \mathbf{of}\ \mathbf{in}_1\, x_1 \Rightarrow M_1; \dots ; \mathbf{in}_n\, x_n \Rightarrow M_n : \tau} \quad \text{for all } n \in \mathbb{N}$$

$$\frac{\Gamma \vdash M : \sigma_1}{\Gamma \vdash \widetilde{\mathbf{inl}}^{\sigma_1, \sigma_2} M : \sigma_1 \oplus \sigma_2} \qquad \frac{\Gamma \vdash M : \sigma_2}{\Gamma \vdash \widetilde{\mathbf{inr}}^{\sigma_1, \sigma_2} M : \sigma_1 \oplus \sigma_2}$$

$$\frac{\Gamma \vdash M : \sigma_1 \oplus \sigma_2 \qquad \Gamma, x_1{:}\sigma_1 \vdash M_1 : \tau \qquad \Gamma, x_2{:}\sigma_2 \vdash M_2 : \tau}{\Gamma \vdash \widetilde{\mathbf{case}}^{\sigma_1, \sigma_2, \tau} M \ \mathbf{of}\ \widetilde{\mathbf{inl}}\ x_1 \Rightarrow M_1; \widetilde{\mathbf{inr}}\ x_2 \Rightarrow M_2 : \tau}$$

$$\frac{\Gamma, x{:}\tau_1 \vdash M : \tau_2}{\Gamma \vdash \lambda x{:}\tau_1.M : \tau_1 \to \tau_2} \qquad \frac{\Gamma \vdash M : \tau_1 \to \tau_2 \qquad \Gamma \vdash N : \tau_1}{\Gamma \vdash MN : \tau_2}$$

$$\frac{\Gamma, x{:}\sigma \vdash M : \tau}{\Gamma \vdash \tilde{\lambda} x{:}\sigma.M : \sigma \circ\!\!\to \tau} \qquad \frac{\Gamma \vdash M : \sigma \circ\!\!\to \tau \qquad \Gamma \vdash N : \sigma}{\Gamma \vdash M \tilde{\ } N : \tau}$$

$$\frac{\Gamma \vdash M : \tau}{\Gamma \vdash \mathbf{up}\, M : \tau_\perp} \qquad \frac{\Gamma \vdash M : \tau_\perp}{\Gamma \vdash \mathbf{down}\, M : \tau}$$

$$\frac{\Gamma \vdash M : \tau[\mu\alpha.\tau/\alpha]}{\Gamma \vdash \mathbf{fold}^{\mu\alpha.\tau} M : \mu\alpha.\tau} \qquad \frac{\Gamma \vdash M : \mu\alpha.\tau}{\Gamma \vdash \mathbf{unfold}^{\mu\alpha.\tau} M : \tau[\mu\alpha.\tau/\alpha]}$$

Some of the terms are known under different names as well. For instance, **up** is sometimes called **thunk** or **freeze** whereas **down** is occasionally referred to as **force** or **melt**. Other names for **fold** are **intro** and **abs**, **unfold** is also denoted by **elim** or **rep**.

4.3 Operational Semantics

Evaluation Rules

As mentioned above, the type formers $\times$, $\sum$ and $\to$ correspond to call-by-name types, whereas terms of type $\oplus$, $\otimes$ and $\circ\!\!\to$ are evaluated using a call-by-value strategy. Hence, SFL values for different kinds of types are defined differently:

$$V \quad ::= \quad \langle M, M\rangle \mid (V, V) \mid \mathbf{in}_i\, M \mid \widetilde{\mathbf{inl}}\, V \mid \widetilde{\mathbf{inr}}\, V \mid \lambda x{:}\tau.M \mid \widetilde{\lambda} x{:}\sigma.M \mid \mathbf{up}\, M \mid \mathbf{fold}\, V$$

The evaluation strategy for SFL is given by the following rules for closed terms:

$$\frac{}{\langle M_1, M_2\rangle \Downarrow \langle M_1, M_2\rangle} \qquad \frac{M \Downarrow \langle M_1, M_2\rangle \qquad M_1 \Downarrow V}{\mathbf{pr_1}\, M \Downarrow V} \qquad \frac{M \Downarrow \langle M_1, M_2\rangle \qquad M_2 \Downarrow V}{\mathbf{pr_2}\, M \Downarrow V}$$

$$\frac{M_1 \Downarrow V_1 \qquad M_2 \Downarrow V_2}{(M_1, M_2) \Downarrow (V_1, V_2)} \qquad \frac{M \Downarrow (V_1, V_2)}{\widetilde{\mathbf{pr_1}}\, M \Downarrow V_1} \qquad \frac{M \Downarrow (V_1, V_2)}{\widetilde{\mathbf{pr_2}}\, M \Downarrow V_2}$$

$$\frac{}{\mathbf{in}_i\, M \Downarrow \mathbf{in}_i\, M} \qquad \text{(for all } i = 1, \ldots, n)$$

$$\frac{M \Downarrow \mathbf{in}_i\, N \qquad M_i[N/x_i] \Downarrow V}{\mathbf{case}\, M\, \mathbf{of}\ \mathbf{in}_1\, x_1 \Rightarrow M_1; \ldots ; \mathbf{in}_n\, x_n \Rightarrow M_n \Downarrow V} \qquad \text{(for all } i = 1, \ldots, n)$$

$$\frac{M \Downarrow V}{\widetilde{\mathbf{inl}}\, M \Downarrow \widetilde{\mathbf{inl}}\, V} \qquad \frac{M \Downarrow \widetilde{\mathbf{inl}}\, V \qquad M_1[V/x_1] \Downarrow V'}{\widetilde{\mathbf{case}}\ M\ \mathbf{of}\ \widetilde{\mathbf{inl}}\, x_1 \Rightarrow M_1\, ; \widetilde{\mathbf{inr}}\, x_2 \Rightarrow M_2 \Downarrow V'}$$

$$\frac{M \Downarrow V}{\widetilde{\mathbf{inr}}\, M \Downarrow \widetilde{\mathbf{inr}}\, V} \qquad \frac{M \Downarrow \widetilde{\mathbf{inr}}\, V \qquad M_2[V/x_2] \Downarrow V'}{\widetilde{\mathbf{case}}\ M\ \mathbf{of}\ \widetilde{\mathbf{inl}}\, x_1 \Rightarrow M_1\, ; \widetilde{\mathbf{inr}}\, x_2 \Rightarrow M_2 \Downarrow V'}$$

$$\frac{}{\lambda x{:}\tau.M \Downarrow \lambda x{:}\tau.M} \qquad \frac{M \Downarrow \lambda x{:}\tau.M' \qquad M'[N/x] \Downarrow V}{M N \Downarrow V}$$

$$\frac{}{\widetilde{\lambda} x{:}\tau.M \Downarrow \widetilde{\lambda} x{:}\tau.M} \qquad \frac{M \Downarrow \widetilde{\lambda} x{:}\tau.M' \qquad N \Downarrow V_1 \qquad M'[V_1/x] \Downarrow V_2}{M\,\tilde{}\,N \Downarrow V_2}$$

$$\frac{}{\mathbf{up}\, M \Downarrow \mathbf{up}\, M} \qquad \frac{M \Downarrow \mathbf{up}\, N \qquad N \Downarrow V}{\mathbf{down}\, M \Downarrow V}$$

$$\frac{M \Downarrow V}{\mathbf{fold}\, M \Downarrow \mathbf{fold}\, V} \qquad \frac{M \Downarrow \mathbf{fold}\, V}{\mathbf{unfold}\, M \Downarrow V}$$

The Fixed Point Combinator Y

All languages supporting recursive definitions need a tool to compute least fixed points. For every SFL type τ, this can be done be defining a *fixed point combinator*

$$\mathbf{Y}_\tau \;:=\; \mathbf{unfold}^{\varrho}(\mathbf{fold}^{\varrho} Z)(\mathbf{fold}^{\varrho} Z)$$

of type $(\tau \to \tau) \to \tau$ (where $\varrho = \mu\alpha.\big(\alpha \to (\tau \to \tau) \to \tau\big)$) with

$$Z \;:=\; \lambda x{:}\varrho.\lambda y{:}\tau{\to}\tau.\, y\big((\mathbf{unfold}^{\varrho} x)\, x\, y\big).$$

(This fixed point combinator goes back to A. Turing.) The derived evaluation rule for $\mathbf{Y}_\tau$ is:

$$\frac{M(\mathbf{Y}_\tau M) \Downarrow V}{\mathbf{Y}_\tau M \Downarrow V}$$

This evaluation rule is not derivable for the simpler term $Z(\mathbf{fold}\, Z)$ which could also be used as a fixed point combinator. The reason is that $\mathbf{unfold}\,(\mathbf{fold}\, Z)(\mathbf{fold}\, Z)$ and $Z(\mathbf{fold}\, Z)$ are equal up to observational equivalence (see below), so it is a matter of taste whether one prefers syntactic simplicity or operational smoothness.

Using the fixed point combinator, we can also define an always diverging term

$$\Omega_\tau \;:=\; \mathbf{Y}_\tau(\lambda x{:}\tau.\, x).$$

Furthermore, we use

$$* \;:=\; \mathbf{up}\, \Omega_{\mathbf{void}}$$

as an abbreviation for the (up to equivalence) only value of type **unit** where

$$\mathbf{void} := \mu\alpha.\alpha$$

is a type not containing any value and

$$\mathbf{unit} := \mathbf{void}_\perp$$

is a type with the (up to equivalence) single value $*$.

Although the denotations $[\![\sum_{i=1}^n \tau_i]\!]$ and $[\![(\tau_1)_\perp \oplus \cdots \oplus (\tau_n)_\perp]\!]$ as well as $[\![\tau_1 \to \tau_2]\!]$ and $[\![(\tau_1)_\perp \circ\!\!\to \tau_2]\!]$ are isomorphic in *SD* (the precise definition of the denotational semantics is given in the next section), it is not possible to define separated sum types and normal function types with their corresponding terms from coalesced sum types, strict function types and liftings in such a way that the evaluation rules from above can be derived. For instance, if we set

$$\tau_1 \to \tau_2 \;:=\; (\tau_1)_\perp \circ\!\!\to \tau_2$$

and

$$\lambda x{:}\tau.M \;:=\; \widetilde{\lambda} y{:}\tau_\perp.\, M[\mathbf{down}\, y/x], \qquad\qquad MN \;:=\; M\tilde{\ }(\mathbf{up}\, N),$$

then the term $(\lambda x{:}\tau.\,\mathbf{up}\, x)\,\Omega_\tau$ of type $\tau_\perp$ evaluates to $\mathbf{up}\,\big(\mathbf{down}\,(\mathbf{up}\,\Omega_\tau)\big)$ rather than to $\mathbf{up}\,\Omega_\tau$. The same phenomenon occurs for terms of a separated sum type defined by using coalesced sums types. However, as for the fixed point combinator, these definitions work up to observational equivalence, which is considered next.

Observational Equivalence

Let us consider the two SFL terms $\Omega_{\mathbf{unit}\times\mathbf{unit}}$ and $\langle\Omega_{\mathbf{unit}}, \Omega_{\mathbf{unit}}\rangle$. Whereas $\Omega_{\mathbf{unit}\times\mathbf{unit}}$ is a diverging term, $\langle\Omega_{\mathbf{unit}}, \Omega_{\mathbf{unit}}\rangle$ is already a value, nevertheless, they are both interpreted as $\perp_{\mathbf{2}\times\mathbf{2}}$ (see next section). Therefore, we should not be able to observe a difference in the operational behaviour of these two terms. Consequently, not all SFL types are suitable for oberserving termination behaviour, in particular, the type **unit** $\times$ **unit** is not. We define the notion of observational equivalence in SFL based on observations on the ground type **unit**, but, as we will see later, any computational type can be used yielding an equivalent definition. This is similar for PCF where function types cannot be used for observations. Based on the operational semantics for SFL, the relation $\sqsubseteq_{SFL}$ as defined below provides a suitable notion of observational (in-)equivalence. This requires the notion of a *context*[2]. Intuitively, a context is an SFL term $C[\cdot]$ with a 'hole' $[\cdot]$ into which we can plug an appropriately typed term M to obtain a well-typed term $C[M]$. In the following definition, $M \Downarrow$ means that there is a value V with $M \Downarrow V$.

DEFINITION 4.1. *Let M, N be closed SFL terms. We write $M \sqsubseteq_{SFL} N$ if $C[M]\Downarrow$ implies $C[N]\Downarrow$ for all contexts $C[\cdot]$ of type* **unit**.

This entails that $M \sqsubseteq_{SFL} N$ implies $C[M] \sqsubseteq_{SFL} C[N]$ for every context $C[\cdot]$. Due to this notion of observational equivalence, the terms $\langle\Omega_{\mathbf{unit}}, \Omega_{\mathbf{unit}}\rangle$ and $\Omega_{\mathbf{unit}\times\mathbf{unit}}$ cannot be observed to be different.

LEMMA 4.2. *Let M, N be closed SFL terms and σ be a computational type. Then $M \sqsubseteq_{SFL} N$ holds if, and only if, $C[M]\Downarrow$ implies $C[N]\Downarrow$ for all contexts $C[\cdot]$ of type σ.*

Proof. We show the slightly more general claim that, for two computational types σ, σ', $C[M]\Downarrow \implies C[N]\Downarrow$ (for all contexts $C[\cdot]$ of type σ) implies $C'[M]\Downarrow \implies C'[N]\Downarrow$ (for all contexts $C'[\cdot]$ of type σ'). This proves the lemma as **unit** is a computational type.

Suppose $C[M]\Downarrow \implies C[N]\Downarrow$ holds for all contexts $C[\cdot]$ of type σ and let C' be a context of type σ' with $C'[M]\Downarrow$. Then $(\widetilde{\lambda} x{:}\sigma'.V)\tilde{\ } C'[M]\Downarrow$ follows where V is an arbitrary value of type σ. By assumption, this entails $(\widetilde{\lambda} x{:}\sigma'.V)\tilde{\ } C'[N]\Downarrow$. Hence, $C'[N]$ terminates as well. ☐

In other words, if M is a closed term of type σ, then the term $(\widetilde{\lambda} x{:}\sigma.*)\tilde{\ } M : \mathbf{unit}$ terminates if, and only if, M does so as well (the assumption of σ being computational is crucial since such terms do not exist for non-computational types). As a consequence of this observation, any computational type would do in place of **unit** in Definition 4.1, yielding the same notion of observational equivalence. Replacing **unit** by a non-computational type, say **unit** $\times$ **unit**, in Definition 4.1 would change the notion of equivalence as e. g. $\Omega_{\mathbf{unit}\times\mathbf{unit}}$ and $\langle\Omega_{\mathbf{unit}}, \Omega_{\mathbf{unit}}\rangle$ could then be separated by the trivial context $[\cdot]$.

Let us focus on the question why it is not semidecidable within SFL whether a term of type **unit** $\times$ **unit** is interpreted as bottom. Syntactically speaking, why can there be no program P of type $\mathbf{unit} \times \mathbf{unit} \to \mathbf{unit}$ such that $P\langle\Omega_{\mathbf{unit}}, *\rangle$ and $P\langle *, \Omega_{\mathbf{unit}}\rangle$ evaluate to $*$, but $P\langle\Omega_{\mathbf{unit}}, \Omega_{\mathbf{unit}}\rangle$ diverges? The reason is that SFL is sequential language, i. e. the evaluation of P cannot abandon the computation of one component in order to start computing another one. In particular, if P starts evaluating the first argument of some term $\langle\Omega, M\rangle$, there is no way to find out whether M terminates. Things get even worse if we replace type **unit** $\times$ **unit** by a function type.

[2]Unfortunately, there is an overloading in terminology: these contexts have nothing to do with the previously considered *type contexts*.

The question arises what exactly goes wrong if one takes non-computational types as components of smash product or coalesced sum types. The usage of arbitrary types as components of a smash product type would contradict the spirit of a call-by-value evaluation strategy in a sequential language in which both components are sequentially evaluated. Similar considerations apply to coalesced sum types and strict function types. More concretely, type constructions like $\tau \oplus \sigma$, $\tau \otimes \sigma$ or $\tau \multimap \tau'$ (where τ is not a computational type) with the normal typing rules would allow us to define closed terms $\widetilde{\mathbf{case}}\ (\widetilde{\mathbf{inl}}\, M)\ \mathbf{of}\ \widetilde{\mathbf{inl}}\, x_1 \Rightarrow * ; \widetilde{\mathbf{inr}}\, x_2 \Rightarrow *$ (for coalesced sums), $\widetilde{\mathbf{pr_1}}\,(*, M)$ (for smash products) and $(\widetilde{\lambda} x{:}\tau.*)\tilde{\ } M$ (for strict functions, see above) of type **unit** that terminate if, and only if, M of type τ is not interpreted as bottom. Since this is sequentially not observable, these terms cannot exist in a purely sequential language.

It has been argued that, unlike for call-by-value languages, there are good reasons for identifying the terms $\Omega_{\tau_1 \times \tau_2}$ and $\langle \Omega_{\tau_1}, \Omega_{\tau_2} \rangle$ as well as $\Omega_{\tau_1 \to \tau_2}$ and $\lambda x{:}\tau_1.\Omega_{\tau_2}$ with respect to observational equivalence in SFL. As we will see later, this fits with the denotational semantics for SFL in *SD* where the interpretations of $\Omega_{\tau_1 \times \tau_2}$ and $\langle \Omega_{\tau_1}, \Omega_{\tau_2} \rangle$ as well as the ones of $\Omega_{\tau_1 \to \tau_2}$ and $\lambda x{:}\tau_1.\Omega_{\tau_2}$, respectively, are equal. However, we can encode terms of other sequential languages in SFL. For example, call-by-value abstraction and application as in FPC refer to the SFL terms

$$\mathbf{up}\,(\widetilde{\lambda} x{:}\sigma.M) \qquad \text{and} \qquad (\mathbf{down}\, M)\tilde{\ } N$$

whereas FPC pairs and products can be defined as

$$\mathbf{up}\,(M, N), \qquad \widetilde{\mathbf{pr_1}}\,(\mathbf{down}\, P), \qquad \widetilde{\mathbf{pr_2}}\,(\mathbf{down}\, P).$$

The denotations of the respective function and product types are liftings of the strict function spaces and smash products in *SD*. The advantage of the more general approach in this work is that the language SFL describes the categorical meaning of products and functions, i. e. the factorisation of the term model of SFL with respect to observational equivalence yields a cartesian closed category.

One might think of different ways of defining a language that is even closer to the structure of the category *SD*. After all, smash products, coalesced sums and strict function spaces do exist in *SD* even for non-SSB objects. One approach is to require semantic side conditions for the typing rules that guarantee strictness of functions. However, since this is not a semidecidable property, this approach fails. A more promising idea is to use zonal contexts, one zone for arbitrary variables and another one for *strict* variables. It is then possible to define constructors for, say, coalesced sums of even non-computational types, however, the typing rules for such a language would still require a semantic criterion telling whether certain argument terms are not interpreted as bottom.

On the one hand, SFL supports many type constructions that are important in Domain Theory, on the other hand, it has a proper operational semantics. It has been argued that a further augmentation by arbitrary smash sum types etc. would destroy the operational meaning of SFL, hence, SFL can be considered the most general metalanguage for the semantics of other (non-polymorphic) sequential languages.

4.4 An Equational Theory for SFL

The following axioms and rules define an equational theory on SFL. It is implicitly assumed that all occurring terms are well-typed with respect to their context. Let Γ be a context with variables $x_1, \dots, x_n$.

- $\Gamma \vdash M = M : \tau$,
- if $\Gamma \vdash M = N : \tau$ then $\Gamma \vdash N = M : \tau$,
- if $\Gamma \vdash M = N : \tau$ and $\Gamma \vdash N = P : \tau$ then $\Gamma \vdash M = P : \tau$,
- if $\Gamma \vdash M_1 = N_1 : \tau_1, \dots, \Gamma \vdash M_n = N_n : \tau_n$ then $\Gamma \vdash C[M_1, \dots, M_n] = C[N_1, \dots, N_n] : \tau$ (for every context $C[\cdot, \dots, \cdot]$),
- $\Gamma \vdash \mathbf{pr_1}\, \langle M_1, M_2 \rangle = M_1 : \tau_1$,
- $\Gamma \vdash \mathbf{pr_2}\, \langle M_1, M_2 \rangle = M_2 : \tau_2$,
- if $M_2[N_1/x_1, \dots, N_n/x_n] \Downarrow$ for all well-typed closed terms $N_1, \dots, N_n$ then $\Gamma \vdash \widetilde{\mathbf{pr_1}}\, (M_1, M_2) = M_1 : \sigma_1$,
- if $M_1[N_1/x_1, \dots, N_n/x_n] \Downarrow$ for all well-typed closed terms $N_1, \dots, N_n$ then $\Gamma \vdash \widetilde{\mathbf{pr_2}}\, (M_1, M_2) = M_2 : \sigma_2$,
- $\Gamma \vdash \big(\mathbf{case}\, (\mathbf{in}_i\, M)\, \mathbf{of}\ \mathbf{in}_1\, x_1 \Rightarrow M_1; \dots; \mathbf{in}_n\, x_n \Rightarrow M_n\big) = M_i[M/x_i] : \tau$ for all $i = 1, \dots, n$,
- if $M[N_1/x_1, \dots, N_n/x_n] \Downarrow$ for all well-typed closed terms $N_1, \dots, N_n$ then $\Gamma \vdash \big(\widetilde{\mathbf{case}}\ (\widetilde{\mathbf{inl}}\, M)\ \mathbf{of}\ \widetilde{\mathbf{inl}}\, y_1 \Rightarrow M_1\, ; \widetilde{\mathbf{inr}}\, y_2 \Rightarrow M_2\big) = M_1[M/y_1] : \tau$,
- if $M[N_1/x_1, \dots, N_n/x_n] \Downarrow$ for all well-typed closed terms $N_1, \dots, N_n$ then $\Gamma \vdash \big(\widetilde{\mathbf{case}}\ (\widetilde{\mathbf{inr}}\, M)\ \mathbf{of}\ \widetilde{\mathbf{inl}}\, y_1 \Rightarrow M_1\, ; \widetilde{\mathbf{inr}}\, y_2 \Rightarrow M_2\big) = M_2[M/y_2] : \tau$,
- $\Gamma \vdash \mathbf{down}\, (\mathbf{up}\, M) = M : \tau$,
- $\Gamma \vdash (\lambda x{:}\tau_1.M)N = M[N/x] : \tau_2$,
- if $x \notin FV(M)$ then $\Gamma \vdash \lambda x{:}\tau_1.Mx = M : \tau_1 \to \tau_2$,
- if $N[N_1/x_1, \dots, N_n/x_n] \Downarrow$ for all well-typed closed terms $N_1, \dots, N_n$ then $\Gamma \vdash (\widetilde{\lambda} x{:}\sigma.M)\tilde{\ } N = M[N/x] : \tau$,
- if $x \notin FV(M)$ then $\Gamma \vdash \widetilde{\lambda} x{:}\sigma.M\tilde{\ }x = M : \sigma \multimap \tau$,
- $\Gamma \vdash \mathbf{unfold}\, (\mathbf{fold}\, M) = M : \tau$,
- $\Gamma \vdash \mathbf{fold}\, (\mathbf{unfold}\, M) = M : \tau$.

Expressions of the form $\Gamma \vdash M = N : \tau$ that are derived with these axioms and rules are called *equations-in-contexts*. Of course, this equational system is by no means complete (which is actually impossible due to Gödel's Incompleteness Theorem). Moreover, there is the problem that the side conditions required for some of the equations are generally not even semidecidable. For closed terms, however, they do not cause any trouble.

Anyway, as we will see in the next section (Corollary 5.13), this equational theory is sound with respect to the denotational semantics as well as the notion of observational equivalence in SFL. This result is by no means trivial, it is actually a consequence of the Full Abstraction Theorem.

4.5 Some SFL Data Types

We finish this section with some examples of concrete data types that are definable in SFL. All of these types are computational types.

- Type **boolean** := **unit** $\oplus$ **unit**.

  As SFL is a sequential language, it is not possible to define the parallel-or function. Nevertheless, there are three kinds of sequential or- and and-implementations in SFL: **l–or** and **l–and**, which are strict in the first component, **r–or** and **r–and**, which are strict in the second one, and **s–or** and **s–and**, which are strict in both arguments.

$$
\begin{array}{rcl}
\mathbf{true} & := & \widetilde{\mathbf{inl}}\ * \\
\mathbf{false} & := & \widetilde{\mathbf{inr}}\ * \\
\mathbf{l\text{–}or} & := & \lambda b{:}\mathbf{boolean}.\lambda c{:}\mathbf{boolean}.\ \widetilde{\mathbf{case}}\ b\ \mathbf{of}\ \widetilde{\mathbf{inl}}\ x_1 \Rightarrow \mathbf{true}\,;\widetilde{\mathbf{inr}}\,x_2 \Rightarrow c \\
\mathbf{r\text{–}or} & := & \lambda b{:}\mathbf{boolean}.\lambda c{:}\mathbf{boolean}.\ \widetilde{\mathbf{case}}\ c\ \mathbf{of}\ \widetilde{\mathbf{inl}}\ x_1 \Rightarrow \mathbf{true}\,;\widetilde{\mathbf{inr}}\,x_2 \Rightarrow b \\
\mathbf{s\text{–}or} & := & \lambda b{:}\mathbf{boolean}.\lambda c{:}\mathbf{boolean}. \\
 & & \quad \widetilde{\mathbf{case}}\ b\ \mathbf{of}\ \widetilde{\mathbf{inl}}\ x_1 \Rightarrow (\tilde{\lambda} y{:}\mathbf{boolean}.\mathbf{true})\tilde{\ }c\,;\widetilde{\mathbf{inr}}\,x_2 \Rightarrow c \\
\mathbf{l\text{–}and} & := & \lambda b{:}\mathbf{boolean}.\lambda c{:}\mathbf{boolean}.\ \widetilde{\mathbf{case}}\ b\ \mathbf{of}\ \widetilde{\mathbf{inl}}\ x_1 \Rightarrow c\,;\widetilde{\mathbf{inr}}\,x_2 \Rightarrow \mathbf{false} \\
\mathbf{r\text{–}and} & := & \lambda b{:}\mathbf{boolean}.\lambda c{:}\mathbf{boolean}.\ \widetilde{\mathbf{case}}\ c\ \mathbf{of}\ \widetilde{\mathbf{inl}}\ x_1 \Rightarrow b\,;\widetilde{\mathbf{inr}}\,x_2 \Rightarrow \mathbf{false} \\
\mathbf{s\text{–}and} & := & \lambda b{:}\mathbf{boolean}.\lambda c{:}\mathbf{boolean}. \\
 & & \quad \widetilde{\mathbf{case}}\ b\ \mathbf{of}\ \widetilde{\mathbf{inl}}\ x_1 \Rightarrow c\,;\widetilde{\mathbf{inr}}\,x_2 \Rightarrow (\tilde{\lambda} y{:}\mathbf{boolean}.\,\mathbf{false})\tilde{\ }c \\
\mathbf{neg} & := & \lambda b{:}\mathbf{boolean}.\ \widetilde{\mathbf{case}}\ b\ \mathbf{of}\ \widetilde{\mathbf{inl}}\ x_1 \Rightarrow \widetilde{\mathbf{inr}}\,x_1\,;\widetilde{\mathbf{inr}}\,x_2 \Rightarrow \widetilde{\mathbf{inl}}\,x_2 \\
\mathbf{cond} & := & \lambda b{:}\mathbf{boolean}.\lambda s{:}\tau.\lambda t{:}\tau.\ \widetilde{\mathbf{case}}\ b\ \mathbf{of}\ \widetilde{\mathbf{inl}}\ x_1 \Rightarrow s\,;\widetilde{\mathbf{inr}}\,x_2 \Rightarrow t
\end{array}
$$

  In the following, we use

$$\mathbf{if}\ M\ \mathbf{then}\ N\ \mathbf{else}\ P\ :=\ \mathbf{cond}\,MNP$$

  as syntactic sugar.

- Type **nat** := $\mu\alpha.(\mathbf{unit} \oplus \alpha)$ of natural numbers.

  There are basically two different ways of defining the type of natural numbers. The one here seems more natural as the underlying domain of its denotation is the flat domain $\mathbb{N}_\perp$.

$$
\begin{array}{rcl}
\mathbf{zero} & := & \mathbf{fold}\,(\widetilde{\mathbf{inl}}\,*) \\
\mathbf{succ} & := & \lambda n{:}\mathbf{nat}.\mathbf{fold}\,(\widetilde{\mathbf{inr}}\,n) \\
\mathbf{pred} & := & \lambda n{:}\mathbf{nat}.\widetilde{\mathbf{case}}\;(\mathbf{unfold}\,n)\;\mathbf{of}\;\widetilde{\mathbf{inl}}\,x_1 \Rightarrow \Omega_{\mathbf{nat}}\,;\widetilde{\mathbf{inr}}\,x_2 \Rightarrow x_2 \\
\mathbf{is\_zero?} & := & \lambda n{:}\mathbf{nat}.\widetilde{\mathbf{case}}\;(\mathbf{unfold}\,n)\;\mathbf{of}\;\widetilde{\mathbf{inl}}\,x_1 \Rightarrow \mathbf{true}\,;\widetilde{\mathbf{inr}}\,x_2 \Rightarrow \mathbf{false} \\
\mathbf{leq?} & := & \mathbf{Y}_{\mathbf{nat}\to\mathbf{nat}\to\mathbf{boolean}}\big(\lambda l{:}\mathbf{nat}\to\mathbf{nat}\to\mathbf{boolean}.\lambda m{:}\mathbf{nat}.\lambda n{:}\mathbf{nat}. \\
& & \quad \mathbf{if}\;\mathbf{is\_zero?}\,m\;\mathbf{then}\;(\widetilde{\lambda} x{:}\mathbf{nat}.\mathbf{true})\widetilde{\ }\,n \\
& & \qquad\qquad\qquad \mathbf{else\;if}\;\mathbf{is\_zero?}\,n\;\mathbf{then\;false} \\
& & \qquad\qquad\qquad\qquad\qquad \mathbf{else}\;l\,(\mathbf{pred}\,m)\,(\mathbf{pred}\,n)\big) \\
\mathbf{equal?} & := & \lambda m{:}\mathbf{nat}.\lambda n{:}\mathbf{nat}.\,\mathbf{l\text{--}and}\,(\mathbf{leq?}\,m\,n)\,(\mathbf{leq?}\,n\,m) \\
\mathbf{add} & := & \lambda m{:}\mathbf{nat}.\,\mathbf{Y}_{\mathbf{nat}\to\mathbf{nat}}\Big(\lambda a{:}\mathbf{nat}\to\mathbf{nat}.\lambda n{:}\mathbf{nat}. \\
& & \qquad\qquad \mathbf{if}\;\mathbf{is\_zero?}\;n\;\mathbf{then}\;m\;\mathbf{else\;succ}\;\big(a\,(\mathbf{pred}\,n)\big)\Big)
\end{array}
$$

Another possibility of defining a type of natural numbers is

$$\mathbf{lazynat} \;:=\; \mu\alpha.(\mathbf{unit} \oplus \alpha_\perp),$$

the so-called *lazy natural numbers*. The interesting feature about **lazynat** is that it has an infinity ordinal:

$$\mathbf{infinity} \;:=\; \mathbf{Y}_{\mathbf{lazynat}}\,\mathbf{lazysucc}$$

where

$$\mathbf{lazysucc} \;:=\; \lambda n{:}\mathbf{lazynat}.\mathbf{fold}\,\big(\widetilde{\mathbf{inr}}\,(\mathbf{up}\,n)\big)$$

defines the successor for the lazy natural numbers.

- Type $\mathbf{list}(\sigma) := \mu\alpha.\big(\mathbf{unit} \oplus (\sigma \otimes \alpha)\big)$ of lists of elements of type σ.

  Here is collection of programs that are important for computations involving lists:

$$
\begin{array}{rcl}
\mathbf{nil} & := & \mathbf{fold}\,(\widetilde{\mathbf{inl}}\,*) \\
\mathbf{cons} & := & \lambda s{:}\sigma.\lambda l{:}\mathbf{list}(\sigma).\,\mathbf{fold}\,\big(\widetilde{\mathbf{inr}}\,(s,l)\big) \\
\mathbf{head} & := & \lambda l{:}\mathbf{list}(\sigma).\,\widetilde{\mathbf{case}}\;(\mathbf{unfold}\,l)\;\mathbf{of}\;\widetilde{\mathbf{inl}}\,x_1 \Rightarrow \Omega_\sigma\,;\widetilde{\mathbf{inr}}\,x_2 \Rightarrow \widetilde{\mathbf{pr_1}}\,x_2 \\
\mathbf{tail} & := & \lambda l{:}\mathbf{list}(\sigma).\,\widetilde{\mathbf{case}}\;(\mathbf{unfold}\,l)\;\mathbf{of}\;\widetilde{\mathbf{inl}}\,x_1 \Rightarrow \Omega_{\mathbf{list}(\sigma)}\,;\widetilde{\mathbf{inr}}\,x_2 \Rightarrow \widetilde{\mathbf{pr_2}}\,x_2 \\
\mathbf{concat} & := & \mathbf{Y}_{\mathbf{list}(\sigma)\to\mathbf{list}(\sigma)\to\mathbf{list}(\sigma)} \\
& & \quad \Big(\lambda c{:}\mathbf{list}(\sigma)\to\mathbf{list}(\sigma)\to\mathbf{list}(\sigma).\lambda l_1{:}\mathbf{list}(\sigma).\lambda l_2{:}\mathbf{list}(\sigma). \\
& & \qquad \widetilde{\mathbf{case}}\;(\mathbf{unfold}\,l_1)\;\mathbf{of}\;\widetilde{\mathbf{inl}}\,x \Rightarrow l_2\,; \\
& & \qquad\qquad\qquad \widetilde{\mathbf{inr}}\,x \Rightarrow \mathbf{cons}\,(\mathbf{head}\,l_1)\big(c\,(\mathbf{tail}\,l_1)\,l_2\big)\Big) \\
\mathbf{revert} & := & \mathbf{Y}_{\mathbf{list}(\sigma)\to\mathbf{list}(\sigma)}\Big(\lambda r{:}\mathbf{list}(\sigma)\to\mathbf{list}(\sigma).\lambda l{:}\mathbf{list}(\sigma). \\
& & \quad \widetilde{\mathbf{case}}\;(\mathbf{unfold}\,l)\;\mathbf{of}\;\widetilde{\mathbf{inl}}\,x \Rightarrow \mathbf{nil}\,; \\
& & \qquad\qquad \widetilde{\mathbf{inr}}\,x \Rightarrow \mathbf{concat}\;\big(r\,(\mathbf{tail}\,l)\big)\big(\mathbf{cons}\,(\mathbf{head}\,l)\,\mathbf{nil}\big)\Big)
\end{array}
$$

This kind of lists does only exist for computational types. Non-SSB-types need to be lifted, yielding a data type that is sometimes called *lazy lists*.

- Type $\mathbf{stream}(\sigma) := \mu\alpha.(\sigma \otimes \alpha_\perp)$ of (possibly infinite) streams of elements of type σ.

  Streams are similar to lists, but can be infinite. As for the other data types, there are different ways of implementing streams, however, we only study the one presented here. Interestingly, the empty stream **s–nil** is implemented in SFL as a diverging program.

$$\begin{array}{rcl}
\mathbf{s\text{–}nil} & := & \Omega_{\mathbf{stream}(\sigma)} \\
\mathbf{s\text{–}cons} & := & \lambda x{:}\sigma.\lambda s{:}\mathbf{stream}(\sigma).\, \mathbf{fold}\,(x, \mathbf{up}\, s) \\
\mathbf{s\text{–}head} & := & \lambda s{:}\mathbf{stream}(\sigma).\, \widetilde{\mathbf{pr_1}}\,(\mathbf{unfold}\, s) \\
\mathbf{s\text{–}tail} & := & \lambda s{:}\mathbf{stream}(\sigma).\, \mathbf{down}\,\big(\widetilde{\mathbf{pr_2}}\,(\mathbf{unfold}\, s)\big)
\end{array}$$

These are basic operations on streams, let us focus to some more complicated ones. The program **map** extends a term M of type $\sigma \to \sigma$ to a term $\mathbf{map}\, M$ of type $\mathbf{stream}(\sigma) \to \mathbf{stream}(\sigma)$, applying M to each component of the stream. For a predicate $P\colon \sigma \to \mathbf{boolean}$, the program **filter** filters all elements in a stream that satisfy P. Finally, if S and T are terms defining streams, then $\mathbf{merge}\, S\, T$ merges the components of S and T into one single stream. Unfortunately, this works only as long as the elements of both streams are defined.

$$\begin{array}{rcl}
\mathbf{map} & := & \lambda f{:}\sigma{\to}\sigma. \\
& & \quad \mathbf{Y}_{\mathbf{stream}(\sigma)\to\mathbf{stream}(\sigma)}\Big(\lambda m{:}\mathbf{stream}(\sigma){\to}\mathbf{stream}(\sigma).\lambda s{:}\mathbf{stream}(\sigma). \\
& & \qquad\qquad \mathbf{cons}\,\big(f\,(\mathbf{s\text{–}head}\, s)\big)\,\big(m\,(\mathbf{s\text{–}tail}\, s)\big)\Big) \\
\mathbf{filter} & := & \lambda p{:}\sigma{\to}\mathbf{boolean}. \\
& & \quad \mathbf{Y}_{\mathbf{stream}(\sigma)}\Big(\lambda f{:}\mathbf{stream}(\sigma){\to}\mathbf{stream}(\sigma).\lambda s{:}\mathbf{stream}(\sigma). \\
& & \qquad \mathbf{if}\ p\,(\mathbf{s\text{–}head}\, s)\ \mathbf{then\ cons}\,(\mathbf{s\text{–}head}\, s)\,\big(f\,(\mathbf{s\text{–}tail}\, s)\big) \\
& & \qquad\qquad \mathbf{else}\ f\,(\mathbf{s\text{–}tail}\, s)\Big) \\
\mathbf{merge} & := & \mathbf{Y}_\tau\Big(\lambda m{:}\tau.\lambda s{:}\mathbf{stream}(\sigma).\lambda t{:}\mathbf{stream}(\sigma). \\
& & \quad \mathbf{cons}\,(\mathbf{s\text{–}head}\, s)\Big(\mathbf{cons}\,(\mathbf{s\text{–}head}\, t)\,\big(m\,(\mathbf{s\text{–}tail}\, s)\,(\mathbf{s\text{–}tail}\, t)\big)\Big)\Big) \\
& & \text{where } \tau := \mathbf{stream}(\sigma) \to \mathbf{stream}(\sigma) \to \mathbf{stream}(\sigma)
\end{array}$$

As they can be used to handle infinite structures, streams are an important data type in programming languages. For instance, the streams of natural numbers and Fibonacci numbers can be defined as follows:

$$
\begin{aligned}
\textbf{natstream} \quad &:= \quad \mathbf{Y}_{\mathbf{nat}\to\mathbf{stream}(\mathbf{nat})} \\
&\qquad \Big(\lambda f{:}\mathbf{nat}{\to}\mathbf{stream}(\mathbf{nat}).\lambda n{:}\mathbf{nat}.\,\mathbf{s{-}cons}\, n\, \big(f\,(\mathbf{succ}\, n)\big)\Big)\,\mathbf{zero} \\
\textbf{fibonacci} \quad &:= \quad \mathbf{Y}_{\mathbf{nat}\to\mathbf{nat}\to\mathbf{stream}(\mathbf{nat})} \\
&\qquad \Big(\lambda f{:}\mathbf{nat}{\to}\mathbf{nat}{\to}\mathbf{stream}(\mathbf{nat}).\lambda m{:}\mathbf{nat}.\lambda n{:}\mathbf{nat}. \\
&\qquad\quad \mathbf{s{-}cons}\, m\, \big(f\, n\, (\mathbf{add}\, m\, n)\big)\Big)\,\mathbf{zero}\,(\mathbf{succ}\ \mathbf{zero})
\end{aligned}
$$

As a last example we look at the *Sieve of Eratosthenes*, calculating the infinite stream of all prime numbers:

$$
\textbf{primes} \quad := \quad \textbf{sieve}\ \big(\mathbf{s{-}tail}\,(\mathbf{s{-}tail}\ \textbf{natstream})\big)
$$

where

$$
\begin{aligned}
\textbf{sieve} \quad &:= \quad \mathbf{Y}_{\mathbf{stream}(\mathbf{nat})}\Bigg(\lambda f{:}\mathbf{stream}(\mathbf{nat}){\to}\mathbf{stream}(\mathbf{nat}).\lambda s{:}\mathbf{stream}(\mathbf{nat}). \\
&\qquad \mathbf{if}\ \mathbf{is\_zero?}\,(\mathbf{s{-}head}\, s) \\
&\qquad\qquad \mathbf{then}\ f(\mathbf{s{-}tail}\, s) \\
&\qquad\qquad \mathbf{else}\ \mathbf{s{-}cons}\,(\mathbf{s{-}head}\, s)\,\Big(f\big(\mathbf{drop}\,(\mathbf{s{-}head}\, s)\,(\mathbf{s{-}tail}\, s)\big)\Big)\Bigg) \\
\textbf{drop} \quad &:= \quad \lambda n{:}\mathbf{nat}.\lambda s{:}\mathbf{stream}(\mathbf{nat}).\,\mathbf{aux}\,(\mathbf{pred}\, n)\, n\, s \\
\textbf{aux} \quad &:= \quad \mathbf{Y}_\tau\Big(\lambda a{:}\tau.\lambda m{:}\mathbf{nat}.\lambda n{:}\mathbf{nat}.\lambda s{:}\mathbf{stream}(\mathbf{nat}) \\
&\qquad \mathbf{if}\ \mathbf{is\_zero?}\, m\ \mathbf{then}\ \mathbf{cons}\ \mathbf{zero}\,\big(a\,(\mathbf{pred}\, n)\, n\,(\mathbf{s{-}tail}\, s)\big) \\
&\qquad\qquad\qquad\quad \mathbf{else}\ \mathbf{cons}\,(\mathbf{s{-}head}\, s)\,\big(a\,(\mathbf{pred}\, m)\, n\,(\mathbf{s{-}tail}\, s)\big)\Big) \\
&\quad \text{with } \tau := \mathbf{nat} \to \mathbf{nat} \to \mathbf{stream}(\mathbf{nat}) \to \mathbf{stream}(\mathbf{nat})
\end{aligned}
$$

5 The Interpretation of SFL in $\mathcal{SD}$

5.1 Denotation of SFL Types and Terms

Type Denotations

The semantics for finite SFL types (i. e. types that do not contain type variables or recursive types except **void**) is the one suggested by the choice of symbols for the connectives, i. e. $[\![\tau_\perp]\!] = [\![\tau]\!]_\perp$, $[\![\sigma_1 \otimes \sigma_2]\!] = [\![\sigma_1]\!] \otimes [\![\sigma_2]\!]$, $[\![\sigma_1 \oplus \sigma_2]\!] = [\![\sigma_1]\!] \oplus [\![\sigma_2]\!]$, $[\![\tau_1 \times \tau_2]\!] = [\![\tau_1]\!] \times [\![\tau_2]\!]$ etc. In particular, the family of Kripke relations of a type denotation $[\![\tau]\!]$ is already determined by the structure of τ. Therefore, we usually do not distinguish between the interpretation $[\![\tau]\!]$ of an SFL type τ and its underlying domain. We write $[\![\tau]\!]_{\mathcal{C}}$ for the $\mathcal{C}$-Kripke relations on $[\![\tau]\!]$.

The interpretation of a simple recursive type $\mu\alpha.\tau$ (i. e. τ does not contain further μ's) is determined by the mixed variant functor $F_\tau\colon \mathcal{SD}_\perp^{op} \times \mathcal{SD}_\perp \to \mathcal{SD}$. Formally, this functor can be defined by induction on the structure of τ:

$$
\begin{aligned}
F_\alpha(D,E) &= E, & F_{\tau_\perp}(D,E) &= \big(F_\tau(D,E)\big)_\perp,\\
F_{\sigma_1\otimes\sigma_2}(D,E) &= F_{\sigma_1}(D,E) \otimes F_{\sigma_2}(D,E), & F_{\sigma_1\oplus\sigma_2}(D,E) &= F_{\sigma_1}(D,E) \oplus F_{\sigma_2}(D,E),\\
F_{\sum_{i=1}^n \tau_i}(D,E) &= \sum_{i=1}^{n} F_{\tau_i}(D,E), & F_{\tau_1\times\tau_2}(D,E) &= F_{\tau_1}(D,E) \times F_{\tau_2}(D,E),\\
F_{\tau_1\to\tau_2}(D,E) &= [F_{\tau_1}(E,D) \to F_{\tau_2}(D,E)], & F_{\sigma\circ\!\!\to\tau}(D,E) &= [F_\sigma(E,D) \circ\!\!\to F_\tau(D,E)].
\end{aligned}
$$

The action of F_τ on morphisms is defined as in Section 2. Due to Proposition 2.1 and Proposition 3.11, respectively, there exists the bilimit $\mathsf{fix}(F_\tau)$ in $\mathcal{SD}_\perp$, which is used as denotation for the recursive type $\mu\alpha.\tau$. The $\mathcal{C}$-Kripke relations on $\mathsf{fix}(F_\tau)$ are defined componentwise. In particular, the type **void** $= \mu\alpha.\alpha$ is interpreted as the bilimit $\prod_{n\in\mathbb{N}} \mathbf{1} \cong \mathbf{1}$, the terminal object in $\mathcal{SD}_\perp$. This construction works since all type constructing functors are locally continuous and the embeddings $e_{n+1,n}\colon F_\tau^{(n)}(\mathbf{1},\mathbf{1}) \circ\!\!\to F_\tau^{(n+1)}(\mathbf{1},\mathbf{1})$ and projections $p_{n,n+1}\colon F_\tau^{(n+1)}(\mathbf{1},\mathbf{1}) \circ\!\!\to F_\tau^{(n)}(\mathbf{1},\mathbf{1})$ are strict and uniform.

For nested recursive types like $\mu\beta.\mu\alpha.\tau$, we need to consider parametrised functors. Even if $\mu\beta.\mu\alpha.\tau$ is a closed type, τ may contain two free type variables α and β. Therefore, the corresponding functor F_τ has four arguments for the negative and positive occurrences of α and β. In the general case, if $\mu\alpha.\tau$ has n free type variables $\beta_1,\dots,\beta_n$, we have to deal with a parametrised functor with $2n+2$ arguments. This can be simplified in the following way: the $2n$ parameters for the (negative and positive occurrences of the) β's are collected in two parameters for the vector $\vec{\beta} = (\beta_1,\dots,\beta_n)$. Hence, we can restrict ourselves to functors with only four parameters: two for $\vec{\beta}$ and two for α.

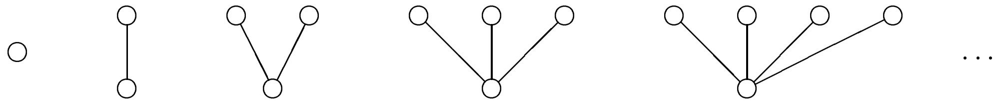
Suppose $F_\tau\colon \mathcal{SD}_\perp^{op} \times \mathcal{SD}_\perp \times \mathcal{SD}_\perp^{op} \times \mathcal{SD}_\perp \to \mathcal{SD}_\perp$ is a functor corresponding to an SFL type τ. Because of Proposition 2.3 and Proposition 3.11, respectively, there is a locally continuous functor $F_{\mu\alpha.\tau}\colon \mathcal{SD}_\perp^{op} \times \mathcal{SD}_\perp \to \mathcal{SD}_\perp$ such that

$$F_{\mu\alpha.\tau}(D^-,D^+) \;\cong\; F_\tau\big(D^-,D^+,F_{\mu\alpha.\tau}(D^+,D^-),F_{\mu\alpha.\tau}(D^-,D^+)\big)$$

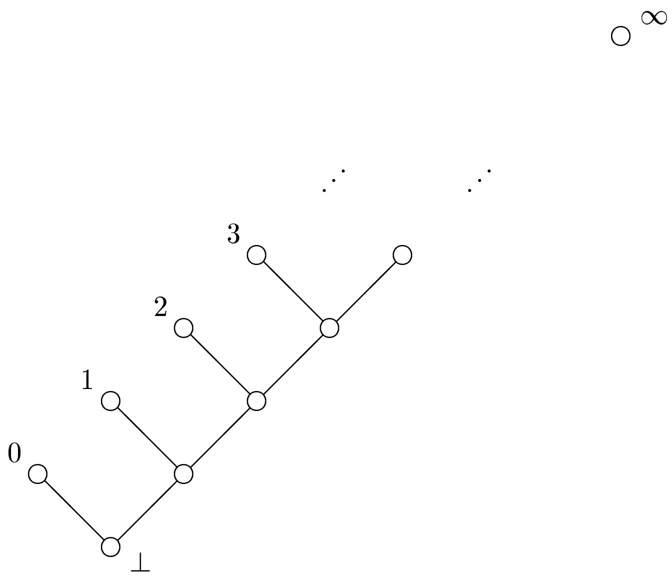
holds for each pair of objects D^-,D^+ in $\mathcal{SD}_\perp$. (Recall that the isomorphisms are given by

fold_{D^-,D^+} and $\mathsf{unfold}_{D^-,D^+}$.) Hence, nested recursive types can be interpreted in the same way as simple ones.

Let us have a closer look at the interpretation of some of the data types in Section 4. For example, the flat Boolean domain $\mathbb{B}_\perp$ is the denotation of type **boolean** $= \mathbf{2} \oplus \mathbf{2}$. The type **nat** $= \mu\alpha.(\mathbf{unit} \oplus \alpha)$ of natural number is interpreted as bilimit of the increasing sequence



which yields the flat domain of natural numbers $\mathbb{N}_\perp$. The underlying domain of the interpretation of the type **lazynat** $= \mu\alpha.(\mathbf{unit} \oplus \alpha_\perp)$ of lazy natural numbers is



where the term **infinity** is interpreted as the top element ∞.

Closure Properties of Type Denotations

As for many other languages, all SFL types live in the subcategory of Scott domains:

PROPOSITION 5.1. *Every denotation of an SFL type τ is a uniformly Scott domain.*

Proof. This kind of proposition is usually proved by induction on the structure of τ. However, since we consider recursive types, this is not possible without modifications. The following technique, which we call *parametrised induction*, shows a method of how to deal with this problem, it will be used frequently throughout this section. Although this particular proposition is a rather direct consequence of Proposition 3.21, we discuss the case of recursive types in detail to illustrate the technique of this proof.

Let ρ be a type variable environment, i. e. a function from the set of all type variables to the set of all closed types. Then $\tau[\rho]$ stands for the type in which every free occurrence of a type variable α in τ is replaced by $\rho(\alpha)$.

In order to verify that every type denotation is a uniformly Scott domain, we prove the following, more general claim for every SFL type τ.

> If $[\![\rho(\alpha)]\!]$ is a uniformly Scott domain for every free type variable α in τ, then so is $[\![\tau[\rho]]\!]$.

Due to Proposition 3.21, the full subcategory of uniformly Scott domains is stable under the interpretation of all finitary type constructions. For a recursive type $\mu\alpha.\tau$, we define a sequence $(\rho_n)_{n\in\mathbb{N}}$ of environments ρ_n in the following way. For every $n \in N$ and every free type variable β in $\mu\alpha.\tau$, $\rho_n(\beta)$ is defined as $\rho(\beta)$. Furthermore, $\rho_n(\alpha)$ is inductively defined by

$$\rho_0(\alpha) \;:=\; \mathbf{void}, \qquad \rho_{n+1}(\alpha) \;:=\; \tau[\rho_n]$$

All denotations $[\![\tau[\rho_n]]\!]$ are uniformly Scott domains by induction hypothesis, thus, so is the bilimit $[\![(\mu\alpha.\tau)[\rho]]\!]$. $\square$

This entails the next result which is crucial for proving that the category *SD* is a fully abstract model for SFL.

COROLLARY 5.2. *The underlying domain of the denotation of an SFL type is algebraic.* $\square$

The following lemma justifies the use of the notation *SSB-types* for computational types in SFL:

LEMMA 5.3. *If σ is a computational SFL type, then $[\![\sigma]\!]$ is an SSB-object in SD.*

Proof. Again, we use a parametrised induction on the structure of the computational type σ by proving the more general claim:

> Let σ be a computational type that may contain free type variables and ρ be a type variable environment. If $[\![\rho(\alpha)]\!]$ is an SSB-object for every free type variable α in σ, then so is $[\![\sigma[\rho]]\!]$.

Of course, this implies that the denotation of every closed computational SSB-type is an SSB-object in *SD*.

The claim is trivial for $\sigma = \alpha$ and follows directly from Proposition 3.15 for all finitary constructions. Let $\mu\alpha.\sigma'$ be a recursive type where σ' may contain free type variables and let ρ_n be defined as in the proof of Proposition 5.1. All denotations $[\![\sigma'[\rho_n]]\!]$ are SSB-objects by induction hypothesis, thus, due to Proposition 3.16, so is the bilimit $[\![(\mu\alpha.\sigma')[\rho]]\!]$. $\square$

Term Interpretations

Having studied the denotation of SFL types, we now focus on terms. They are interpreted as terms-in-contexts with respect to their operational behaviour:

- $[\![x_1 : \tau_1, \ldots, x_n : \tau_n \vdash x_i : \tau_i]\!] = \mathsf{pr}_i : \langle d_1, \ldots, d_n\rangle \mapsto d_i$,
- $[\![\Gamma \vdash \langle M_1, M_2\rangle : \tau_1 \otimes \tau_2]\!] = \langle [\![\Gamma \vdash M_1 : \tau_1]\!], [\![\Gamma \vdash M_2 : \tau_2]\!]\rangle$,
- $[\![\Gamma \vdash \mathbf{pr_1}\, M : \tau_1]\!] = \mathsf{pr}_1 \circ [\![\Gamma \vdash M : \tau_1 \times \tau_2]\!]$,
- $[\![\Gamma \vdash \mathbf{pr_2}\, M : \tau_2]\!] = \mathsf{pr}_2 \circ [\![\Gamma \vdash M : \tau_2 \times \tau_2]\!]$,
- $[\![\Gamma \vdash (M_1, M_2) : \sigma_1 \otimes \sigma_2]\!] = ([\![\Gamma \vdash M_1 : \sigma_1]\!], [\![\Gamma \vdash M_2 : \sigma_2]\!])$,
- $[\![\Gamma \vdash \widetilde{\mathbf{pr_1}}\, M : \sigma_1]\!] = \widetilde{\mathsf{pr}_1} \circ [\![\Gamma \vdash M : \sigma_1 \otimes \sigma_2]\!]$,
- $[\![\Gamma \vdash \widetilde{\mathbf{pr_2}}\, M : \sigma_2]\!] = \widetilde{\mathsf{pr}_2} \circ [\![\Gamma \vdash M : \sigma_1 \otimes \sigma_2]\!]$,
- $[\![\Gamma \vdash \mathbf{in}_i\, M : \sum_{i=1}^n \tau_i]\!] = \mathsf{in}_i \circ [\![\Gamma \vdash M : \tau_i]\!]$,
- $[\![\Gamma \vdash \mathbf{case}\, M\, \mathbf{of}\ \mathbf{in}_1\, x_1 \Rightarrow M_1 ; \ldots ; \mathbf{in}_n\, x_n \Rightarrow M_n : \tau]\!]$
  $= \lambda\!\!\lambda y^{[\![\Gamma]\!]}.[f_1^y, \ldots, f_n^y] \circ [\![\Gamma \vdash M {:} \sum_{i=1}^n \tau_i]\!](y)$ where $f_i^y = (\mathsf{curry}\, [\![\Gamma, x_i{:}\tau_i \vdash M_i : \tau_i]\!])(y)$,
- $[\![\Gamma \vdash \widetilde{\mathbf{inl}}\, M : \sigma_1 \oplus \sigma_2]\!] = \widetilde{\mathsf{inl}} \circ [\![\Gamma \vdash M : \sigma_1]\!]$,
- $[\![\Gamma \vdash \widetilde{\mathbf{inr}}\, M : \sigma_1 \oplus \sigma_2]\!] = \widetilde{\mathsf{inr}} \circ [\![\Gamma \vdash M : \sigma_2]\!]$,
- $[\![\Gamma \vdash \widetilde{\mathbf{case}}\, M\ \mathbf{of}\ \widetilde{\mathbf{inl}}\, x_1 \Rightarrow M_1 ; \widetilde{\mathbf{inr}}\, x_2 \Rightarrow M_2 : \tau]\!] = \lambda\!\!\lambda y^{[\![\Gamma]\!]}.\widetilde{[f_1^y, f_2^y]} \circ [\![\Gamma \vdash M : \sigma_1 \oplus \sigma_2]\!](y)$
  where $f_i^y = \mathsf{str}\big(\mathsf{curry}\, [\![\Gamma, x_i{:}\sigma_i \vdash M_i : \tau]\!])(y)\big)$ for $i = 1, 2$,
- $[\![\Gamma \vdash \mathbf{up}\, M : \tau_\perp]\!] = \mathsf{up} \circ [\![\Gamma \vdash M : \tau]\!]$,
- $[\![\Gamma \vdash \mathbf{down}\, M : \tau]\!] = \mathsf{down} \circ [\![\Gamma \vdash M : \tau_\perp]\!]$,
- $[\![\Gamma \vdash \lambda x{:}\tau_1.M : \tau_2]\!] = \mathsf{curry}\, [\![\Gamma, x{:}\tau_1 \vdash M : \tau_2]\!]$,
- $[\![\Gamma \vdash M_1 M_2 : \tau_1]\!] = \mathsf{eval} \circ \langle [\![\Gamma \vdash M_1 : \tau_2 \to \tau_1]\!], [\![\Gamma \vdash M_2 : \tau_2]\!]\rangle$,
- $[\![\Gamma \vdash \widetilde{\lambda} x{:}\sigma.M : \sigma \circ\!\!\rightarrow \tau]\!] = \lambda\!\!\lambda y^{[\![\Gamma]\!]}.\mathsf{str}\big((\mathsf{curry}\, [\![\Gamma, x{:}\sigma \vdash M : \tau]\!])(y)\big)$,
- $[\![\Gamma \vdash M_1{}^{\sim} M_2 : \tau]\!] = \widetilde{\mathsf{eval}} \circ ([\![\Gamma \vdash M_1 : \sigma \circ\!\!\rightarrow \tau]\!], [\![\Gamma \vdash M_2 : \sigma]\!])$,
- $[\![\Gamma \vdash \mathbf{fold}\, M : \mu\alpha.\tau]\!] = \mathsf{fold} \circ [\![\Gamma \vdash M : \tau[\mu\alpha.\tau/\alpha]]\!]$,
- $[\![\Gamma \vdash \mathbf{unfold}\, M : \tau[\mu\alpha.\tau/\alpha]]\!] = \mathsf{unfold} \circ [\![\Gamma \vdash M : \mu\alpha.\tau]\!]$

where, for all SFL contexts $\Gamma = x_1{:}\tau_1, \dots, x_n{:}\tau_n$, the interpretation $[\![\Gamma]\!]$ stands for the sequential domain which is given by the cartesian product $[\![\tau_1]\!] \times \cdots \times [\![\tau_n]\!]$ of denotations of all types $\tau_1, \dots, \tau_n$ in Γ. Furthermore, if M is a closed SFL term of type τ, we write $[\![M]\!]$ as abbreviation for $[\![\varnothing \vdash M : \tau]\!](\perp_{\mathbf{1}})$.

SFL terms are indeed interpreted as morphisms in *SD*. The proof for this is an easy induction. For most cases, this follows from Theorem 3.10.

The projections for smash product types exist due to Proposition 3.18 since the denotations of σ_1 and σ_2 are SSB-objects (see Lemma 5.3).

The subterms M_1 and M_2 in $\widetilde{\mathbf{case}}\ M\ \mathbf{of}\ \widetilde{\mathbf{inl}}\,x_1 \Rightarrow M_1\,;\widetilde{\mathbf{inr}}\,x_2 \Rightarrow M_2$ are interpreted as functions $\lambda y{:}[\![\Gamma]\!].f_i^y \colon [\![\Gamma]\!] \to ([\![\sigma_i]\!] \circ\!\!\to [\![\tau]\!])$ $(i = 1,2)$ where f_1^y and f_2^y are strict for all $y \in [\![\Gamma]\!]$. Due to Lemma 5.3, $[\![\sigma_1]\!]$ and $[\![\sigma_2]\!]$ are SSB-objects, thus, f_1^y and f_2^y are morphisms in $SD_\perp$ for all $y \in [\![\Gamma]\!]$ because of Lemma 3.14. Therefore, $\widetilde{[f_1, f_2]}$ is a (strict) morphism in *SD* as well. This shows that the denotation of a $\widetilde{\mathbf{case}}$-term is always a morphism in *SD*, for instance:

$$[\![\widetilde{\mathbf{case}}\ (\widetilde{\mathbf{inl}}\,\Omega)\ \mathbf{of}\ \widetilde{\mathbf{inl}}\,x_1 \Rightarrow *\,;\widetilde{\mathbf{inr}}\,x_2 \Rightarrow *]\!] \;=\; \perp_2$$

The proof that denotations of strict function terms and strict applications are morphisms in *SD* follows again from Lemma 3.14 and Lemma 5.3.

Finally, the maps fold and unfold are morphisms since they are part of the bilimit construction.

As expected, the interpretation of a fixed point combinator $\mathbf{Y}_\tau$ computes the least fixed point of a given function $f \colon [\![\tau]\!] \to [\![\tau]\!]$ in *SD*:

LEMMA 5.4. *Let τ be an SFL type. Then the denotation of the fixed point combinator $\mathbf{Y}_\tau$ is given by $[\![\mathbf{Y}_\tau]\!](f) = \bigsqcup^{\uparrow}_{n\in\mathbb{N}} f^n(\perp_{\mathbf{1}})$.*

Proof. We first examine the functor $F = F_{\alpha\to(\tau\to\tau)\to\tau}$ that corresponds to the recursive type $\mu\alpha.\big(\alpha \to (\tau \to \tau) \to \tau\big)$. An object A in $SD_\perp$ is mapped to $F(A,A) = \big[A \to [[\tau \to \tau]\!] \to [\![\tau]\!]]\big]$ whereas a pair of morphisms $f \colon B \circ\!\!\to A$, $g \colon A \circ\!\!\to B$ is mapped to a morphism $F(f,g) = \lambda h{:}A{\to}[\![\tau{\to}\tau]\!]{\to}[\![\tau]\!].h \circ f$ from $F(A,A)$ to $F(B,B)$. In particular, the embedding projection pairs for the sequence $\mathbf{1}, F(\mathbf{1},\mathbf{1}), F^{(2)}(\mathbf{1},\mathbf{1}), \dots$ are given by

$$\begin{aligned} e_{n+1,n} &= \lambda x{:}F^{(n)}(\mathbf{1},\mathbf{1}).\lambda y{:}F^{(n)}(\mathbf{1},\mathbf{1}).x\big(p_{n-1,n}(y)\big),\\ p_{n,n+1} &= \lambda x{:}F^{(n+1)}(\mathbf{1},\mathbf{1}).\lambda y{:}F^{(n-1)}(\mathbf{1},\mathbf{1}).x\big(e_{n,n-1}(y)\big) \end{aligned}$$

(with $n > 0$), or, more generally,

$$\begin{aligned} e_{m,n} &= \lambda x{:}F^{(n)}(\mathbf{1},\mathbf{1}).\lambda y{:}F^{(m-1)}(\mathbf{1},\mathbf{1}).x\big(p_{n-1,m-1}(y)\big),\\ p_{n,m} &= \lambda x{:}F^{(m)}(\mathbf{1},\mathbf{1}).\lambda y{:}F^{(n-1)}(\mathbf{1},\mathbf{1}).x\big(e_{m-1,n-1}(y)\big) \end{aligned}$$

(with $m \geq n > 0$). The corresponding functions $\mathsf{fold}_F \colon F\big(\mathsf{fix}(F), \mathsf{fix}(F)\big) \to \mathsf{fix}(F)$ and $\mathsf{unfold}_F \colon \mathsf{fix}_F \to F\big(\mathsf{fix}(F), \mathsf{fix}(F)\big)$ are defined by

$$\begin{aligned} \mathsf{fold}_F(f) &= \bigsqcup^{\uparrow}_{n\in\mathbb{N}} e_{n+1}\Big(\lambda x{:}F^{(n)}(\mathbf{1},\mathbf{1}).f\big(e_n(x)\big)\Big),\\ \mathsf{unfold}_F(d) &= \lambda x{:}\mathsf{fix}(F).\bigsqcup^{\uparrow}_{n\in\mathbb{N}} \big(p_{n+1}(d)\big)\big(p_n(x)\big) \end{aligned}$$

whereas the interpretation of $Z = \lambda x{:}\mu\alpha.\big(\alpha{\to}(\tau{\to}\tau){\to}\tau\big).\lambda y{:}\tau{\to}\tau.y\big((\mathbf{unfold}\,x)xy\big)$ is given by

$$\begin{aligned}
[\![Z]\!] &= \lambda x{:}\mathsf{fix}(F).\lambda f{:}[\![\tau{\to}\tau]\!].f\Big(\big(\mathsf{unfold}_F(x)\big)xf\Big)\\
&= \lambda x{:}\mathsf{fix}(F).\lambda f{:}[\![\tau{\to}\tau]\!].f\Big(\bigsqcup_{n\in\mathbb{N}}^{\uparrow}\big(p_{n+1,n}(x)\big)\big(p_n(x)\big)f\Big).
\end{aligned}$$

This implies

$$\begin{aligned}
\mathsf{fold}([\![Z]\!]) &= \bigsqcup_{n\in\mathbb{N}}^{\uparrow} e_{n+1}\Big(\lambda x{:}F^{(n)}(\mathbf{1},\mathbf{1}).[\![Z]\!]\big(e_n(x)\big)\Big),\\
&= \bigsqcup_{n\in\mathbb{N}}^{\uparrow} e_{n+1}\bigg(\lambda x{:}F^{(n)}(\mathbf{1},\mathbf{1}).\lambda f{:}[\![\tau{\to}\tau]\!].f\Big(\mathsf{unfold}_F\big(e_n(x)\big)\big(e_n(x)\big)f\Big)\bigg)\\
&= \bigsqcup_{n\in\mathbb{N}}^{\uparrow} e_{n+1}\Bigg(\lambda x{:}F^{(n)}(\mathbf{1},\mathbf{1}).\lambda f{:}[\![\tau{\to}\tau]\!].f\bigg(\bigsqcup_{m\in\mathbb{N}}^{\uparrow}\Big(p_{m+1}\big(e_n(x)\big)\Big)\Big(p_m\big(e_n(x)\big)\Big)f\bigg)\Bigg)\\
&= \bigsqcup_{n\in\mathbb{N}}^{\uparrow} e_{n+1}(\zeta_n)
\end{aligned}$$

with

$$\zeta_n \;:=\; \lambda x{:}F^{(n)}(\mathbf{1},\mathbf{1}).\lambda f{:}[\![\tau{\to}\tau]\!].f\big((e_{n+1,n}x)xf\big)$$

and, therefore,

$$\begin{aligned}
[\![\mathbf{Y}_\tau]\!] &= [\![\mathbf{unfold}\,(\mathbf{fold}\,Z)(\mathbf{fold}\,Z)]\!]\\
&= \big(\mathsf{unfold}_F \circ \mathsf{fold}_F([\![Z]\!])\big)\big(\mathsf{fold}_F([\![Z]\!])\big)\\
&= [\![Z]\!]\big(\mathsf{fold}_F([\![Z]\!])\big)\\
&= \lambda f{:}[\![\tau{\to}\tau]\!].f\Big(\bigsqcup_{n\in\mathbb{N}}^{\uparrow}\big(p_{n+1}(\bigsqcup_{m\in\mathbb{N}}^{\uparrow} e_{m+1}\zeta_m)\big)\big(p_n(\bigsqcup_{m\in\mathbb{N}}^{\uparrow} e_{m+1}\zeta_m)\big)f\Big)\\
&= \bigsqcup_{n\in\mathbb{N}}^{\uparrow} \lambda f{:}[\![\tau{\to}\tau]\!].f\big((p_{n+1}\circ e_{n+1}\zeta_n)(p_n\circ e_{n+1}\zeta_n)f\big)\\
&= \bigsqcup_{n\in\mathbb{N}}^{\uparrow} \lambda f{:}[\![\tau{\to}\tau]\!].f\big(\zeta_n(p_{n,n+1}\zeta_n)f\big)
\end{aligned}$$

holds. Furthermore,

$$\begin{aligned}
&\zeta_n\big(e_{n,m+1}\circ p_{m+1,n+1}(\zeta_n)\big)(f)\\
&\quad= \big(\lambda x{:}F^{(n)}(\mathbf{1},\mathbf{1}).\lambda f{:}[\![\tau{\to}\tau]\!].f\big((e_{n+1,n}x)xf\big)\big)\big(e_{n,m+1}\circ p_{m+1,n+1}(\zeta_n)\big)(f)\\
&\quad= f\Big(\Big(e_{n+1,n}\big(e_{n,m+1}\circ p_{m+1,n+1}(\zeta_n)\big)\Big)\big(e_{n,m+1}\circ p_{m+1,n+1}(\zeta_n)\big)f\Big)\\
&\quad= f\Big(\big(e_{n+1,m+1}\circ p_{m+1,n+1}(\zeta_n)\big)\big(e_{n,m+1}\circ p_{m+1,n+1}(\zeta_n)\big)f\Big)\\
&\quad= f\Big(\big(p_{m+1,n+1}(\zeta_n)\big)\big(p_{m,n}\circ e_{n,m+1}\circ p_{m+1,n+1}(\zeta_n)\big)f\Big)\\
&\quad= f\Big(\big(p_{m+1,n+1}(\zeta_n)\big)\big(p_{m,n+1}(\zeta_n)\big)f\Big)\\
&\quad= f\Big(\zeta_n\big(e_{n,m}\circ p_{m,n+1}(\zeta_n)\big)(f)\Big)
\end{aligned}$$

is true for all natural numbers m, n with $n > m$. This entails

$$\begin{aligned}
f\big(\zeta_n(p_{n,n+1}\zeta_n)f\big) &= f\big(\zeta_n(e_{n,n} \circ p_{n,n+1}\zeta_n)f\big) \\
&= f^2\big(\zeta_n(e_{n,n-1} \circ p_{n-1,n+1}\zeta_n)f\big) \\
&= f^3\big(\zeta_n(e_{n,n-2} \circ p_{n-2,n+1}\zeta_n)f\big) \\
&\vdots \\
&= f^{n+1}\big(\zeta_n(e_{n,0} \circ p_{0,n+1}\zeta_n)f\big) \\
&= f^{n+1}\big(\zeta_n\big(e_{n,0}(\perp_{\mathbf{1}})\big)f\big) \\
&= f^{n+1}\big(\zeta_n(\perp_{F^{(n)}(\mathbf{1},\mathbf{1})})f\big) \\
&= f^{n+2}(\perp_{[\![\tau]\!]}),
\end{aligned}$$

hence, $[\![\mathbf{Y}_\tau]\!](f) = \bigsqcup^{\uparrow}{}_{n\in\mathbb{N}} f\big(\zeta_n(p_{n,n+1}\zeta_n)f\big) = \bigsqcup^{\uparrow}{}_{n\in\mathbb{N}} f^{n+2}(\perp_{[\![\tau]\!]}) = \bigsqcup^{\uparrow}{}_{n\in\mathbb{N}} f^{n}(\perp_{[\![\tau]\!]})$ follows. This is certainly the least fixed point of the function $f\colon [\![\tau]\!] \to [\![\tau]\!]$. $\square$

As a consequence, the term Ω_τ is interpreted as the bottom element $\perp_{[\![\tau]\!]}$.

5.2 Computational Adequacy

Some basic properties of programming languages hold for SFL as well. The following lemma can be proved by an easy induction on the structure of M:

LEMMA 5.5. ***(Substitution Lemma)*** *Let $\Gamma, x{:}\tau' \vdash M : \tau$ and $\Gamma \vdash N : \tau'$ be terms-in-contexts. Then*

$$[\![\Gamma \vdash M[N/x] : \tau]\!](d) \quad = \quad [\![\Gamma, x{:}\tau' \vdash M : \tau]\!]\big(d, [\![\Gamma \vdash N : \tau']\!](d)\big)$$

holds for all $d \in [\![\Gamma]\!]$. $\square$

The following proposition is crucial for proving an adequacy result for SFL.

PROPOSITION 5.6. *Let M be a closed SFL term with $[\![M]\!] \neq \perp$. Then M evaluates to a value.*

Proof. The standard idea for proving a proposition like this is to define a family $\lhd = (\lhd_\tau)_{\tau \text{ closed SFL type}}$ of relations between elements of the denotation of a type τ and closed terms of type τ by induction of the structure of τ. However, this is not possible for SFL since SFL contains recursive types. To deal with this problem, we apply the concept of *parametrised invariant relations*, introduced by A. Pitts [Pit96].

We first need to define some notations. A relation R between the denotation $[\![\tau]\!]$ of a type τ and the set Λ^0_τ of all closed SFL terms of type τ is said to be *ω-complete* if $(\perp_{[\![\tau]\!]}, M) \in R$ holds for all terms $M \in \Lambda^0_\tau$ and, if C is an ω-chain in $[\![\tau]\!]$ with $(d, M) \in R$ for all $d \in C$, then $(\bigsqcup^{\uparrow}{}_{d\in C}\, d, M) \in R$ is true as well. The set of all ω-complete relations on $[\![\tau]\!] \times \Lambda^0_\tau$ is denoted by Rel_τ.

Given a morphism $f\colon [\![\tau]\!] \to [\![\tau]\!]$ in $SD_\perp$ and relations $R, S \in \mathsf{Rel}_\tau$, we write $f\colon R \subset S$ if $(d, M) \in R$ implies $\big(f(d), M\big) \in S$ for all $d \in [\![\tau]\!]$ and $M \in \Lambda^0_\tau$. This makes

$(\mathsf{Rel}_\tau)_{\tau \text{ closed SFL type}}$ a relational structure on the subcategory of type denotations of $SD_\perp$ in the sense of Definition 2.4.

Consider the parametrised functor $F_\tau\colon SD_\perp^{op} \times SD_\perp \times \cdots \times SD_\perp^{op} \times SD_\perp \to SD_\perp$ with $2n$ arguments corresponding to a recursive SFL type $\mu\alpha_1.\dots\mu\alpha_n.\tau$. Moreover, let $\tau_1', \dots, \tau_n'$ be closed SFL types and R_i be relations in $\mathsf{Rel}_{\tau_i'}$ (for $i = 1, \dots, n$). In the following, we define a relation $F_\tau(\vec{R}) := F_\tau(R_1, R_1, R_2, R_2, \dots, R_n, R_n) \in \mathsf{Rel}_{\tau[\tau_1'/\alpha_1, \dots, \tau_n'/\alpha_n]}$ such that $F_\tau(\overrightarrow{u^{-+}}) := F_\tau(u_1^-, u_1^+, \dots, u_n^-, u_n^+)\colon F_\tau(\vec{R}) \subset F_\tau(\vec{S})$ holds for all morphisms $u_i^-, u_i^+\colon [\![\tau_i']\!] \to [\![\tau_i']\!]$ in $SD_\perp$ with $u_i^-\colon S_i \subset R_i$, $u_i^+\colon R_i \subset S_i$ and $i = 1, \dots, n$. This is done by induction on the structure of τ. For finitary type constructions, the relations are defined as follows:

$$F_{\tau_1 \times \tau_2}(\vec{R}) = \big\{(d, M) \bigm| d = \perp \text{ or } M \Downarrow \langle M_1, M_2\rangle, \big(\mathsf{pr}_1(d), M_1\big) \in F_{\tau_1}(\vec{R}),\ \big(\mathsf{pr}_2(d), M_2\big) \in F_{\tau_2}(\vec{R})\big\},$$

$$F_{\sigma_1 \otimes \sigma_2}(\vec{R}) = \big\{(d, M) \bigm| d = \perp \text{ or } M \Downarrow (V_1, V_2),\ \big(\widetilde{\mathsf{pr}}_1(d), V_1\big) \in F_{\sigma_1}(\vec{R}),\ \big(\widetilde{\mathsf{pr}}_2(d), V_2\big) \in F_{\sigma_2}(\vec{R})\big\},$$

$$F_{\sum_{i=1}^m \tau_i}(\vec{R}) = \{(\perp, M)\} \cup \bigcup_{i=1}^m \big\{(\mathsf{in}_i\, d, M) \bigm| M \Downarrow \mathbf{in}_i\, M',\ (d, M') \in F_{\tau_i}(\vec{R})\big\},$$

$$F_{\sigma_1 \oplus \sigma_2}(\vec{R}) = \{(\perp, M)\} \cup \big\{(\widetilde{\mathsf{inl}}\, d, M) \bigm| M \Downarrow \widetilde{\mathbf{inl}}\, V,\ (d, V) \in F_{\sigma_1}(\vec{R})\big\} \cup \big\{(\widetilde{\mathsf{inr}}\, d, M) \bigm| M \Downarrow \widetilde{\mathbf{inr}}\, V,\ (d, V) \in F_{\sigma_2}(\vec{R})\big\},$$

$$F_{\tau_1 \to \tau_2}(\vec{R}) = \big\{(d, M) \bigm| d = \perp \text{ or } M \Downarrow \lambda x{:}\tau_1.M',\ (e, N) \in F_{\tau_1}(\vec{R}) \Rightarrow \big(d(e), M'[N/x]\big) \in F_{\tau_2}(\vec{R})\big\},$$

$$F_{\sigma \multimap \tau'}(\vec{R}) = \big\{(d, M) \bigm| d = \perp \text{ or } M \Downarrow \widetilde{\lambda} x{:}\sigma.M'(e, N) \in F_\sigma(\vec{R}) \Rightarrow \big(\widetilde{\mathsf{eval}}(d, e), (\widetilde{\lambda} x{:}\sigma.M')\tilde{\ } N\big) \in F_{\tau'}(\vec{R})\big\},$$

$$F_{\tau'_\perp}(\vec{R}) = \big\{(d, M) \bigm| d = \perp \text{ or } M \Downarrow \mathbf{up}\, M' \text{ and } \big(\mathsf{down}(d), M'\big) \in F_{\tau'}(\vec{R})\big\}.$$

It is straightforward to verify that those relations $F_\tau(\vec{R})$ are all ω-complete.

Next, we show that, if $u_i^-, u_i^+\colon [\![\tau_i']\!] \multimap [\![\tau_i']\!]$ are morphisms in $SD_\perp$ with $u_i^-\colon S_i \subset R_i$ and $u_i^+\colon R_i \subset S_i$ ($i = 1, \dots, n$), then also $F_\tau(u_1^-, u_1^+, \dots, u_n^-, u_n^+)\colon F_\tau(\vec{R}) \subset F_\tau(\vec{S})$ holds. The claim is proved for $F_{\sigma_1 \otimes \sigma_2}$ and $F_{\tau_1 \to \tau_2}$, the other cases can be verified in a similar way. Firstly, suppose $(d, M) \in F_{\sigma_1 \otimes \sigma_2}(\vec{R})$. The case $d = \perp$ is trivial, therefore, assume that M evaluates to a value (V_1, V_2) with $\big(\widetilde{\mathsf{pr}}_1(d), V_1\big) \in F_{\sigma_1}(\vec{R})$ and $\big(\widetilde{\mathsf{pr}}_2(d), V_2\big) \in F_{\sigma_2}(\vec{R})$. By induction hypothesis, $\big(F_{\sigma_i}(\overrightarrow{u^{-+}}) \circ \widetilde{\mathsf{pr}}_i(d), V_i\big) \in F_{\sigma_i}(\vec{S})$ holds for $i = 1, 2$. Hence, $\big(\widetilde{\mathsf{pr}}_1 \circ F_{\sigma_1 \otimes \sigma_2}(\overrightarrow{u^{-+}})(d), V_1\big) = \big((F_{\sigma_1}(\overrightarrow{u^{-+}}) \circ \widetilde{\mathsf{pr}}_1 \restriction \{d \mid F_{\sigma_2}(\overrightarrow{u^{-+}}) \circ \widetilde{\mathsf{pr}}_2(d) \neq \perp\})(d), V_1\big) \in F_{\sigma_1}(\vec{S})$ is true as well, and $\big(\widetilde{\mathsf{pr}}_2 \circ F_{\sigma_2}(\overrightarrow{u^{-+}})(d), V_2\big) \in F_{\sigma_2}(\vec{S})$ follows analogously. This entails $\big(F_{\sigma_1 \otimes \sigma_2}(\overrightarrow{u^{-+}})(d), M\big) \in F_{\sigma_1 \otimes \sigma_2}(\vec{S})$.

For the case $\tau = \tau_1 \to \tau_2$, suppose $(d, M) \in F_{\tau_1 \to \tau_2}(\vec{R})$. If we exclude the trivial case $d = \bot$, then M evaluates to a value $\lambda x{:}\tau_1.M'$ such that $\big(d(e), M'[N/x]\big) \in F_{\tau_2}(\vec{R})$ holds for all $(e,N) \in F_{\tau_1}(\vec{R})$. Now assume $(e',N') \in F_{\tau_1}(\vec{S})$, we have to prove that $\big(F_{\tau_1\to\tau_2}(\overrightarrow{u^{-+}})(d)(e'), M'[N'/x]\big) = \big(F_{\tau_2}(\overrightarrow{u^{-+}}) \circ d \circ F_{\tau_1}(\overrightarrow{u^{+-}})(e'), M'[N'/x]\big) \in F_{\tau_2}(\vec{S})$ holds. Since $(e',N') \in F_{\tau_1}(\vec{S})$ implies $\big(F_{\tau_1}(\overrightarrow{u^{+-}})(e'), N'\big) \in F_{\tau_1}(\vec{R})$ by induction hypothesis, we obtain $\big(d \circ F_{\tau_1}(\overrightarrow{u^{+-}})(e'), M'[N'/x]\big) \in F_{\tau_2}(\vec{R})$ by instantiating e by $F_{\tau_1}(\overrightarrow{u^{+-}})(e')$ and N by N'. Furthermore, applying the induction hypothesis to the functor F_{τ_2} yields $\big(F_{\tau_1\to\tau_2}(\overrightarrow{u^{-+}})(d)(e'), M[N'/x]\big) = \big(F_{\tau_2}(\overrightarrow{u^{-+}}) \circ d \circ F_{\tau_1}(\overrightarrow{u^{+-}})(e'), M[N'/x]\big) \in F_{\tau_2}(\vec{S})$. This finally entails $\big(F_{\tau_1\to\tau_2}(\overrightarrow{u^{-+}})(d), M\big) \in F_{\tau_1\to\tau_2}(\vec{S})$.

The definition of $F_{\mu\beta.\tau'}(\vec{R})$ is more difficult. As done before, we restrict ourselves to just two parameters corresponding to the negative and positive occurrence of one single free type variable α_1 in $\mu\alpha.\tau'$.

Let $\mathcal{P}$ be the set of all pairs (ρ^-,ρ^+) of functions

$$\rho^-,\rho^+ \colon \mathsf{Rel}_{\tau_1'} \times \mathsf{Rel}_{\tau_1'} \to \mathsf{Rel}_{\mu\beta.\tau'}.$$

Since $\mathsf{Rel}_{\mu\beta.\tau'}$ is closed under intersections, the set $\mathcal{P}$ is a complete lattice, the order being given by

$$(\rho_1^-,\rho_1^+) \le (\rho_2^-,\rho_2^+) \iff \forall R.\ \rho_2^-(R,R) \subseteq \rho_1^-(R,R),\ \rho_1^+(R,R) \subseteq \rho_2^+(R,R).$$

Consider the function $\psi\colon \mathcal{P} \to \mathcal{P}$ defined by $\psi(\rho^-,\rho^+) = \big(\psi^-(\rho^-,\rho^+), \psi^+(\rho^-,\rho^+)\big)$ with

$$\begin{aligned}\psi^-(\rho^-,\rho^+)(R,R) \;:=\; \big\{(d,M) \bigm| {} & d = \bot \text{ or} \\ & M \Downarrow \mathbf{fold}\, V,\ (\mathsf{unfold}\, d, V) \in F_{\tau'}\big(R,R,\rho^+(R,R),\rho^-(R,R)\big)\big\},\end{aligned}$$

and

$$\begin{aligned}\psi^+(\rho^-,\rho^+)(R,R) \;:=\; \big\{(d,M) \bigm| {} & d = \bot \text{ or} \\ & M \Downarrow \mathbf{fold}\, V,\ (\mathsf{unfold}\, d, V) \in F_{\tau'}\big(R,R,\rho^-(R,R),\rho^+(R,R)\big)\big\}.\end{aligned}$$

We first check that ψ is monotone. Let (ρ_1^-,ρ_1^+) and (ρ_2^-,ρ_2^+) be pairs of functions in $\mathcal{P}$ with $(\rho_1^-,\rho_1^+) \le (\rho_2^-,\rho_2^+)$. Then $\rho_2^-(R,R) \subseteq \rho_1^-(R,R)$ and $\rho_1^+(R,R) \subseteq \rho_2^+(R,R)$ follow by definition for each relation $R \in \mathsf{Rel}_{\tau_1'}$. Since

$$\begin{aligned}\mathsf{id}_{[\![\tau'[\mu\beta.\tau'[\tau_1'/\alpha_1]/\beta,\tau_1'/\alpha_1]]\!]} = {} & F_{\tau'}(\mathsf{id}_{[\![\tau_1']\!]}, \mathsf{id}_{[\![\tau_1']\!]}, \mathsf{id}_{[\![\mu\beta.\tau'[\tau_1'/\alpha_1]]\!]}, \mathsf{id}_{[\![\mu\beta.\tau'[\tau_1'/\alpha_1]]\!]}) \colon \\ & F_{\tau'}\big(R,R,\rho_2^+(R,R),\rho_2^-(R,R)\big) \subset F_{\tau'}\big(R,R,\rho_1^+(R,R),\rho_1^-(R,R)\big)\end{aligned}$$

and

$$\begin{aligned}\mathsf{id}_{[\![\tau'[\mu\beta.\tau'[\tau_1'/\alpha_1]/\beta,\tau_1'/\alpha_1]]\!]} = {} & F_{\tau'}(\mathsf{id}_{[\![\tau_1']\!]}, \mathsf{id}_{[\![\tau_1']\!]}, \mathsf{id}_{[\![\mu\beta.\tau'[\tau_1'/\alpha_1]]\!]}, \mathsf{id}_{[\![\mu\beta.\tau'[\tau_1'/\alpha_1]]\!]}) \colon \\ & F_{\tau'}\big(R,R,\rho_1^-(R,R),\rho_1^+(R,R)\big) \subset F_{\tau'}\big(R,R,\rho_2^-(R,R),\rho_2^+(R,R)\big)\end{aligned}$$

are true by induction hypothesis, $\psi^-(\rho_2^-,\rho_2^+)(R,R) \le \psi^-(\rho_1^-,\rho_1^+)(R,R)$ as well as $\psi^+(\rho_1^-,\rho_1^+)(R,R) \le \psi^-(\rho_2^-,\rho_2^+)(R,R)$ follow. This entails $\psi(\rho_1^-,\rho_1^+) \le \psi(\rho_2^-,\rho_2^+)$.

Since $\mathcal{P}$ is a complete lattice, ψ has a least fixed point (φ^-, φ^+) satisfying the equations $\varphi^- = \psi^-(\varphi^-, \varphi^+)$ and $\varphi^+ = \psi^+(\varphi^-, \varphi^+)$. To define the action of the functor $F_{\mu\beta.\tau'}$ on (R, R) it suffices to prove $\varphi^- = \varphi^+$, since then $F_{\mu\beta.\tau'}(R, R)$ can be chosen as $F_{\mu\beta.\tau'}(R, R) := \varphi^-(R, R) = \varphi^+(R, R)$.

Because (φ^-, φ^+) is a fixed point of ψ, so is (φ^+, φ^-). Moreover, (φ^-, φ^+) is the least fixed point of ψ, thus, $(\varphi^-, \varphi^+) \leq (\varphi^+, \varphi^-)$ follows.

For the reverse direction, consider the ω-complete subset

$$\mathcal{M} \;:=\; \{(e^-, e^+) \mid e^-, e^+ \colon \varphi^-(R, R) \subset \varphi^+(R, R)\}$$

of $\left[F_{\mu\beta.\tau'}([\![\tau_1']\!], [\![\tau_1']\!]) \hookrightarrow F_{\mu\beta.\tau'}([\![\tau_1']\!], [\![\tau_1']\!])\right] \times \left[F_{\mu\beta.\tau'}([\![\tau_1']\!], [\![\tau_1']\!]) \hookrightarrow F_{\mu\beta.\tau'}([\![\tau_1']\!], [\![\tau_1']\!])\right]$. In the following, we show that $\mathcal{M}$ is also closed under the function $\Phi = (\Phi^-, \Phi^+)$ as defined in Proposition 2.3. Assume $e^-, e^+ \colon \varphi^-(R, R) \subset \varphi^+(R, R)$, we need to prove that

$$\Phi^-(e^-, e^+) \;=\; \mathsf{fold} \circ F_{\tau'}(\mathsf{id}_{[\![\tau_1']\!]}, \mathsf{id}_{[\![\tau_1']\!]}, e^+, e^-) \circ \mathsf{unfold} \colon \varphi^-(R, R) \subset \varphi^+(R, R)$$

and

$$\Phi^+(e^-, e^+) \;=\; \mathsf{fold} \circ F_{\tau'}(\mathsf{id}_{[\![\tau_1']\!]}, \mathsf{id}_{[\![\tau_1']\!]}, e^-, e^+) \circ \mathsf{unfold} \colon \varphi^-(R, R) \subset \varphi^+(R, R)$$

are true. Both cases follow exactly the same pattern, therefore, we restrict ourselves to the first one. Let us suppose $(d, M) \in \varphi^-(R, R) = \psi^-(\varphi^-, \varphi^+)(R, R)$. In the case of $d = \bot$, $\mathsf{fold} \circ F_{\tau'}(\mathsf{id}_{[\![\tau_1']\!]}, \mathsf{id}_{[\![\tau_1']\!]}, e^+, e^-) \circ \mathsf{unfold}(d) = \bot$ follows, which entails the claim trivially. Otherwise, the term M evaluates to a value $\mathbf{fold}\, V$ such that $\big(\mathsf{unfold}(d), V\big) \in F_{\tau'}\big(R, R, \varphi^+(R, R), \varphi^-(R, R)\big)$ holds. By induction hypothesis, the relations $F_{\tau'}\big(R, R, \varphi^+(R, R), \varphi^-(R, R)\big)$ and $F_{\tau'}\big(R, R, \varphi^-(R, R), \varphi^+(R, R)\big)$ are defined in such a way that

$$F_{\tau'}(u_1^-, u_1^+, u_2^-, u_2^+) \colon F_{\tau'}\big(R, R, \varphi^+(R, R), \varphi^-(R, R)\big) \subset F_{\tau'}\big(R, R, \varphi^-(R, R), \varphi^+(R, R)\big)$$

holds for all $u_1^-, u_1^+ \colon R \subset R$ and $u_2^-, u_2^+ \colon \varphi^-(R, R) \subset \varphi^+(R, R)$. In particular, choosing $u_1^- := u_1^+ := \mathsf{id}_{[\![\tau_1']\!]}$, $u_2^- := e^+$ and $u_2^+ := e^-$ yields

$$F_{\tau'}(\mathsf{id}_{[\![\tau_1']\!]}, \mathsf{id}_{[\![\tau_1']\!]}, e^+, e^-)\big(\mathsf{unfold}(d)\big) \in F_{\tau'}\big(R, R, \varphi^-(R, R), \varphi^+(R, R)\big).$$

Therefore,

$$\mathsf{fold} \circ F_{\tau'}(\mathsf{id}_{[\![\tau_1']\!]}, \mathsf{id}_{[\![\tau_1']\!]}, e^+, e^-) \circ \mathsf{unfold}(d) \in \psi^+(\varphi^-, \varphi^+)(R, R) = \varphi^+(R, R)$$

follows, which finally entails $\Phi(e^-, e^+) = \big(\Phi^-(e^-, e^+), \Phi^+(e^-, e^+)\big) \in \mathcal{M}$. Furthermore, since the set $\mathcal{M}$ is ω-complete, $(\mathsf{id}_{[\![\tau_1']\!]}, \mathsf{id}_{[\![\tau_1']\!]}) = \bigsqcup^{\uparrow}_{n \in \mathbb{N}} \Phi^n(\bot, \bot) \in \mathcal{M}$ follows by Proposition 2.3 and Proposition 3.11, respectively. This leads to $\varphi^-(R, R) \subseteq \varphi^+(R, R)$ which implies $(\varphi^+, \varphi^-) \leq (\varphi^-, \varphi^+)$. As we have already seen, $(\varphi^-, \varphi^+) \leq (\varphi^+, \varphi^-)$ holds as well, therefore, we get $\varphi^- = \varphi^+$. Hence, the relation $F_{\mu\beta.\tau'}(R, R)$ can be defined by $\varphi^-(R, R) = \varphi^+(R, R)$. More precisely, $F_{\mu\beta.\tau'}(R, R)$ satisfies the property

$$F_{\mu\beta.\tau'}(R, R) \;=\; \{(d, M) \mid d = \bot \text{ or } M \Downarrow \mathbf{fold}\, V,\; \big(\mathsf{unfold}(d), V\big) \in F_{\tau'[\mu\beta.\tau'/\beta]}(R, R)\}.$$

It remains to show that, if $u^-, u^+ \colon [\![\tau_1']\!] \hookrightarrow [\![\tau_1']\!]$ are morphisms in $SD_\perp$ with $u^- \colon S \subset R$ and $u^+ \colon R \subset S$ for relations $R, S \in \mathsf{Rel}_{\tau_1'}$, then $F_{\mu\beta.\tau'}(u^-, u^+) \colon F_{\mu\beta.\tau'}(R,R) \subset F_{\mu\beta.\tau'}(S,S)$ holds as well. Before we are able to prove this, we first need to introduce some more notations: Given two morphisms $f \colon [\![\tau'[\mu\beta.\tau'[\tau_1'/\alpha_1]/\beta, \tau_1'/\alpha_1]]\!] \hookrightarrow [\![\mu\beta.\tau[\tau_1'/\alpha_1]]\!]$ and $g \colon [\![\mu\beta.\tau[\tau_1'/\alpha_1]]\!] \hookrightarrow [\![\tau'[\mu\beta.\tau'[\tau_1'/\alpha_1]/\beta, \tau_1'/\alpha_1]]\!]$ in $SD_\perp$ as well a relation $R \in \mathsf{Rel}_{\tau_1'}$, we write

$$f \colon F_{\tau'}\big(R, R, F_{\mu\beta.\tau'}(R,R), F_{\mu\beta.\tau'}(R,R)\big) \subset F_{\mu\beta.\tau'}(R,R)$$

if $\big(f(d), \mathbf{fold}\, M\big) \in F_{\mu\beta.\tau'}(R,R)$ is true for all elements d and SFL terms M with $(d, M) \in F_{\tau'}\big(R, R, F_{\mu\beta.\tau'}(R,R), F_{\mu\beta.\tau'}(R,R)\big)$, and

$$g \colon F_{\mu\beta.\tau'}(R,R) \subset F_{\tau'}\big(R, R, F_{\mu\beta.\tau'}(R,R), F_{\mu\beta.\tau'}(R,R)\big)$$

if $(d, M) \in F_{\mu\beta.\tau'}(R,R)$ implies $\big(g(d), V\big) \in F_{\tau'}\big(R, R, F_{\mu\beta.\tau'}(R,R), F_{\mu\beta.\tau'}(R,R)\big)$ for some value $\mathbf{fold}\, V$ with $M \Downarrow \mathbf{fold}\, V$. Note that the family $(\mathsf{Rel}_\tau)_{\tau \text{ SFL type}}$ with the extended definition of $\subset$ is still a relational structure on the subcategory of type denotations of $SD_\perp$. According to the definition of $F_{\mu\beta.\tau'}(R,R)$,

$$\mathsf{fold} \colon F_{\tau'}\big(R, R, F_{\mu\beta.\tau'}(R,R), F_{\mu\beta.\tau'}(R,R)\big) \subset F_{\mu\beta.\tau'}(R,R)$$

and

$$\mathsf{unfold} \colon F_{\mu\beta.\tau'}(R,R) \subset F_{\tau'}\big(R, R, F_{\mu\beta.\tau'}(R,R), F_{\mu\beta.\tau'}(R,R)\big)$$

are easy to verify. Since

$$\begin{aligned} &F_{\tau'}(u^-, u^+, \mathsf{id}_{[\![\mu\beta.\tau'[\tau_1'/\alpha_1]]\!]}, \mathsf{id}_{[\![\mu\beta.\tau'[\tau_1'/\alpha_1]]\!]}) \colon \\ &\qquad F_{\tau'}\big(R, R, F_{\mu\beta.\tau'}(R,R), F_{\mu\beta.\tau'}(R,R)\big) \subset F_{\tau'}\big(S, S, F_{\mu\beta.\tau'}(R,R), F_{\mu\beta.\tau'}(R,R)\big) \end{aligned}$$

and

$$\begin{aligned} &F_{\tau'}(u^+, u^-, \mathsf{id}_{[\![\mu\beta.\tau'[\tau_1'/\alpha_1]]\!]}, \mathsf{id}_{[\![\mu\beta.\tau'[\tau_1'/\alpha_1]]\!]}) \colon \\ &\qquad F_{\tau'}\big(S, S, F_{\mu\beta.\tau'}(R,R), F_{\mu\beta.\tau'}(R,R)\big) \subset F_{\tau'}\big(R, R, F_{\mu\beta.\tau'}(R,R), F_{\mu\beta.\tau'}(R,R)\big) \end{aligned}$$

hold by induction hypothesis,

$$\begin{aligned} &F_{\tau'}(u^-, u^+, \mathsf{id}_{[\![\mu\beta.\tau'[\tau_1'/\alpha_1]]\!]}, \mathsf{id}_{[\![\mu\beta.\tau'[\tau_1'/\alpha_1]]\!]}) \circ \mathsf{unfold} \colon \\ &\qquad\qquad F_{\mu\beta.\tau'}(R,R) \subset F_{\tau'}\big(S, S, F_{\mu\beta.\tau'}(R,R), F_{\mu\beta.\tau'}(R,R)\big) \end{aligned}$$

and

$$\begin{aligned} &\mathsf{fold} \circ F_{\tau'}(u^+, u^-, \mathsf{id}_{[\![\mu\beta.\tau'[\tau_1'/\alpha_1]]\!]}, \mathsf{id}_{[\![\mu\beta.\tau'[\tau_1'/\alpha_1]]\!]}) \colon \\ &\qquad\qquad F_{\tau'}\big(S, S, F_{\mu\beta.\tau'}(R,R), F_{\mu\beta.\tau'}(R,R)\big) \subset F_{\mu\beta.\tau'}(R,R) \end{aligned}$$

are true by the definition of relational structures. Moreover, $F_{\mu\beta.\tau'}(R,R)$ is ω-admissible, therefore, we are able to apply Proposition 2.5 and Proposition 3.11, respectively, by instantiating $A = B = [\![\mu\beta.\tau'[\tau_1/\alpha_1]]\!]$, $F = F_{\mu\beta.\tau'}$, $G = F_{\tau'}$, $D^- = D^+ = [\![\tau_1']\!]$, $R^- = R^+ = S$

and $P = Q = F_{\mu\beta.\tau'}(R,R)$ as well as $f = F_{\tau'}(u^-, u^+, \mathsf{id}_{[\![\mu\beta.\tau'[\tau_1'/\alpha_1]]\!]}, \mathsf{id}_{[\![\mu\beta.\tau'[\tau_1'/\alpha_1]]\!]}) \circ \mathsf{unfold}$ and $g = \mathsf{fold} \circ F_{\tau'}(u^+, u^-, \mathsf{id}_{[\![\mu\beta.\tau'[\tau_1'/\alpha_1]]\!]}, \mathsf{id}_{[\![\mu\beta.\tau'[\tau_1'/\alpha_1]]\!]})$. Thus,

$$h = F_{\mu\beta.\tau'}(u^-, u^+) \colon F_{\mu\beta.\tau'}(R,R) \subset F_{\mu\beta.\tau'}(S,S)$$

and

$$k = F_{\mu\beta.\tau'}(u^+, u^-) \colon F_{\mu\beta.\tau'}(S,S) \subset F_{\mu\beta.\tau'}(R,R)$$

follow.

This completes the first part of the proof of Proposition 5.6. For every SFL type τ with free variables $\alpha_1, \dots, \alpha_n$ we have constructed a relation $F_\tau(\vec{R})$, which is dependent on relations $R_1, \dots, R_n$. Therefore, we can now define a family $(\lhd_\tau)_{\tau \text{ closed SFL type}}$ of relations $\lhd_\tau \subseteq [\![\tau]\!] \times \Lambda^0_\tau$ by setting

$$\lhd_\tau \ := \ F_\tau()$$

satisfying the following property as required in the beginning of the proof:

$$d \lhd_\tau M \quad \Longleftrightarrow \quad d = \bot_{[\![\tau]\!]} \ \text{ or } \ (M \Downarrow V \text{ and } d \lhd_\tau V)$$

where the relation $\lhd$ on values is given by:

- $d \lhd_{\tau_1 \times \tau_2} \langle M_1, M_2 \rangle$ if $\mathsf{pr}_1(d) \lhd_{\tau_1} M_1$, $\mathsf{pr}_2(d) \lhd_{\tau_2} M_2$,
- $d \lhd_{\sigma_1 \otimes \sigma_2} (V_1, V_2)$ if $\widetilde{\mathsf{pr}_1}(d) \lhd_{\sigma_1} V_1$, $\widetilde{\mathsf{pr}_2}(d) \lhd_{\sigma_2} V_2$,
- $d \lhd_{\sum_{i=1}^n \tau_i} \mathbf{in}_i\, M$ if $d = \mathsf{in}_i\, e$ and $e \lhd_{\tau_i} M$,
- $d \lhd_{\sigma_1 \oplus \sigma_2} \widetilde{\mathbf{inl}}\, V$ if $d = \widetilde{\mathsf{inl}}\, e$ and $e \lhd_{\sigma_1} V$,
- $d \lhd_{\sigma_1 \oplus \sigma_2} \widetilde{\mathbf{inr}}\, V$ if $d = \widetilde{\mathsf{inr}}\, e$ and $e \lhd_{\sigma_2} V$,
- $d \lhd_{\tau_1 \to \tau_2} \lambda x{:}\tau_1.M$ if $d(e) \lhd_{\tau_2} M[N/x]$ holds for all e, N with $e \lhd_{\tau_1} N$,
- $d \lhd_{\sigma \circ\!\!\to \tau} \widetilde{\lambda} x{:}\sigma.M$ if $\widetilde{\mathsf{eval}}(d, e) \lhd_\tau (\widetilde{\lambda} x{:}\sigma.M)\tilde{\ } N$ holds for all e, N with $e \lhd_\sigma N$,
- $d \lhd_{\tau_\bot} \mathbf{up}\, M$ if $\mathsf{down}(d) \lhd_\tau M$,
- $d \lhd_{\mu\alpha.\tau} \mathbf{fold}\, V$ if $\mathsf{unfold}(d) \lhd_{\tau[\mu\alpha.\tau/\alpha]} V$.

This enables us to prove Proposition 5.6 by verifying the following more general claim:

> *Let M be an SFL term of type τ such that all free variables of M are contained in the context Γ. Moreover, let $d_1 \in [\![\tau_1]\!]$, ..., $d_n \in [\![\tau_n]\!]$ be elements and $\Gamma \vdash N_1 : \tau_1, \dots, \Gamma \vdash N_n : \tau_n$ be terms-in-contexts with $d_i \lhd_{\tau_i} N_i$. Then $[\![\Gamma \vdash M : \tau]\!](\vec{d}) \lhd_\tau M[\vec{N}/\vec{x}]$ holds.*

That implies $[\![M]\!] \lhd_\tau M$ for every closed term M of type τ, which finally entails $[\![M]\!] = \bot_{[\![\tau]\!]}$ or $M \Downarrow$.

The proof is an induction on the structure of M. Most cases follow a certain pattern, therefore, we focus just on the more interesting ones:

$M = \mathbf{pr_1}\, M'$: If $[\![\Gamma \vdash M' : \tau_1 \times \tau_2]\!](\vec{d}) = \bot$ is true, then $[\![\Gamma \vdash \mathbf{pr_1}\, M' : \tau_1]\!](\vec{d}) = \bot$ follows. This leads directly to $[\![\Gamma \vdash \mathbf{pr_1}\, M' : \tau_1]\!](\vec{d}) \lhd_{\tau_1} \mathbf{pr_1}\, M'[\vec{N}/\vec{x}]$. Otherwise, $[\![\Gamma \vdash M' : \tau_1 \times \tau_2]\!](\vec{d}) \lhd_{\tau_1 \times \tau_2} M'[\vec{N}/\vec{x}]$ (holds by induction hypothesis) entails that there are some SFL terms N_1, N_2 with $M'[\vec{N}/\vec{x}] \Downarrow \langle N_1, N_2 \rangle$ and $\langle [\![\Gamma \vdash \mathbf{pr_1}\, M' : \tau_1]\!](\vec{d}), [\![\Gamma \vdash \mathbf{pr_2}\, M' : \tau_2]\!](\vec{d}) \rangle = [\![\Gamma \vdash M' : \tau_1 \times \tau_2]\!](\vec{d}) \lhd_{\tau_1 \times \tau_2} \langle N_1, N_2 \rangle$. This implies $[\![\Gamma \vdash \mathbf{pr_1}\, M' : \tau_1]\!](\vec{d}) \lhd_{\tau_1} N_1$. The case $[\![\Gamma \vdash \mathbf{pr_1}\, M' : \tau_1]\!](\vec{d}) = \bot$ is trivial, thus, assume that there is a value V with $N_1 \Downarrow V$ and $[\![\Gamma \vdash \mathbf{pr_1}\, M' : \tau_1]\!](\vec{d}) \lhd_{\tau_1} V$. Because of $\mathbf{pr_1}\, M'[\vec{N}/\vec{x}] \Downarrow V$, this proves $[\![\Gamma \vdash \mathbf{pr_1}\, M' : \tau_1]\!](\vec{d}) \lhd_{\tau_1} \mathbf{pr_1}\, M'[\vec{N}/\vec{x}]$.

$M = (M_1, M_2)$: By induction hypothesis, $[\![\Gamma \vdash M_i : \sigma_i]\!](\vec{d}) \lhd_{\sigma_i} M_i[\vec{N}/\vec{x}]$ holds for $i = 1, 2$. If $[\![\Gamma \vdash M_i : \sigma_i]\!](\vec{d}) = \bot$ is true for $i = 1$ or $i = 2$, then $[\![\Gamma \vdash (M_1, M_2) : \sigma_1 \otimes \sigma_2]\!](\vec{d}) = \bot$ follows. Therefore, suppose that there are values V_1, V_2 with $M_i[\vec{N}/\vec{x}] \Downarrow V_i$ and $[\![\Gamma \vdash M_i : \sigma_i]\!](\vec{d}) \lhd_{\sigma_i} V_i$. Then, $(M_1, M_2)[\vec{N}/\vec{x}] = (M_1[\vec{N}/\vec{x}], M_2[\vec{N}/\vec{x}]) \Downarrow (V_1, V_2)$ and, hence, $[\![\Gamma \vdash (M_1, M_2) : \sigma_1 \otimes \sigma_2]\!](\vec{d}) \lhd_{\sigma_1 \otimes \sigma_2} (M_1, M_2)[\vec{N}/\vec{x}]$ follows.

$M = \widetilde{\mathbf{case}}\ M'\ \mathbf{of}\ \widetilde{\mathbf{inl}}\, y_1 \Rightarrow M_1\, ; \widetilde{\mathbf{inr}}\, y_2 \Rightarrow M_2$: Firstly, $[\![\Gamma \vdash M' : \sigma_1 \oplus \sigma_2]\!](\vec{d}) \lhd_{\sigma_1 \oplus \sigma_2} M'[\vec{N}/\vec{x}]$ holds by induction hypothesis. The case $[\![\Gamma \vdash M' : \sigma_1 \oplus \sigma_2]\!](\vec{d}) = \bot$ leads to $[\![\Gamma \vdash M : \tau]\!](\vec{d}) = \bot$ and is therefore trivial. Hence, assume (w.l.o.g.) that there is a value V with $M'[\vec{N}/\vec{x}] \Downarrow \widetilde{\mathbf{inl}}\, V$ and $[\![\Gamma \vdash M' : \sigma_1 \oplus \sigma_2]\!](\vec{d}) = \widetilde{\mathsf{inl}}\ e \lhd_{\sigma_1 \oplus \sigma_2} \widetilde{\mathbf{inl}}\, V$. By induction hypothesis, $[\![\Gamma, y_1{:}\sigma_1 \vdash M_1 : \tau]\!](\vec{d}, e) \lhd_\tau M_1[\vec{N}/\vec{x}, V/y_1]$ holds. If $[\![\Gamma, y_1{:}\sigma_1 \vdash M_1 : \tau]\!](\vec{d}, e)$ is the bottom element, then so is $[\![\Gamma \vdash M : \tau]\!](\vec{d})$. Otherwise, there is a value W with $M_1[\vec{N}/\vec{x}, V/y_1] \Downarrow W$ and $[\![\Gamma \vdash M : \tau]\!](\vec{d}) = [\![\Gamma, y_1{:}\sigma_1 \vdash M_1 : \tau]\!](\vec{d}, e) \lhd_\tau W$. Because of $M \Downarrow W$, this completes the proof.

$M = \mathbf{down}\, M'$: This time, the induction hypothesis yields $[\![\Gamma \vdash M' : \tau_\bot]\!](\vec{d}) \lhd_{\tau_\bot} M'[\vec{N}/\vec{x}]$. The case $[\![\Gamma \vdash M' : \tau_\bot]\!](\vec{d}) = \bot$ is trivial, so suppose that $M'[\vec{N}/\vec{x}]$ evaluates to a value $\mathbf{up}\, N$ with $[\![\Gamma \vdash M' : \tau_\bot]\!](\vec{d}) \lhd_{\tau_\bot} \mathbf{up}\, N$. This entails $[\![\Gamma \vdash \mathbf{down}\, M' : \tau]\!](\vec{d}) \lhd_\tau N$. Suppose $[\![\Gamma \vdash \mathbf{down}\, M' : \tau]\!](\vec{d}) \neq \bot$ holds, then N evaluates to some value V with $[\![\Gamma \vdash \mathbf{down}\, M' : \tau]\!](\vec{d}) \lhd_\tau V$. Because of $\mathbf{down}\, M'[\vec{N}/\vec{x}] \Downarrow V$, this leads to $[\![\Gamma \vdash \mathbf{down}\, M' : \tau]\!](\vec{d}) \lhd_\tau M'[\vec{N}/\vec{x}]$.

$M = M_1 M_2$: By induction hypothesis, we know $[\![\Gamma \vdash M_1 : \tau' \to \tau]\!](\vec{d}) \lhd_{\tau' \to \tau} M_1[\vec{N}/\vec{x}]$. The case $[\![\Gamma \vdash M_1 : \tau' \to \tau]\!](\vec{d}) = \bot$ is trivial so assume there is a term N with $M_1[\vec{N}/\vec{x}] \Downarrow \lambda y{:}\tau.N$ and $[\![\Gamma \vdash M_1 : \tau' \to \tau]\!](\vec{d}) \lhd_{\tau' \to \tau} \lambda y{:}\tau.N$. Because of $[\![\Gamma \vdash M_2 : \tau']\!](\vec{d}) \lhd_{\tau'} M_2[\vec{N}/\vec{x}]$ (holds by induction hypothesis as well), $[\![\Gamma \vdash M_1 M_2 : \tau]\!](\vec{d}) = \big([\![\Gamma \vdash M_1 : \tau' \to \tau]\!](\vec{d})\big)\big([\![\Gamma \vdash M_2 : \tau']\!](\vec{d})\big) \lhd_\tau N\big[M_2[\vec{N}/\vec{x}]/y\big]$ follows. Now suppose $[\![\Gamma \vdash M_1 M_2 : \tau]\!](\vec{d}) \neq \bot$, then there is a value V with $N\big[M_2[\vec{N}/\vec{x}]/y\big] \Downarrow V$ and $[\![\Gamma \vdash M_1 M_2 : \tau]\!](\vec{d}) \lhd_\tau V$. This entails $(M_1 M_2)[\vec{N}/\vec{x}] \Downarrow V$, which completes this case.

$M = M_1 \tilde{\ } M_2$: Assume $[\![\Gamma \vdash M_1 \tilde{\ } M_2 : \tau]\!](\vec{d}) \neq \bot$ (which implies $[\![\Gamma \vdash M_1 : \sigma \multimapdotboth \tau]\!](\vec{d}) \neq \bot$ and $[\![\Gamma \vdash M_2 : \sigma]\!](\vec{d}) \neq \bot$). Since $[\![\Gamma \vdash M_1 : \sigma \circ\!\!\to \tau]\!](\vec{d}) \lhd_{\sigma \circ\!\to \tau} M_1[\vec{N}/\vec{x}]$ holds by induction hypothesis, there is an SFL term N with $M_1[\vec{N}/\vec{x}] \Downarrow \widetilde{\lambda} y{:}\sigma.N$ and $[\![\Gamma \vdash M_1 : \sigma \circ\!\!\to \tau]\!](\vec{d}) \lhd_{\sigma \circ\!\to \tau} \widetilde{\lambda} y{:}\sigma.N$. Furthermore, $[\![\Gamma \vdash M_2 : \sigma]\!](\vec{d}) \lhd_\sigma M_2[\vec{N}/\vec{x}]$ holds by induction hypothesis as well, hence, there is a value V such that $M_2[\vec{N}/\vec{x}] \Downarrow V$ and $[\![\Gamma \vdash M_1 : \sigma]\!](\vec{d}) \lhd_\sigma V$ are true. This leads to $[\![\Gamma \vdash M_1 \tilde{\ } M_2 : \tau]\!](\vec{d}) =$

$\big(\llbracket\Gamma \vdash M_1 : \sigma \hookrightarrow \tau\rrbracket(\vec{d})\big)\big(\llbracket\Gamma \vdash M_2 : \sigma\rrbracket(\vec{d})\big) \lhd_\tau (N[V/y])$. Bearing in mind the assumption $\llbracket\Gamma \vdash M_1\,\tilde{}\,M_2 : \tau\rrbracket(\vec{d}) \neq \bot$, this entails that there is a value W with $N[V/y] \Downarrow W$ and $\llbracket\Gamma \vdash M_1\,\tilde{}\,M_2 : \tau\rrbracket(\vec{d}) \lhd_\tau W$. This finishes this part of the proof because of $M_1\,\tilde{}\,M_2 \Downarrow W$.

$M = \mathbf{unfold}\, M'$: Firstly, $\llbracket\Gamma \vdash M' : \mu\alpha.\tau'\rrbracket(\vec{d}) \lhd_{\mu\alpha.\tau'} M'[\vec{N}/\vec{x}]$ holds by induction hypothesis. Assume $\llbracket\Gamma \vdash M' : \mu\alpha.\tau'\rrbracket(\vec{d}) \neq \bot$ (otherwise the claim holds trivially), then $M'[\vec{N}/\vec{x}]$ evaluates to a value $\mathbf{fold}\, V$ with $\llbracket\Gamma \vdash M' : \mu\alpha.\tau'\rrbracket(\vec{d}) \lhd_{\mu\alpha.\tau'} \mathbf{fold}\, V$. This implies $\llbracket\Gamma \vdash \mathbf{unfold}\, M' : \tau'[\mu\alpha.\tau'/\alpha]\rrbracket(\vec{d}) \lhd_{\tau'[\mu\alpha.\tau'\alpha]} V$, which completes the proof because of $(\mathbf{unfold}\, M')[\vec{N}/\vec{x}] = \mathbf{unfold}\,(M'[\vec{N}/\vec{x}]) \Downarrow V$. ☐

This entails the following adequacy result:

THEOREM 5.7. ***(Computational adequacy)*** *Let M be a closed SFL term of type* **unit**. *Then M terminates if, and only if, $\llbracket M\rrbracket = \top$ holds.*

Proof. Using the Substitution Lemma 5.5, it is an easy induction to verify that $\llbracket M\rrbracket = \llbracket V\rrbracket$ is true if $M \Downarrow V$ holds. For type **unit**, all values are interpreted as $\top$. Thus, $M \Downarrow$ implies $\llbracket M\rrbracket = \top$.

On the other hand, if M diverges, then $\llbracket M\rrbracket = \top$ follows from Proposition 5.6. ☐

5.3 Full Abstraction

For the full abstraction result, we define one particular subcategory $\mathcal{C}$ of *SPIC*. It is essentially the same definition as in [OR95b] and [RS97]. Objects in $\mathcal{C}$ are pairs $([\Gamma], S^\Gamma)$ where $\Gamma = x_1{:}\tau_1, \ldots, x_m{:}\tau_m$ is a context consisting of finite types $\tau_1, \ldots, \tau_m$, $[\Gamma]$ is the product $\llbracket\tau_1\rrbracket \times \cdots \times \llbracket\tau_m\rrbracket$ and S^Γ is the sequentiality system that is given by

$$S_n^\Gamma \;=\; \{\llbracket\Gamma \vdash M : n \odot \mathbf{unit}\rrbracket \mid \Gamma \vdash M : n \odot \mathbf{unit} \text{ is an SFL term-in-context}\}$$

(where $0 \odot \mathbf{unit} = \mathbf{void}$ and $n \odot \mathbf{unit} = \overbrace{\mathbf{unit} \oplus \cdots \oplus \mathbf{unit}}^{n\text{-times}}$). It can easily be shown that this is indeed a sequentiality system on $[\Gamma]$. The morphisms in $\mathcal{C}$ are projections $\pi\colon ([\Gamma \cup \Delta], S^{\Gamma\cup\Delta}) \to ([\Gamma], S^\Gamma)$. It is not difficult to verify that this definition makes $\mathcal{C}$ a subcategory of *SPIC*. For every SFL type τ, we use $\llbracket\tau\rrbracket_{\mathcal{C}}^{[\Gamma]}$ as abbreviation for the relation $\llbracket\tau\rrbracket_{\mathcal{C}}^{([\Gamma],S^\Gamma)}$.

The key lemma for the Full Abstraction Theorem for SFL is the following:

LEMMA 5.8. *Let $\mathcal{C}$ be the category from above and let $p \in \llbracket\tau\rrbracket$ be an element of the denotation of a finite SFL type. Then p is related by $\llbracket\tau\rrbracket_{\mathcal{C}}^{\varnothing}$ if, and only if, there is a closed SFL term P with $\llbracket P\rrbracket = p$.*

Proof. We show the following more general claim:

> Let $\Gamma = x_1{:}\varrho_1, \ldots, x_n{:}\varrho_n$ be a context and $p\colon [\varrho_1, \ldots, \varrho_n] \to \llbracket\tau\rrbracket$ a function. Then $p \in \llbracket\tau\rrbracket_{\mathcal{C}}^{[\Gamma]}$ holds if, and only if, there is an SFL term P with $\llbracket\Gamma \vdash P : \tau\rrbracket = p$.

This proves the lemma by choosing $\Gamma = \varnothing$. The proof goes by induction on the structure of the type τ, the verifications for separated sum types and strict function types are similar to the ones for coalesced sum types and normal function types, respectively. The other nontrivial cases are:

$\tau = \tau'_\perp$: Assume $p \in [\![\tau'_\perp]\!]^{[\Gamma]}_{\mathcal{C}}$, then $\mathsf{down} \circ p \in [\![\tau']\!]^{[\Gamma]}_{\mathcal{C}}$ and $\big\{\{x \in [\Gamma] \mid p(x) \neq \perp\}\big\}^{-} \in S$ follow by definition. The latter entails $\left(\lambda\!\!\lambda x{:}[\Gamma].\begin{cases}1 & \text{if } p(x) \neq \perp \\ \perp & \text{if } p(x) = \perp\end{cases}\right) \in S_1 = \mathbf{2}^{[\Gamma]}_{\mathcal{C}} = [\![\mathbf{unit}]\!]^{[\Gamma]}_{\mathcal{C}}$. Furthermore, by induction hypothesis, there exist two SFL terms D and T (possibly containing $x_1, \dots, x_n$ as free variables) with $[\![\Gamma \vdash D : \tau']\!] = \mathsf{down} \circ p$ and $[\![\Gamma \vdash T : \mathbf{unit}]\!](x) = \perp_{\mathbf{unit}} \Leftrightarrow p(x) = \perp_{\tau_\perp}$. Thus, p can be defined by $(\widetilde{\lambda} x{:}\mathbf{unit}.D)\tilde{\ }T$.

For the reverse direction, suppose that p is definable by a term P. Then so is $\mathsf{down} \circ p$ by a term Q and $[\![Q]\!] = \mathsf{down} \circ p \in [\![\tau']\!]^{[\Gamma]}_{\mathcal{C}}$ follows by induction hypothesis. Moreover, the partition $\big\{\{x \in [\Gamma] \mid p(x) \neq \perp\}\big\}^{-}$ is denoted by $[\![\Gamma \vdash (\widetilde{\lambda} y{:}\tau'_\perp.*)\tilde{\ }P : \mathbf{unit}]\!]$, hence, it is an element in S. This implies $p = [\![\Gamma \vdash P : \tau'_\perp]\!] \in [\![\tau'_\perp]\!]^{[\Gamma]}_{\mathcal{C}}$.

$\tau = \sigma_1 \oplus \sigma_2$: Given $p \in [\![\sigma_1 \oplus \sigma_2]\!]^{[\Gamma]}_{\mathcal{C}}$, then there are functions $\varphi \in S_2$, $p_1 \in [\![\sigma_1]\!]^{[\Gamma]}_{\mathcal{C}}$ and $p_2 \in [\![\sigma_2]\!]^{[\Gamma]}_{\mathcal{C}}$ with $p(x) = \begin{cases}\widetilde{\mathsf{inl}} \circ p_1(x) & \text{if } \varphi(x) = 1 \\ \widetilde{\mathsf{inr}} \circ p_2(x) & \text{if } \varphi(x) = 2 \\ \perp & \text{if } \varphi(x) = \perp\end{cases}$. By induction hypothesis, φ, p_1 and p_2 are defined by terms Q, P_1 and P_2. Hence, p can be defined by $P := \widetilde{\mathbf{case}}\ Q\ \mathbf{of}\ \widetilde{\mathbf{inl}}\, x_1 \Rightarrow \widetilde{\mathbf{inl}}\, P_1\, ; \widetilde{\mathbf{inr}}\, x_2 \Rightarrow \widetilde{\mathbf{inr}}\, P_2$.

Conversely, let P be a term with $[\![\Gamma \vdash P : \sigma_1 \oplus \sigma_2]\!] = p$. Define the terms

$$\begin{aligned}
Q &:= \widetilde{\mathbf{case}}\ P\ \mathbf{of}\ \widetilde{\mathbf{inl}}\, x_1 \Rightarrow \widetilde{\mathbf{inl}}\, *\, ; \widetilde{\mathbf{inr}}\, x_2 \Rightarrow \widetilde{\mathbf{inr}}\, *, \\
P_1 &:= \widetilde{\mathbf{case}}\ P\ \mathbf{of}\ \widetilde{\mathbf{inl}}\, x_1 \Rightarrow x_1\, ; \widetilde{\mathbf{inr}}\, x_2 \Rightarrow \Omega_{\sigma_2}, \\
P_2 &:= \widetilde{\mathbf{case}}\ P\ \mathbf{of}\ \widetilde{\mathbf{inl}}\, x_1 \Rightarrow \Omega_{\sigma_1}\, ; \widetilde{\mathbf{inr}}\, x_2 \Rightarrow x_2,
\end{aligned}$$

then $\varphi := [\![\Gamma \vdash Q : \mathbf{unit} \oplus \mathbf{unit}]\!] \in S_2$ as well as $p_1 := [\![\Gamma \vdash P_1 : \sigma_1]\!] \in [\![\sigma_1]\!]^{[\Gamma]}_{\mathcal{C}}$ and $p_2 := [\![\Gamma \vdash P_2 : \sigma_2]\!] \in [\![\sigma_2]\!]^{[\Gamma]}_{\mathcal{C}}$ follow by induction hypothesis. This implies $p \in [\![\sigma_1 \oplus \sigma_2]\!]^{[\Gamma]}_{\mathcal{C}}$.

$\tau = \tau_1 \to \tau_2$: Suppose $p \in [\![\tau_1 \to \tau_2]\!]^{[\Gamma]}_{\mathcal{C}}$. Let $\mu\colon ([\Gamma] \cup [\![\tau_1]\!], S^{[\Gamma]\cup[\![\tau_1]\!]}) \xrightarrow{\mathcal{C}} ([\Gamma], S^{[\Gamma]})$ be the canonical projection. Since $m := [\![\Gamma, y{:}\tau_1 \vdash y : \tau_1]\!] \in [\![\tau_1]\!]^{[\Gamma]\cup[\![\tau_1]\!]}_{\mathcal{C}}$ is true by induction hypothesis, $q := \lambda\!\!\lambda x{:}[\Gamma]\cup[\![\tau_1]\!].\big(p(\mu x)\big)\big(m(x)\big) \in [\![\tau_2]\!]^{[\Gamma]\cup[\![\tau_1]\!]}_{\mathcal{C}}$ holds as well. Again by induction hypothesis, there is an SFL term Q with $[\![\Gamma, y{:}\tau_1 \vdash Q : \tau_2]\!] = q$. Choosing $P := \lambda y{:}\tau_1.Q$ yields $[\![\Gamma \vdash P : \tau_1 \to \tau_2]\!] = p$.

For the reverse direction, assume that there is a term P with $[\![\Gamma \vdash P : \tau_1 \to \tau_2]\!] = p$ where Γ contains types $\varrho_1, \dots, \varrho_n$. Let $\mu\colon ([\Gamma] \cup [\Delta], S^{[\Gamma]\cup[\Delta]}) \xrightarrow{\mathcal{C}} ([\Gamma], S^{[\Gamma]})$ be a morphism in $\mathcal{C}$ and assume $q \in [\![\tau_1]\!]^{[\Gamma]\cup[\Delta]}_{\mathcal{C}}$, we have to show

$$r \ := \ \lambda\!\!\lambda y{:}[\Gamma] \cup [\Delta].\Big(p\big(\mu(y)\big)\Big)\big(q(y)\big) \ \in \ [\![\tau_2]\!]^{[\Gamma]\cup[\Delta]}_{\mathcal{C}}.$$

By induction hypothesis, there is a term Q with $[\![\Gamma \cup \Delta \vdash Q : \tau_1]\!] = q$. The morphism μ is a simple projection, hence it is definable by a term-in-context $\Gamma \cup \Delta \vdash M : \varrho_1 \times \cdots \times \varrho_n$. Putting these terms together yields

$$[\![\Gamma \cup \Delta \vdash \big(P[\mathbf{pr_1} M/x_1, \ldots, \mathbf{pr_n} M/x_n]\big) Q : \tau_2]\!] = r \in [\![\tau_2]\!]_{\mathcal{C}}^{[\Gamma]\cup[\Delta]}$$

by induction hypothesis (where $\mathbf{pr_i}$ stands for the projection to the i-th component of a m-tuple). □

As an immediate consequence of Lemma 5.8, we get the following result:

COROLLARY 5.9. *Every element in the denotation of a finite SFL type is definable.*

Proof. Since all relations $[\![\tau]\!]_{\mathcal{C}}^{\varnothing}$ (for a subcategory $\mathcal{C}$ of *SPIC* and a finite SFL type τ) satisfy condition (R2), the constant functions $\lambda\!\!\lambda x{:}\varnothing.d\colon \varnothing \to [\![\tau]\!]$ are related by $[\![\tau]\!]_{\mathcal{C}}^{\varnothing}$ for every element $d \in [\![\tau]\!]$. By Lemma 5.8, these functions can be defined by SFL terms. □

Also projections and embeddings of a bilimit can be defined in SFL:

LEMMA 5.10. *Let $\mu\alpha.\tau$ be a closed type that is interpreted by the bilimit of a functor F_τ. Then, for every projection $p_n\colon [\![\mu\alpha.\tau]\!] \to F_\tau^{(n)}(\mathbf{1},\mathbf{1})$, there is a closed term P_n with $[\![P_n]\!] = p_n$ and, for every embedding $e_n\colon F_\tau^{(n)}(\mathbf{1},\mathbf{1}) \to [\![\mu\alpha.\tau]\!]$, there is a closed term E_n with $[\![E_n]\!] = e_n$.*

Proof. Let us first assume that τ is a finite type. We define a term-in-context

$$f : \psi \to \varrho,\, g : \varrho \to \psi \;\vdash\; M_\tau : \tau[\varrho/\alpha] \to \tau[\psi/\alpha]$$

by induction of the structure of τ:

- $M_\alpha := g$,
- $M_{\tau_1\times\tau_2} := \lambda x{:}\tau_1[\varrho/\alpha]\times\tau_2[\varrho/\alpha].\,\big\langle M_1(\mathbf{pr_1}\,x), M_2(\mathbf{pr_2}\,x)\big\rangle$,
- $M_{\sigma_1\otimes\sigma_2} := \lambda x{:}\sigma_1[\varrho/\alpha]\times\sigma_2[\varrho/\alpha].\,\big(M_1(\widetilde{\mathbf{pr_1}}\,x), M_2(\widetilde{\mathbf{pr_2}}\,x)\big)$,
- $M_{\sum_{i=1}^n \tau_i} := \lambda x{:}\sum_{i=1}^n \tau_i[\varrho/\alpha].\,\mathbf{case}\,x\,\mathbf{of}\,\mathbf{in}_1\,y_1 \Rightarrow M_{\tau_1}y_1; \ldots; \mathbf{in}_n\,y_n \Rightarrow M_{\tau_n}y_n$,
- $M_{\sigma_1\oplus\sigma_2} := \lambda x{:}\sigma_1[\varrho/\alpha]\oplus\sigma_2[\varrho/\alpha].\,\widetilde{\mathbf{case}}\,x\,\mathbf{of}\,\widetilde{\mathbf{inl}}\,y_1{\Rightarrow}\widetilde{\mathbf{inl}}\,(M_{\sigma_1}y_1)\,;\widetilde{\mathbf{inr}}\,y_2{\Rightarrow}\widetilde{\mathbf{inr}}\,(M_{\sigma_2}y_2)$,
- $M_{\tau_\perp} := \lambda x{:}(\tau[\varrho/\alpha])_\perp.\,\Big(\tilde{\lambda} y{:}(\tau[\varrho/\alpha])_\perp.\,\mathbf{up}\,\big(M_\tau(\mathbf{down}\,y)\big)\Big)^{\sim} x$,
- $M_{\tau_1\to\tau_2} := \lambda x{:}\tau_1[\varrho/\alpha]\to\tau_2[\varrho/\alpha].\lambda y{:}\tau_1[\psi/\alpha].\,M_{\tau_2}\big(x(M_{\tau_1}[f/g,g/f]y)\big)$,
- $M_{\sigma\multimap\tau'} := \lambda x{:}\sigma[\varrho/\alpha]\multimap\tau'[\varrho/\alpha].\lambda y{:}\sigma[\psi/\alpha].\,M_{\tau'}\big(x^{\sim}(M_\sigma[f/g,g/f]y)\big)$.

It is easy to check that $[\![M_\tau]\!](f,g) = F_\tau(f,g)$ is true for all SFL types ϱ, ψ and all morphisms $f\colon [\![\psi]\!] \to [\![\varrho]\!]$ and $g\colon [\![\varrho]\!] \to [\![\psi]\!]$. To deal with nested recursive types we need to consider the case that the type $\mu\alpha.\tau$ may contain several free variables. In that case, the corresponding functor has $2n+2$ arguments. Exactly in the same way as for finite types, we then can construct a term-in-context

$$\begin{array}{rc} & f_{\alpha_1}\colon \psi_{\alpha_1} \to \varrho_{\alpha_1},\, g_{\alpha_1}\colon \varrho_{\alpha_1} \to \psi_{\alpha_1}, \\ & \vdots \\ & f_{\alpha_n}\colon \psi_{\alpha_n} \to \varrho_{\alpha_n},\, g_{\alpha_n}\colon \varrho_{\alpha_n} \to \psi_{\alpha_n}, \\ & f_\alpha\colon \varrho_\alpha \to \psi_\alpha,\, g_\alpha\colon \psi_{\alpha_1} \to \varrho_\alpha \\ \vdash & M_\tau : \tau[\varrho_{\alpha_1}/\alpha_1,\ldots,\varrho_{\alpha_n}/\alpha_n,\varrho_\alpha/\alpha] \to \tau[\psi_{\alpha_1}/\alpha_1,\ldots,\psi_{\alpha_n}/\alpha_n,\psi_\alpha/\alpha] \end{array}$$

such that

$$[\![M_\tau]\!](f_{\alpha_1},g_{\alpha_1},\ldots,f_{\alpha_n},g_{\alpha_n},f_\alpha,g_\alpha) \;=\; F_\tau(f_{\alpha_1},g_{\alpha_1},\ldots,f_{\alpha_n},g_{\alpha_n},f_\alpha,g_\alpha)$$

is true for all morphisms $f_{\alpha_i}\colon [\![\psi_{\alpha_i}]\!] \to [\![\varrho_{\alpha_i}]\!]$, $g_{\alpha_i}\colon [\![\varrho_{\alpha_i}]\!] \to [\![\psi_{\alpha_i}]\!]$ $(i = 1,\ldots,n)$, $f\colon [\![\psi]\!] \to [\![\varrho]\!]$, and $g\colon [\![\varrho]\!] \to [\![\psi]\!]$. (A detailed construction of this term does not give any deeper conceptual insights, but generates much more notational hassle; that is why the simpler parameter free notion is presented here.) The only new case is $\tau = \mu\beta.\tau'$. Let us assume w.l.o.g. that τ' does not contain any free type variables except α and β. We have to find a term $M_{\mu\beta.\tau'}$ defining $F_{\mu\beta.\tau'}$ which is characterised by the equation

$$F_{\mu\beta.\tau'}(f,g) \;=\; \mathsf{fold}_{[\![\psi]\!],[\![\psi]\!]} \circ F_{\tau'}\big(f,g,F_{\mu\beta.\tau'}(g,f),F_{\mu\beta.\tau'}(f,g)\big) \circ \mathsf{unfold}_{[\![\varrho]\!],[\![\varrho]\!]}$$

(see Proposition 2.3 and Proposition 3.11, respectively) where $F_{\tau'}$ is definable by induction hypothesis. However, this is a bit sneaky as $F_{\mu\beta.\tau'}$ is applied to different arguments. Therefore, we solve the system of simultaneous equations which is possible due to *Bekič's rule* [Bek69]:

$$\begin{array}{rcl} F^-_{\mu\beta.\tau'}(g,f) &=& \mathsf{fold}_{[\![\varrho]\!],[\![\varrho]\!]} \circ F^-_{\tau'}\big(g,f,F^+_{\mu\beta.\tau'}(f,g),F^-_{\mu\beta.\tau'}(g,f)\big) \circ \mathsf{unfold}_{[\![\psi]\!],[\![\psi]\!]} \\ F^+_{\mu\beta.\tau'}(f,g) &=& \mathsf{fold}_{[\![\psi]\!],[\![\psi]\!]} \circ F^+_{\tau'}\big(f,g,F^-_{\mu\beta.\tau'}(g,f),F^+_{\mu\beta.\tau'}(f,g)\big) \circ \mathsf{unfold}_{[\![\varrho]\!],[\![\varrho]\!]} \end{array}$$

By induction hypothesis, there are terms-in-contexts

$$\begin{array}{l} f:\psi\to\varrho,\, g:\varrho\to\psi,\, k:\tau'[\psi/\alpha]\to\tau'[\varrho/\alpha],\, h:\tau'[\varrho/\alpha]\to\tau'[\psi/\alpha] \\ \qquad\qquad\qquad \vdash\; M^+_{\tau'} : \tau'\big[\varrho/\alpha,\mu\beta.\tau'[\varrho/\alpha]/\beta\big] \to \tau'\big[\psi/\alpha,\mu\beta.\tau'[\psi/\alpha]/\beta\big], \end{array}$$

$$\begin{array}{l} g:\varrho\to\psi,\, f:\psi\to\varrho,\, h:\tau'[\varrho/\alpha]\to\tau'[\psi/\alpha],\, k:\tau'[\psi/\alpha]\to\tau'[\varrho/\alpha] \\ \qquad\qquad\qquad \vdash\; M^-_{\tau'} : \tau'\big[\psi/\alpha,\mu\beta.\tau'[\psi/\alpha]/\beta\big] \to \tau'\big[\varrho/\alpha,\mu\beta.\tau'[\varrho/\alpha]/\beta\big], \end{array}$$

defining $F^-_{\tau'}$ and $F^+_{\tau'}$, respectively. Then the term

$$f:\psi\to\varrho,\, g:\varrho\to\psi \;\vdash\; M_{\mu\beta.\tau'} : \tau'[\varrho/\alpha] \to \tau'[\psi/\alpha]$$

can be defined by the second component of the solution of this system of equations:

$$
\begin{aligned}
M_{\mu\beta.\tau'} \quad = \quad & \mathbf{pr_2}\Big(\mathbf{Y}_\delta\Big(\lambda m{:}\delta.\Big\langle \lambda x{:}\tau'[\psi/\alpha].\,\mathbf{fold}\,\big((M^-_{\tau'}[\mathbf{pr_2}\,m/h,\mathbf{pr_1}\,m/k])(\mathbf{unfold}\;x)\big),\\
& \qquad\qquad \lambda x{:}\tau'[\varrho/\alpha].\,\mathbf{fold}\,\big((M^+_{\tau'}[\mathbf{pr_1}\,m/k,\mathbf{pr_2}\,m/h])(\mathbf{unfold}\;x)\big)\Big\rangle\Big)\Big)
\end{aligned}
$$

where $\delta := \big(\tau'[\psi/\alpha] \to \tau'[\varrho/\alpha]\big) \times \big(\tau'[\varrho/\alpha] \to \tau'[\psi/\alpha]\big)$.

In accordance with Proposition 2.1 and Proposition 3.11, respectively, the terms E_n and P_n (for $n \in \mathbb{N}$) can be inductively defined by:

$$
\begin{aligned}
E_0 &:= \lambda x{:}\mathbf{void}.\Omega_{\mu\alpha.\tau},\\
P_0 &:= \lambda x{:}\mu\alpha.\tau.\Omega_{\mathbf{void}},\\
E_{n+1} &:= \mathbf{fold}\big((\lambda f{:}\mu\alpha.\tau{\to}\tau^n_\alpha.\lambda g{:}\tau^n_\alpha{\to}\mu\alpha.\tau.\,M_\tau)\,P_n\,E_n\big),\\
P_{n+1} &:= \lambda x{:}\mu\alpha.\tau.\big((\lambda g{:}\tau^n_\alpha{\to}\mu\alpha.\lambda f{:}\mu\alpha.\tau{\to}\tau^n_\alpha.\,M_\tau)\,E_n\,P_n\big)\mathbf{unfold}(x)
\end{aligned}
$$

where τ^n_α is given by $\tau^0_\alpha := \mathbf{void}$ and $\tau^{n+1}_\alpha := \tau[\tau^n_\alpha/\alpha]$. □

Given an SFL type τ, Lemma 5.10 allows us to define closed SFL terms Q^τ_n of type $\tau \to \tau$ with $[\![Q^\tau_n]\!] = \pi^{[\![\tau]\!]}_n$ for all projections $\pi^{[\![\tau]\!]}_n\colon [\![\tau]\!] \to [\![\tau]\!]$, making $[\![\tau]\!]$ an SFP domain as claimed in Proposition 5.1:

$$
\begin{aligned}
Q^{\tau_1\times\tau_2}_n &:= \lambda x{:}\tau_1{\times}\tau_2.\,\langle Q^{\tau_1}_n(\mathbf{pr_1}\,x), Q^{\tau_2}_n(\mathbf{pr_2}\,x)\rangle\\
Q^{\tau_1\to\tau_2}_n &:= \lambda f{:}\tau_1{\to}\tau_2.\lambda x{:}\tau_1.\,Q^{\tau_2}_n\big(f(Q^{\tau_1}_n x)\big)\\
Q^{\mu\alpha.\tau'}_n &:= \lambda x{:}\mu\alpha.\tau'.\,E_n\big(Q^{(\tau')^n_\alpha}_n(P_n x)\big)
\end{aligned}
$$

where τ^n_α is defined as in the proof of Lemma 5.10. It is left to the reader to complete this definition.

Now it is possible to extend the definability result from finite domains to compact elements in arbitrary domains:

LEMMA 5.11. *Every compact element in the denotation of an SFL type is definable.*

Proof. Let $d \in [\![\tau]\!]$ be a compact element in the denotation of a closed SFL type τ. Since $[\![\tau]\!]$ is an SFL-domain, $d = \bigsqcup^{\uparrow}{}_{n\in\mathbb{N}}\,\pi^{[\![\tau]\!]}_n(d)$ holds for an increasing sequence $(\pi_n)_{n\in\mathbb{N}}$ of finite projections $\pi^{[\![\tau]\!]}_n\colon [\![\tau]\!] \to [\![\tau]\!]$. Hence, there is natural number $n \in \mathbb{N}$ with $d = \pi^{[\![\tau]\!]}_n(d)$ for all $d \in [\![\tau]\!]$. We prove the following property for the type τ:

($\star$) There exists a closed finite SFL type ψ_τ and an embedding $e\colon [\![\psi_\tau]\!] \to [\![\tau]\!]$ and a projection $p\colon [\![\tau]\!] \to [\![\psi_\tau]\!]$ with $\pi^{[\![\tau]\!]}_n = e \circ p$ such that e and p are definable by terms E and P.

Due to Corollary 5.9, $p(d)$ is definable by an SFL term M, thus, $d = \pi^{[\![\tau]\!]}_n(d) = e\big(p(d)\big)$ can be defined by the term EM.

Instead of proving property ($\star$) directly, we show the following more general claim:

Let τ be an SFL type that may contain free type variables and ρ be a type variable environment. If $\rho(\alpha)$ satisfies property $(\star)$ for all free type variables α in τ, then so does $\tau[\rho]$.

This implies that property $(\star)$ is true for all closed types, which finishes the proof.

The proof is a parametrised induction on the structure of τ. We just focus on some of the cases to illustrate the technique. Let ρ be a type variable environment such that property $(\star)$ is true for all types $\rho(\alpha)$ where α is a free type variable in τ.

$\tau = \tau_1 \times \tau_2$: By induction hypothesis, there are four SFL programs E_1, E_2, P_1, P_2 defining functions $e_1 \colon [\![\psi_{\tau_1[\rho]}]\!] \to [\![\tau_1[\rho]]\!]$, $e_2 \colon [\![\psi_{\tau_2[\rho]}]\!] \to [\![\tau_2[\rho]]\!]$, $p_1 \colon [\![\tau_1[\rho]]\!] \to [\![\psi_{\tau_1[\rho]}]\!]$ and $p_2 \colon [\![\tau_2[\rho]]\!] \to [\![\psi_{\tau_2[\rho]}]\!]$ with $\pi_n^{[\![\tau_1[\rho]]\!]} = e_1 \circ p_1$ and $\pi_n^{[\![\tau_2[\rho]]\!]} = e_2 \circ p_2$. Thus, choosing $\psi_{(\tau_1 \times \tau_2)[\rho]} := \psi_{\tau_1[\rho]} \times \psi_{\tau_2[\rho]}$, $E := \lambda x{:}\psi_{(\tau_1\times\tau_2)[\rho]}.\langle E_1(\mathbf{pr_1}\, x), E_2(\mathbf{pr_2}\, x)\rangle$ and $P := \lambda x{:}(\tau_1\times\tau_2)[\rho].\langle P_1(\mathbf{pr_1}\, x), P_2(\mathbf{pr_2}\, x)\rangle$ do the job for $(\tau_1 \times \tau_2)[\rho]$.

$\tau = \tau_1 \to \tau_2$: Take the same terms E_1, E_2, P_1, P_2 as in the case for product types, again, they exist by induction hypothesis. Moreover, define $\psi_{(\tau_1\to\tau_2)[\rho]} := \psi_{\tau_1[\rho]} \to \psi_{\tau_2[\rho]}$. If we set

$$E \;:=\; \lambda f{:}\psi_{(\tau_1\to\tau_2)[\rho]}.\, E_2(fP_1), \qquad\qquad P \;:=\; \lambda g{:}(\tau_1{\to}\tau_2)[\rho].\, P_2(gE_1),$$

then $(e \circ p)(g) = e_2 \circ p_2 \circ g \circ e_1 \circ p_1 = \pi_n^{[\![\tau_2[\rho]]\!]} \circ g \circ \pi_n^{[\![\tau_1[\rho]]\!]} = g$ follows by induction hypothesis.

$\tau = \mu\alpha.\tau'$: Because $\pi_n^{[\![\mu\alpha.\tau']\!]} \colon [\![\mu\alpha.\tau']\!] \to [\![\mu\alpha.\tau']\!]$ is a finite projection, there is an $m \in \mathbb{N}$ with $e_m \circ p_m \circ \pi_n^{[\![\mu\alpha.\tau']\!]} = \pi_n^{[\![\mu\alpha.\tau']\!]}$ (recall that (e_m, p_m) is the embedding projection pair between $[\![\mu\alpha.\tau']\!]$ and its m-th component). Define $\psi_{(\mu\alpha.\tau')[\rho]} := \psi_{\tau'[\rho_m]}$ where the environment ρ_m is the same as defined in the proof of Proposition 5.1. The functions e_m and p_m are defined by the terms E_m and P_m on page 86. $\square$

Now we have the tools to prove the main result of this section:

THEOREM 5.12. ***(Full abstraction)*** *Let M, N be closed terms of the same type. Then:*

$$[\![M]\!] \sqsubseteq [\![N]\!] \iff M \sqsubseteq_{SFL} N$$

Proof. Suppose $M \not\sqsubseteq_{SFL} N$. Then there is a context $C[\cdot]$ of type **unit** with $C[M]\Downarrow$, but $C[N]\not\Downarrow$. Due to Theorem 5.7, this implies $[\![C[M]]\!] = \top_{\mathbf{2}} \not\sqsubseteq \bot_{\mathbf{2}} = [\![C[N]]\!]$, contradicting $[\![M]\!] \sqsubseteq [\![N]\!]$.

As in previous proofs, for the "$\Leftarrow$"-direction we need to generalise the claim in order to deal with free type variables:

Let ρ be type variable environment and τ be an SFL type. If, for all free type variables α in τ and for all closed SFL terms M, N of type $\rho(\alpha)$, $M \sqsubseteq_{SFL} N$ implies $[\![M]\!] \sqsubseteq [\![N]\!]$, then $M \sqsubseteq_{SFL} N$ implies $[\![M]\!] \sqsubseteq [\![N]\!]$ for all closed SFL terms M, N of type $\tau[\rho]$.

Then $M \sqsubseteq_{SFL} N$ implies $[\![M]\!] \sqsubseteq [\![N]\!]$ for all closed terms M, N of a closed type τ.

The proof is a parametrised induction of the structure of the type τ of M and N. Most of the cases for finitary constructions are straightforward, we focus on liftings and function types as well as on recursive types. Let ρ be an environment such that $M \sqsubseteq_{SFL} N$ implies $[\![M]\!] \sqsubseteq [\![N]\!]$ for all closed SFL terms M, N of type $\rho(\alpha)$ for all type variables α in τ.

$\tau = \tau'_\perp$: Let us fix two closed terms M, N of type $\tau'[\rho]$ with $M \sqsubseteq_{\mathrm{SFL}} N$. Then $\mathbf{down}\, M \sqsubseteq_{\mathrm{SFL}} \mathbf{down}\, N$ and, therefore, $[\![\mathbf{down}\, M]\!] \sqsubseteq [\![\mathbf{down}\, N]\!]$ (by induction hypothesis) follow. This implies $[\![M]\!] \sqsubseteq \mathsf{up}([\![\mathbf{down}\, M]\!]) \sqsubseteq \mathsf{up}([\![\mathbf{down}\, N]\!])$. In case of $\mathsf{up}([\![\mathbf{down}\, N]\!]) = [\![N]\!]$, this clearly shows $[\![M]\!] \sqsubseteq [\![N]\!]$. Otherwise, $[\![M]\!] = \mathsf{up}(\perp_{[\![\tau'_\perp[\rho]]\!]})$ and $[\![N]\!] = \perp_{[\![\tau'_\perp[\rho]]\!]}$ would follow. However, as $M \sqsubseteq_{\mathrm{SFL}} N$ implies $\big(\widetilde{\lambda} x{:}\tau'_\perp[\rho].\, *\big)^{\sim} M \sqsubseteq_{\mathrm{SFL}} \big(\widetilde{\lambda} x{:}\tau'_\perp[\rho].\, *\big)^{\sim} N$, this is a contradiction to Theorem 5.7 because of $[\![\big(\widetilde{\lambda} x{:}\tau'_\perp[\rho].\, *\big)^{\sim} M]\!] = \top_{\mathbf{2}}$ and $[\![\big(\widetilde{\lambda} x{:}\tau'_\perp[\rho].\, *\big)^{\sim} N]\!] = \perp_{\mathbf{2}}$.

$\tau = \tau_1 \to \tau_2$: Let M, N be terms of type $(\tau_1 {\to} \tau_2)[\rho]$ with $M \sqsubseteq_{\mathrm{SFL}} N$ and $d \in [\![\tau_1[\rho]]\!]$ be an arbitrary element. Because of Corollary 5.2 there is a directed set of compact elements $d_i \in [\![\tau_1[\rho]]\!]$ $(i \in I)$ with ${\bigsqcup^{\uparrow}}_{i \in I} d_i = d$. Those elements d_i are definable by SFL terms D_i due to Lemma 5.11. Hence, $M D_i \sqsubseteq_{\mathrm{SFL}} N D_i$ follows, which implies $[\![M]\!](d_i) \sqsubseteq [\![N]\!](d_i)$ by induction hypothesis. Finally, this leads to $[\![M]\!](d) = [\![M]\!]({\bigsqcup^{\uparrow}}_{i \in I} d_i) \sqsubseteq [\![N]\!]({\bigsqcup^{\uparrow}}_{i \in I} d_i) = [\![N]\!](d)$.

$\tau = \mu\alpha.\tau'$: Again, we use the type variable environment ρ_n defined in the proof of Proposition 5.1. By induction hypothesis, $M \sqsubseteq_{SFL} N$ implies $[\![M]\!] \sqsubseteq [\![N]\!]$ for all closed SFL terms M, N of type $\tau'[\rho_n]$ for all $n \in \mathbb{N}$. Let M, N be two fixed terms of type $(\mu\alpha.\tau')[\rho]$ with $[\![M]\!] \not\sqsubseteq [\![N]\!]$. Then there exists an $n \in \mathbb{N}$ with $p_n([\![M]\!]) \not\sqsubseteq p_n([\![N]\!])$. Because of Lemma 5.10, there is a term P_n with $[\![P_n M]\!] = p_n([\![M]\!]) \not\sqsubseteq p_n([\![N]\!]) = [\![P_n N]\!]$. Since $P_n M$ and $P_n N$ are closed terms of type $\tau'[\rho_n]$, $P_n M \not\sqsubseteq_{SFL} P_n N$ follows. This entails $M \not\sqsubseteq_{SFL} N$. □

As a consequence, the equations-in-contexts are correct in the following sense:

COROLLARY 5.13. ***(Soundness)*** *Let $\Gamma \vdash M = N : \tau$ be an SFL equation-in-context as defined on page 64. Then $[\![\Gamma \vdash M : \tau]\!] = [\![\Gamma \vdash N : \tau]\!]$ holds.*

Proof. The proof is an easy induction on the derivation of $\Gamma \vdash M = N : \tau$. Of particular interest are those cases that have an additional assumption. Suppose that σ is a computational SFL type and assume that $\Gamma \vdash M' : \sigma$ is a term-in-context such that $M'[N_1/x_1, \ldots, N_n/x_n]$ terminates for all well-typed closed terms $N_1, \ldots, N_n$. Then the term $(\widetilde{\lambda} x{:}\sigma.*)^{\sim} M'[N_1/x_1, \ldots, N_n/x_n]$ terminates as well and Theorem 5.7 implies $[\![(\widetilde{\lambda} x{:}\sigma.*)^{\sim} M'[N_1/x_1, \ldots, N_n/x_n]]\!] = \chi_{[\![\sigma]\!]}([\![M'[N_1/x_1, \ldots, N_n/x_n]]\!]) \neq \perp_{\mathbf{2}}$. Hence, $[\![M'[N_1/x_1, \ldots, N_n/x_n]]\!] \neq \perp_{[\![\sigma]\!]}$ follows and the Substitution Lemma 5.5 yields $[\![\Gamma \vdash M' : \sigma]\!]([\![N_1]\!], \ldots, [\![N_n]\!]) \neq \perp_{[\![\sigma]\!]}$. Because $[\![\Gamma]\!]$ is an algebraic domain (Corollary 5.2) and because every compact element of $[\![\Gamma]\!]$ is definable in SFL (Lemma 5.11), this entails $[\![\Gamma \vdash M' : \sigma]\!](d) \neq \perp_{[\![\sigma]\!]}$ for all elements $d \in [\![\Gamma]\!]$. The actual proof for the soundness of the equation rules is now straightforward. □

6 A Call-by-Value Based Language Equivalent to SFL

In the previous sections, we discussed the category SD and its nice properties and examined its syntactic counterpart—the language SFL—which has a lot of interesting features from an operational point of view. In order to describe the structure of a cartesian closed category, we were more or less forced to treat SFL as a call-by-name language, although we added a couple of call-by-value features.

Levy's Call-by-Push Calculus

A different approach leading to a very similar result was made by P. Levy [Lev99]. He developed a so-called *call-by-push* language whose type system is basically the same as that of SFL. Terms are divided into *values* and *computations*. Unlike in SFL, values stand for results of concrete calculations, in particular, $\lambda x.M$ is not considered as a value. Consequently, Levy considers e. g. function terms as elements of a computation type and numbers as objects of a value type.

The idea of having two different kinds of types is essentially the same as in SFL. However, whereas call-by-push types are either computation types or value types, one sort of SFL types (the SSB-types or computational types) can be considered as a subclass of the other one (the class of all SFL types). It turns out that Levy-style value types correspond to computational types in SFL and call-by-push computation types can be seen as general types in SFL. Unfortunately, this creates a slightly confusing terminology since SFL computational types (SSB-types) do not correspond to computation types, but to value types. Nevertheless, in both cases the terminology is sound in its own right. In particular, unlike in Levy's call-by-push calculus where terms of value types are actually values (which justifies the terminology *value type*), there are SFL terms of computational type that are not values. In the reformulation of Levy's call-by-push language, we give up the idea of having only values as terms of computational types. However, we distinguish between normal forms of computational types (i. e. SSB-types) and arbitrary types and use the expression *terminal computation* instead of *value* if the type is not an SSB-type.

Levy uses two type formers U and F where U takes computations to values and, vice versa, F takes values to computations. Depending on the interpretation of these type formers, they can be used to describe computational effects like, for instance, global storage, finite output with errors, nondeterminism and control. In this sense, the SFL lifting operator $\cdot_\perp$, taking an arbitrary type to an SSB-type, is a special instance of Levy's type former F. The according type former U is just an embedding from SSB-types to arbitrary SFL types that is not explicitly specified within the syntax of SFL.

A more technical difference between the two approaches is the lack of separated sum types and call-by-name function types in Levy's call-by-push calculus. As described in Section 4, they cannot be defined satisfying the usual evaluation rules (only up to observational equivalence). Nevertheless, as they are not as natural from a point of view of programming as from a denotationally oriented approach, we will waive them as basic constructors in the following as well.

More importantly, Levy's call-by-push calculus does not support recursive types. The main subject of this section is the language SFPL (sequential functional programming language), which is a reformulation of Levy's call-by-push calculus extended with recursive types.

6.1 The Language SFPL

Whereas SFL is a generalisation of the call-by-name language PCF in the sense that it has the same operational semantics and the same notion of observational equivalence, SFPL can be understood as an extension of the call-by-value language FPC. We study how SFPL can be interpreted in the category $SD_\perp$ and show that $SD_\perp$ gives rise to a fully abstract model for SFPL.

SFPL Types

As explained above the type system for SFPL is the same as for SFL (except that we omit call-by-name function types and separated sum types):

$$\begin{array}{rcl}\sigma & ::= & \alpha \mid \tau_\perp \mid \sigma \otimes \sigma \mid \sigma \oplus \sigma \mid \mu\alpha.\sigma \\ \tau & ::= & \sigma \mid \tau \times \tau \mid \sigma \multimapdotinv \tau \mid \mu\alpha.\tau\end{array}$$

where α ranges over the set of all type variables. We still use σ, σ_1, σ_2 etc. for SSB-types and τ, τ_1, τ_2 for arbitrary types. As done before we define further types $\mathbf{void} := \mu\alpha.\alpha$ and $\mathbf{unit} := \mathbf{void}_\perp$.

SFPL Terms

Also the terms are basically the same as in SFL:

$$\begin{array}{rl} M ::= & x \mid \langle M, M\rangle \mid \mathbf{pr_1}\, M \mid \mathbf{pr_2}\, M \mid (M, M) \mid \mathbf{let}^{\sigma,\sigma,\tau}\, M \ \mathbf{be}\ (x, y)\ \mathbf{in}\ M \mid \\ & \widetilde{\mathbf{inl}}^{\sigma,\sigma} M \mid \widetilde{\mathbf{inr}}^{\sigma,\sigma} M \mid \widetilde{\mathbf{case}}^{\sigma,\sigma,\tau} M \ \mathbf{of}\ \widetilde{\mathbf{inl}}\, x \Rightarrow M; \widetilde{\mathbf{inr}}\, x \Rightarrow M \mid \mathbf{up}\, M \mid \\ & \mathbf{down}\, M \mid \widetilde{\lambda} x{:}\sigma.M \mid M\tilde{\ }M \mid \mathbf{fold}^{\mu\alpha.\tau} M \mid \mathbf{unfold}^{\mu\alpha.\tau} M \end{array}$$

As in SFL, we usually omit type notations. Since SFPL is directed more towards call-by-value style programming languages, we use a '**let**'-expression as eliminator for smash product types. Nevertheless, call-by-value projections can be easily defined by

$$\begin{array}{rcl}\widetilde{\mathbf{pr_1}}\, M & := & \mathbf{let}\ M\ \mathbf{be}\ (x, y)\ \mathbf{in}\ x, \\ \widetilde{\mathbf{pr_2}}\, M & := & \mathbf{let}\ M\ \mathbf{be}\ (x, y)\ \mathbf{in}\ y.\end{array}$$

This does not change the expressiveness of the language as the **let**-term can be regained from the projections by setting

$$\mathbf{let}\ M\ \mathbf{be}\ (x, y)\ \mathbf{in}\ N \ := \ \big((\widetilde{\lambda} x{:}\sigma_1.\widetilde{\lambda} y{:}\sigma_2.N)\tilde{\ }(\widetilde{\mathbf{pr_1}}\, M)\big)\tilde{\ }(\widetilde{\mathbf{pr_2}}\, M).$$

Since we do not have separated sums and call-by-name functions, we could easily avoid the tildes for the corresponding terms. However, we will still use them in the following to avoid unnecessary confusion with the SFL syntax.

An important difference to SFL is that SFPL contexts must not contain non-computational types. The reason is that, for an arbitrary SFPL type σ, the interpretation

$$[\![x : \sigma \vdash * : \mathbf{unit}]\!] \colon [\![\sigma]\!] \to \mathbf{2}$$

gives rise to a bottom separating morphism from $[\![\sigma]\!]$ to $\mathbf{2}$ (the precise definition of the denotation is given later). Therefore, σ is interpreted as an SSB-object.

The SFPL typing rules are:

$$\frac{}{\Gamma, x{:}\sigma \vdash x : \sigma}$$

$$\frac{\Gamma \vdash M : \tau_1 \qquad \Gamma \vdash N : \tau_2}{\Gamma \vdash \langle M, N\rangle : \tau_1 \times \tau_2} \qquad \frac{\Gamma \vdash M : \tau_1 \times \tau_2}{\Gamma \vdash \mathbf{pr_1}\, M : \tau_1} \qquad \frac{\Gamma \vdash M : \tau_1 \times \tau_2}{\Gamma \vdash \mathbf{pr_2}\, M : \tau_2}$$

$$\frac{\Gamma \vdash M : \sigma_1 \qquad \Gamma \vdash N : \sigma_2}{\Gamma \vdash (M, N) : \sigma_1 \otimes \sigma_2} \qquad \frac{\Gamma \vdash M : \sigma_1 \otimes \sigma_2 \qquad \Gamma, x_1{:}\sigma_1, x_2{:}\sigma_2 \vdash N : \tau}{\Gamma \vdash \mathbf{let}\ M\ \mathbf{be}\ (x, y)\ \mathbf{in}\ N : \tau}$$

$$\frac{\Gamma \vdash M : \sigma_1}{\Gamma \vdash \widetilde{\mathbf{inl}}^{\sigma_1,\sigma_2} M : \sigma_1 \oplus \sigma_2} \qquad \frac{\Gamma \vdash M : \sigma_2}{\Gamma \vdash \widetilde{\mathbf{inr}}^{\sigma_1,\sigma_2} M : \sigma_1 \oplus \sigma_2}$$

$$\frac{\Gamma \vdash M : \sigma_1 \oplus \sigma_2 \qquad \Gamma, x_1{:}\sigma_1 \vdash M_1 : \tau \qquad \Gamma, x_2{:}\sigma_2 \vdash M_2 : \tau}{\Gamma \vdash \widetilde{\mathbf{case}}^{\sigma_1,\sigma_2,\tau} M\ \mathbf{of}\ \widetilde{\mathbf{inl}}\ x_1 \Rightarrow M_1; \widetilde{\mathbf{inr}}\ x_2 \Rightarrow M_2 : \tau}$$

$$\frac{\Gamma, x{:}\sigma \vdash M : \tau}{\Gamma \vdash \widetilde{\lambda} x{:}\sigma.M : \sigma \multimap \tau} \qquad \frac{\Gamma \vdash M : \sigma \multimap \tau \qquad \Gamma \vdash N : \sigma}{\Gamma \vdash M\,\widetilde{}\,N : \tau}$$

$$\frac{\Gamma \vdash M : \tau}{\Gamma \vdash \mathbf{up}\, M : \tau_\perp} \qquad \frac{\Gamma \vdash M : \tau_\perp}{\Gamma \vdash \mathbf{down}\, M : \tau}$$

$$\frac{\Gamma \vdash M : \tau[\mu\alpha.\tau/\alpha]}{\Gamma \vdash \mathbf{fold}^{\mu\alpha.\tau} M : \mu\alpha.\tau} \qquad \frac{\Gamma \vdash M : \mu\alpha.\tau}{\Gamma \vdash \mathbf{unfold}^{\mu\alpha.\tau} M : \tau[\mu\alpha.\tau/\alpha]}$$

Again, all types are supposed to be closed.

6.2 Operational Semantics

Unlike in the language SFL, where a term such as $\widetilde{\lambda} x{:}\sigma.*$ is considered as a value, we now distinguish between *values* and *terminal computations*. Values in SFPL are canonical forms that can be understood as a proper result of a computation, in particular, their type is always computational. Terminal computations, however, are terms that cannot be evaluated any further, but in general do not stand for a result, such as functions or call-by-name pairs.

Values:

$$V \quad ::= \quad (V, V) \mid \widetilde{\mathbf{inl}}\, V \mid \widetilde{\mathbf{inr}}\, V \mid \mathbf{up}\, M \mid \mathbf{fold}\, V$$

Terminal computations:

$$T \quad ::= \quad V \mid \langle M, M\rangle \mid \widetilde{\lambda} x{:}\sigma.M \mid \mathbf{fold}\, T$$

Note that SFPL values are exactly the terminal computations of SSB-types.

For the evaluation rules, we again assume that all terms are closed:

$$\frac{}{\langle M_1, M_2\rangle \Downarrow \langle M_1, M_2\rangle} \qquad \frac{M \Downarrow \langle M_1, M_2\rangle \qquad M_1 \Downarrow T}{\mathbf{pr_1}\, M \Downarrow T} \qquad \frac{M \Downarrow \langle M_1, M_2\rangle \qquad M_2 \Downarrow T}{\mathbf{pr_2}\, M \Downarrow T}$$

$$\frac{M_1 \Downarrow V_1 \qquad M_2 \Downarrow V_2}{(M_1, M_2) \Downarrow (V_1, V_2)} \qquad \frac{M \Downarrow (V_1, V_2) \qquad N[V_1/x_1, V_2/x_2] \Downarrow T}{\mathbf{let}\ M\ \mathbf{be}\ (x_1, x_2)\ \mathbf{in}\ N \Downarrow T}$$

$$\frac{M \Downarrow V}{\widetilde{\mathbf{inl}}\, M \Downarrow \widetilde{\mathbf{inl}}\, V} \qquad \frac{M \Downarrow \widetilde{\mathbf{inl}}\, V \qquad M_1[V/x_1] \Downarrow T}{\widetilde{\mathbf{case}}\ M\ \mathbf{of}\ \widetilde{\mathbf{inl}}\ x_1 \Rightarrow M_1\,;\widetilde{\mathbf{inr}}\, x_2 \Rightarrow M_2 \Downarrow T}$$

$$\frac{M \Downarrow V}{\widetilde{\mathbf{inr}}\, M \Downarrow \widetilde{\mathbf{inr}}\, V} \qquad \frac{M \Downarrow \widetilde{\mathbf{inr}}\, V \qquad M_2[V/x_2] \Downarrow T}{\widetilde{\mathbf{case}}\ M\ \mathbf{of}\ \widetilde{\mathbf{inl}}\ x_1 \Rightarrow M_1\,;\widetilde{\mathbf{inr}}\, x_2 \Rightarrow M_2 \Downarrow T}$$

$$\frac{}{\widetilde{\lambda} x{:}\tau.M \Downarrow \widetilde{\lambda} x{:}\tau.M} \qquad \frac{M \Downarrow \widetilde{\lambda} x{:}\tau.M' \qquad N \Downarrow V \qquad M'[V/x] \Downarrow T}{M\,\widetilde{}\,N \Downarrow T}$$

$$\frac{}{\mathbf{up}\, M \Downarrow \mathbf{up}\, M} \qquad \frac{M \Downarrow \mathbf{up}\, N \qquad N \Downarrow T}{\mathbf{down}\, M \Downarrow T}$$

$$\frac{M \Downarrow T}{\mathbf{fold}\, M \Downarrow \mathbf{fold}\, T} \qquad \frac{M \Downarrow \mathbf{fold}\, T}{\mathbf{unfold}\, M \Downarrow T}$$

It is easy to verify that the normal evaluation rules for the smash product projections $\widetilde{\mathbf{pr_1}}$ and $\widetilde{\mathbf{pr_2}}$ as defined on page 90 and those of the **let**-constructor can be derived from each other.

It has already been mentioned that call-by-name function types and separated sum types as well as the types **void** and **unit** can be defined in SFPL in the same way as in SFL. The translation of call-by-name function types and the corresponding terms into strict function types (up to observational equivalence) is given in Section 4, it is left to the reader to work out the details for separated sum types. From now on, we use expressions such as $\lambda x{:}\tau.M$, $\mathbf{case}\, M\, \mathbf{of}\ \mathbf{in}_1\, x_1 \Rightarrow M_1; \ldots; \mathbf{in}_n\, x_n \Rightarrow M_n : \tau$ etc. as syntactic sugar in SFPL.

Furthermore, it is also possible to define recursive functions. As it is more common in real programming languages we directly construct a term $\mathbf{fix}\, x.M$ to compute a fixed point of the term-in-context $\Gamma, x{:}\tau_\perp \vdash M : \tau$ rather than using a fixed point combinator as in SFL. The definition is:

$$\mathbf{fix}\, x.M \ := \ \Big(\mathbf{unfold}\Big(\mathbf{down}\,\big(\mathbf{up}\,(\mathbf{fold}\ Z)\big)\Big)\Big)\widetilde{}\,\big(\mathbf{up}\,(\mathbf{fold}\ Z)\big)$$

where

$$Z \;:=\; \widetilde{\lambda} y{:}\big(\mu\alpha.(\alpha_\perp \multimap \tau)\big)_\perp.\,(\widetilde{\lambda} x{:}\tau_\perp.\,M)^{\sim}\Big(\mathbf{up}\,\Big(\big(\mathbf{unfold}\,(\mathbf{down}\;y)\big)^{\sim} y\Big)\Big).$$

It is easy to verify that the following evaluation rule is derivable:

$$\frac{M[\mathbf{up}\,(\mathbf{fix}\,x.M)/x] \Downarrow V}{\mathbf{fix}\,z.M \Downarrow V}$$

As $\mathsf{unfold} \circ \mathsf{down} \circ \mathsf{up} \circ \mathsf{fold}$ is an isomorphism in SD, $\mathbf{fix}\,x.M$ has the same interpretation as the much simpler term $\mathbf{fix}'\,x.M = Z^{\sim}\big(\mathbf{up}\,(\mathbf{fold}\;Z)\big)$. Accordingly, it can be used as a fixed point operator as well, however, it is not possible to derive the analogue evaluation rule for $\mathbf{fix}'\,x.M$. As for SFL, one has to decide whether a clearly described evaluation strategy is worth the effort of encoding this rather complicated isomorphism. Either way, as we will see later, $\mathbf{fix}\,x.M$ indeed computes the least fixed point of the term-in-context $x:\tau_\perp \vdash M:\tau$ in the sense that

$$[\![\mathbf{fix}\,x.M]\!] \;=\; \bigsqcap\{x \in [\![\tau]\!] \mid [\![M]\!]\big(\mathsf{up}(x)\big) = x\}$$

holds. In particular, we can define the always diverging term

$$\Omega_\tau \;:=\; \mathbf{fix}\,x.\,\mathbf{down}\,x,$$

for every SFPL type τ, and

$$* \;:=\; \mathbf{up}\,\Omega_{\mathbf{void}}$$

as (up to equivalence) only value of type **unit**.

Observational equivalence in SFPL is defined exactly in the same way as in SFL. Also here, the type **unit** could be replaced by any other computational type in the definition of observational (in-)equivalence.

DEFINITION 6.1. *Let M, N be closed SFPL terms. We write $M \sqsubseteq_{SFPL} N$ if $C[M]\Downarrow$ implies $C[N]\Downarrow$ for all contexts $C[\cdot]$ of type* **unit**.

Also the equations-in-contexts in SFPL are basically the same as in SFL. The new equation for the **let**-constructor is:

- if $M_1[N_1/x_1,\dots,N_n/x_n]\Downarrow$ and $M_2[N_1/x_1,\dots,N_n/x_n]\Downarrow$ for all well-typed closed terms $N_1,\dots,N_n$, then $\Gamma \vdash \mathbf{let}\ (M_1,M_2)\ \mathbf{be}\ (y_1,y_2)\ \mathbf{in}\ M = M[M_1/y_1, M_2/y_2] : \tau$

where Γ is a context with variables $x_1,\dots,x_n$. In Levy's call-by-push calculus, all terms of value types terminate by definition, hence, Levy does not need to require the side condition that all terms $M_i[\vec{N}/\vec{x}]$ terminate.

The type system and the evaluation rules for SFPL as well as the equations-in-contexts look very similar to the ones of SFL. Nevertheless, the underlying concepts of SFPL are different. This becomes clear by looking at denotations of terms-in-contexts of the form $\Gamma \vdash M : \tau$. In SD, such SFL-terms are interpreted as functions from $[\![\Gamma]\!]$ to $[\![\tau]\!]$ where

$[\![\Gamma]\!]$ is the product of denotations of all types in Γ. Since SFPL is based on the call-by-value paradigm, SFPL terms-in-contexts are interpreted as functions that are strict in all arguments. Hence, for $[\![\Gamma]\!]$ we need to consider the smash product of interpretations of types in Γ (which is bottom if one of its components is bottom) rather than the cartesian product.

Levy uses the Scott model restricted to strict functions as a model for his language. However, as for the language SFL, the category *SD* provides a suitable framework for SFPL as well. As SFPL is a call-by-value language, we focus on the subcategory $SD_\perp$ with only strict functions as morphisms. This category is symmetric monoidal closed and, therefore, admits an interpretation of strict function types in correspondence with smash product types (although we do not use the whole symmetric monoidal structure as there are no types of the form $\tau \multimap \tau$ for arbitrary τ).

6.3 Interpretation of SFPL in $SD_\perp$

SFPL types are interpreted in $SD_\perp$ in the same way as SFL types in *SD*. In particular, there is no problem in interpreting recursive types as all bilimits are computed in $SD_\perp$ rather than in *SD* anyway. As already pointed out an SFPL term-in-context $x_1{:}\sigma_1, \ldots, x_n{:}\sigma_n \vdash M : \tau$ is interpreted as a function

$$[\![x_1{:}\sigma_1, \ldots, x_n{:}\sigma_n \vdash M : \tau]\!] \colon \bigotimes_{i=1}^{n} [\![\sigma_i]\!] \multimap [\![\tau]\!].$$

As expected, most SFPL terms are interpreted in the same way as they are interpreted as SFL terms (of course, the domains need to be changed from cartesian products to smash products and the $\vdash$-symbol is interpreted 'strictly'), the only new constructor is interpreted as follows:

$$\begin{aligned}&[\![\Gamma \vdash \mathbf{let}\ M\ \mathbf{be}\ (x_1, x_2)\ \mathbf{in}\ N]\!] \\ &\qquad\qquad = \widetilde{\mathsf{eval}} \circ \big(\widetilde{\mathsf{curry}}([\![\Gamma, x_1{:}\sigma_1, x_2{:}\sigma_2 \vdash N : \tau]\!] \circ \widetilde{\mathsf{ass}}), [\![\Gamma \vdash M : \sigma_1 \otimes \sigma_2]\!]\big)\end{aligned}$$

where $\widetilde{\mathsf{ass}} \colon [\![\Gamma]\!] \otimes ([\![\sigma_1]\!] \otimes [\![\sigma_2]\!]) \multimap ([\![\Gamma]\!] \otimes [\![\sigma_1]\!]) \otimes [\![\sigma_2]\!]$ is an isomorphism expressing associativity and $\widetilde{\mathsf{curry}}$ and $\widetilde{\mathsf{eval}}$ are strict functions as described in Section 2.

As closed SFPL terms are interpreted in the same way as SFL terms, both languages, SFL and SFPL, are equally expressive:

LEMMA 6.2. *Let τ be an SFPL type and $d \in [\![\tau]\!]$. Then the following are equivalent:*

1. *There is a closed SFL term M with $[\![M]\!] = d$.*
2. *There is a closed SFPL term M with $[\![M]\!] = d$.*

□

It is without loss of generality to require τ to be an SFPL type as the denotation of every SFL type is isomorphic to that of some SFPL type.

The semantics for SFPL in the category $SD_\perp$ interprets the term $\mathbf{fix}\, x.M$ in the usual way:

LEMMA 6.3. *Given a term-in-context* $x{:}\tau_\perp \vdash M : \tau$ *with* $f = [\![x{:}\tau_\perp \vdash M : \tau]\!]$, *then* $\mathbf{fix}\, x.M$ *computes the least fixed of* M, *i.e.*

$$[\![\mathbf{fix}\, x.M]\!] \;=\; \bigsqcup_{n\in\mathbb{N}}^{\uparrow} (f \circ \mathbf{up}\,)^n(\perp_{[\![\tau]\!]}).$$

Proof. As for the SFL fixed point combinator $\mathbf{Y}$ verifying that $\mathbf{fix}\, x.M$ computes indeed the least fixed point of f is rather tedious.

The isomorphism fold and unfold for the bilimit $\mathsf{fix}(F) = [\![(\mu\alpha.(\alpha_\perp \multimap \tau)_\perp)]\!]$ of the corresponding functor F are given by:

$$\mathsf{fold} \;=\; \bigsqcup_n^{\uparrow} e_{n+1} \circ F(e_n, p_n) \;=\; \bigsqcup_n^{\uparrow} e_{n+1} \circ \lambda\!\!\lambda h^{F(\mathsf{fix}(F),\mathsf{fix}(F))}.h \circ e_{n\perp}$$

and

$$\mathsf{unfold} \;=\; \bigsqcup_n^{\uparrow} F(p_n, e_n) \circ p_{n+1} \;=\; \bigsqcup_n^{\uparrow} \lambda\!\!\lambda d^{\mathsf{fix}(F)}.p_{n+1}(d) \circ p_{n\perp}.$$

This yields:

$$\begin{aligned}
\mathsf{fold}([\![Z]\!]) &= \bigsqcup_n^{\uparrow} e_{n+1}([\![Z]\!] \circ e_{n\perp}) \\
&= \bigsqcup_n^{\uparrow} e_{n+1}\Big(\Big(\lambda\!\!\lambda x^{(F(\mathsf{fix}(F),\mathsf{fix}(F))_\perp}.f \circ \mathsf{up}\big((\mathsf{unfold} \circ \mathsf{down}(y))(y)\big)\Big) \circ e_{n\perp}\Big) \\
&= \bigsqcup_n^{\uparrow} e_{n+1}\Big(\lambda\!\!\lambda d^{F^{(n)}(\mathbf{1},\mathbf{1})_\perp}.f \circ \mathsf{up}\big((\mathsf{unfold} \circ \mathsf{down} \circ e_{n\perp}(d))(e_{n\perp}(d))\big)\Big) \\
&= \bigsqcup_n^{\uparrow} e_{n+1}\Big(\lambda\!\!\lambda d^{F^{(n)}(\mathbf{1},\mathbf{1})_\perp}.f \circ \mathsf{up}\Big(\bigsqcup_m^{\uparrow} (p_{m+1}(e_n \circ \mathsf{down}(d)) \circ p_{m\perp})(e_{n\perp}(d))\Big)\Big) \\
&= \bigsqcup_n^{\uparrow} e_{n+1}\Big(\lambda\!\!\lambda d^{F^{(n)}(\mathbf{1},\mathbf{1})_\perp}.f \circ \mathsf{up}\big((e_{n+1,n} \circ \mathsf{down}(d) \circ p_{n\perp})(e_{n\perp}(d))\big)\Big) \\
&= \bigsqcup_n^{\uparrow} e_{n+1}\Big(\lambda\!\!\lambda d^{F^{(n)}(\mathbf{1},\mathbf{1})_\perp}.f \circ \mathsf{up}\big((e_{n+1,n} \circ \mathsf{down}(d))(d)\big)\Big)
\end{aligned}$$

Note that all applications are meant to be strict. Plugging this into the denotation of $\mathbf{fix}\, x.M$ leads to:

$$\begin{aligned}
&[\![\mathbf{fix}\, x.M]\!] \\
&\quad= [\![Z]\!](\mathsf{up} \circ \mathsf{fold}([\![Z]\!])) \\
&\quad= f \circ \mathsf{up}\Big(\big(\mathsf{unfold} \circ \mathsf{down}(\mathsf{up} \circ \mathsf{fold}([\![Z]\!]))\big)(\mathsf{up} \circ \mathsf{fold}([\![Z]\!]))\Big) \\
&\quad= f \circ \mathsf{up}\Big(\Big(\mathsf{unfold}\Big(\bigsqcup_n^{\uparrow} e_{n+1}\big(\lambda\!\!\lambda d^{F^{(n)}(\mathbf{1},\mathbf{1})_\perp}.f \circ \mathsf{up}((e_{n+1,n} \circ \mathsf{down}(d))(d))\big)\Big)\Big)(\cdots)\Big) \\
&\quad= f \circ \mathsf{up}\Big(\Big(\bigsqcup_n^{\uparrow} p_{n+1}\Big(e_{n+1}\big(\lambda\!\!\lambda d^{F^{(n)}(\mathbf{1},\mathbf{1})_\perp}.f \circ \mathsf{up}((e_{n+1,n} \circ \mathsf{down}(d))(d))\big)\Big) \circ p_{n\perp}\Big) \cdots\Big) \\
&\quad= \bigsqcup_n^{\uparrow} f \circ \mathsf{up}\big((\lambda\!\!\lambda y^{\mathsf{fix}(F)_\perp}.f \circ \mathsf{up}(\zeta_n))(\mathsf{up} \circ \mathsf{fold}([\![Z]\!]))\big)
\end{aligned}$$

where $\zeta_n = \big(e_{n+1,1} \circ \mathsf{down}\big(p_{n\perp}(y)\big)\big)\big(p_{n\perp}(y)\big)$. Because of

$$\begin{array}{rcl} e_{n+1,n} \circ p_{n,n+1} & = & F(p_{n-1,n}, e_{n,n-1}) \circ p_{n,n+1} \\ & = & (\lambda h^{F^{(n)}(\mathbf{1},\mathbf{1})}.h \circ p_{n-1,n\perp}) \circ p_{n,n+1} \\ & = & \lambda d^{F^{(n+1)}(\mathbf{1},\mathbf{1})}.\big(p_{n,n+1}(d)\big) \circ p_{n-1,n\perp} \end{array}$$

we can compute ζ_n as follows:

$$\begin{array}{rcl} \zeta_n & = & \Big(e_{n+1,1} \circ \mathsf{down}\big(p_{n\perp}(y)\big)\Big)\big(p_{n\perp}(y)\big) \\ & = & \Big(e_{n+1,1} \circ p_n\big(\mathsf{down}(y)\big)\Big)\big(p_{n\perp}(y)\big) \\ & = & \Big(e_{n+1,n} \circ p_{n,n+1} \circ p_{n+1}\big(\mathsf{down}(y)\big)\Big)\big(p_{n\perp}(y)\big) \\ & = & \bigg(\Big(\lambda d^{F^{(n+1)}(\mathbf{1},\mathbf{1})}.\big(p_{n,n+1}(d)\big) \circ p_{n-1,n\perp}\Big)\Big(p_{n+1}\big(\mathsf{down}(y)\big)\Big)\bigg)\big(p_{n\perp}(y)\big) \\ & = & p_n\big(\mathsf{down}(y)\big) \circ p_{n-1,n\perp}\big(p_{n\perp}(y)\big), \end{array}$$

for $n > 0$. Furthermore,

$$\begin{array}{rl} & \Big(p_n\big(\mathsf{fold}(\llbracket Z \rrbracket)\big)\Big)\big(p_{n-1\perp}(y)\big) \\ = & \bigg(\lambda d^{F^{(n-1)}(\mathbf{1},\mathbf{1})_\perp}.f \circ \mathsf{up}\Big(\big(e_{n,n-1} \circ \mathsf{down}(d)\big)(d)\Big)\bigg)\big(p_{n-1\perp}(y)\big) \\ = & f \circ \mathsf{up}\bigg(\Big(e_{n,n-1} \circ \mathsf{down}\big(p_{n-1\perp}(y)\big)\Big)\big(p_{n-1\perp}(y)\big)\bigg) \\ = & f \circ \mathsf{up}(\zeta_{n-1}) \end{array}$$

holds, which implies

$$\begin{array}{rl} & \big(\lambda y^{\mathsf{fix}(F)_\perp}.f \circ \mathsf{up}(\zeta_n)\big)\big(\mathsf{up} \circ \mathsf{fold}(\llbracket Z \rrbracket)\big) \\ = & f \circ \mathsf{up}\bigg(p_n\big(\mathsf{fold}(\llbracket Z \rrbracket)\big) \circ p_{n-1,n\perp}\Big(p_{n\perp}\big(\mathsf{up} \circ \mathsf{fold}(\llbracket Z \rrbracket)\big)\Big)\bigg) \\ = & f \circ \mathsf{up}\bigg(\Big(\lambda d^{F^{(n)}(\mathbf{1},\mathbf{1})_\perp}.p_n \circ \mathsf{fold}(\llbracket Z \rrbracket)\big(p_{n-1,n\perp}(d)\big)\Big)\Big(p_{n\perp}\big(\mathsf{up} \circ \mathsf{fold}(\llbracket Z \rrbracket)\big)\Big)\bigg) \\ = & f \circ \mathsf{up}\bigg(\Big(\lambda y^{\mathsf{fix}(F)_\perp}.p_n \circ \mathsf{fold}(\llbracket Z \rrbracket)\big(p_{n-1\perp}(y)\big)\Big)\big(\mathsf{up} \circ \mathsf{fold}(\llbracket Z \rrbracket)\big)\bigg) \\ = & f \circ \mathsf{up}\bigg(\Big(\lambda y^{\mathsf{fix}(F)_\perp}.f \circ \mathsf{up}(\zeta_{n-1})\Big)\big(\mathsf{up} \circ \mathsf{fold}(\llbracket Z \rrbracket)\big)\bigg). \end{array}$$

This entails

$$\begin{array}{rcl} \llbracket \mathbf{fix}\, x.M \rrbracket & = & \bigsqcup_n^{\uparrow} f \circ \mathsf{up}\Big(\big(\lambda y^{\mathsf{fix}(F)_\perp}.f \circ \mathsf{up}(\zeta_n)\big)\big(\mathsf{up} \circ \mathsf{fold}(\llbracket Z \rrbracket)\big)\Big) \\ & = & \bigsqcup_n^{\uparrow} (f \circ \mathsf{up})^n\Big(\big(\lambda y^{\mathsf{fix}(F)_\perp}.f \circ \mathsf{up}(\zeta_1)\big)\big(\mathsf{up} \circ \mathsf{fold}(\llbracket Z \rrbracket)\big)\Big). \end{array}$$

Using

$$
\begin{aligned}
&\big(\lambda y^{\mathsf{fix}(F)_\perp}.f \circ \mathsf{up}(\zeta_1)\big)\big(\mathsf{up} \circ \mathsf{fold}([\![Z]\!])\big)\\
&= \; f \circ \mathsf{up}\Big(p_1\big(\mathsf{down} \circ \mathsf{up} \circ \mathsf{fold}([\![Z]\!])\big) \circ p_{1,0\perp}\Big(p_{1\perp}\big(\mathsf{up} \circ \mathsf{fold}([\![Z]\!])\big)\Big)\Big)\\
&= \; f \circ \mathsf{up}\Big(\Big(\lambda d{:}\mathbf{2}.f \circ \mathsf{up}\Big(\big(e_{1,0} \circ \mathsf{down}(d)\big)(d)\Big)\Big) \circ p_{1,0\perp}\Big(p_{1\perp}\big(\mathsf{up} \circ \mathsf{fold}([\![Z]\!])\big)\Big)\Big)\\
&= \; f \circ \mathsf{up}\Big(\Big(\lambda d{:}\mathbf{2}.f \circ \mathsf{up}\big(\perp_{F(1,1)}(d)\big)\Big) \circ p_{1,0\perp}\Big(p_{1\perp}\big(\mathsf{up} \circ \mathsf{fold}([\![Z]\!])\big)\Big)\Big)\\
&= \; f \circ \mathsf{up}\Big(\Big(\lambda d{:}\mathbf{2}.f \circ \mathsf{up}(\perp_{[\![\tau]\!]})\Big) \circ p_{1,0\perp}\Big(p_{1\perp}\big(\mathsf{up} \circ \mathsf{fold}([\![Z]\!])\big)\Big)\Big)\\
&= \; f \circ \mathsf{up}\big(f \circ \mathsf{up}(\perp_{[\![\tau]\!]})\big)
\end{aligned}
$$

finally leads to

$$
[\![\mathbf{fix}\, x.M]\!] \;=\; \bigsqcup_n{}^{\uparrow} (f \circ \mathsf{up})^{n+2}(\perp_{[\![\tau]\!]}) \;=\; \bigsqcup_n{}^{\uparrow} (f \circ \mathsf{up})^{n}(\perp_{[\![\tau]\!]}).
$$

□

6.4 Computational Adequacy and Full Abstraction

Due to Lemma 6.2, we can use the SFL data types **nat**, **boolean**, **list**(σ) etc. in SFPL as well. Moreover, Lemma 6.2 can be used to transfer some lemmas and propositions of the previous section to SFPL. The first of the following results, however, needs to be proved again by a straightforward induction:

LEMMA 6.4. ***(Substitution Lemma)*** *Let $\Gamma, x{:}\sigma \vdash M : \tau$ and $\Gamma \vdash N : \sigma$ be SFPL terms-in-contexts. Then*

$$
[\![\Gamma \vdash M[N/x] : \tau]\!](d) \quad = \quad [\![\Gamma, x{:}\sigma \vdash M : \tau]\!]\big(d, [\![\Gamma \vdash N : \sigma]\!](d)\big)
$$

holds for all $d \in [\![\Gamma]\!]$. □

The equivalent result to Proposition 5.6 is:

COROLLARY 6.5. *Let M be a closed SFPL term with $[\![M]\!] \neq \perp$. Then M evaluates to a terminal computation (which is possibly a value).*

Proof. Replacing the **let**-constructor by smash product projections, every closed SFPL term M can be considered as SFL term. Since the denotations as well as the evaluation rules for closed SFPL terms are the same as for SFL terms, the claim follows from Proposition 5.6. □

As shown in the previous section, Lemma 6.4 and Corollary 6.5 suffice to prove computational adequacy:

COROLLARY 6.6. ***(Computational adequacy)*** *Let M be a closed SFPL term of type* **unit**. *Then M terminates if, and only if, $[\![M]\!] = \top$ holds.* □

The following result can also be transferred to SFPL:

COROLLARY 6.7. *Every compact element in the denotation of an SFPL type is definable in SFPL.*

Proof. Each compact element can be defined in SFL, thus, the claim follows from Lemma 6.2. $\square$

The category $SD_\perp$ does indeed provide a fully abstract model for SFPL:

COROLLARY 6.8. *(**Full abstraction**) Let M, N be closed SFPL terms of the same type. Then:*

$$[\![M]\!] \sqsubseteq [\![N]\!] \iff M \sqsubseteq_{SFPL} N$$

Proof. The interpretation of a context $C[\cdot_\sigma]$ of type **unit** with a hole of a computational type σ is a morphism $[\![C[\cdot_\sigma]]\!]\colon [\![\sigma]\!] \circ\!\!\rightarrow \mathbf{2}$ and, vice versa, for every definable function $f\colon [\![\sigma]\!] \circ\!\!\rightarrow \mathbf{2}$, there is a context $C_f[\cdot_\sigma]$ with $f = [\![\lambda x{:}[\![\sigma]\!].C_f[x]]\!]$. Therefore, observational inequivalence can be expressed semantically by applying Corollary 6.6. Let M, N be closed SFPL terms of type τ.

$$\begin{aligned}
M \sqsubseteq_{\mathrm{SFPL}} N &\iff \forall C[\cdot_\tau] : \mathbf{unit}.\ \ C[M] \sqsubseteq_{\mathrm{SFPL}} C[N] \\
&\iff \forall C[\cdot_{\tau_\perp}] : \mathbf{unit}.\ \ C[\mathbf{up}\ M] \sqsubseteq_{\mathrm{SFPL}} C[\mathbf{up}\ N] \\
&\iff \forall C[\cdot_{\tau_\perp}] : \mathbf{unit}.\ \ C[\mathbf{up}\ M]\Downarrow \implies C[\mathbf{up}\ N]\Downarrow \\
&\iff \forall C[\cdot_{\tau_\perp}] : \mathbf{unit}.\ \ [\![C[\mathbf{up}\ M]]\!] = \top \implies [\![C[\mathbf{up}\ N]]\!] = \top \\
&\iff \forall f\colon [\![\tau_\perp]\!] \circ\!\!\rightarrow \mathbf{2}\ \text{definable}.\ \ f([\![\mathbf{up}\ M]\!]) = \top \implies f([\![\mathbf{up}\ N]\!]) = \top
\end{aligned}$$

Such a function $f\colon [\![\tau_\perp]\!] \circ\!\!\rightarrow \mathbf{2}$ is definable in SFPL if, and only if, it is definable in SFL (Lemma 6.2), thus, the claim follows from the Full Abstraction Theorem 5.12 for SFL. $\square$

Moreover, the equational theory for SFPL is correct:

COROLLARY 6.9. *(**Soundness**) Let $\Gamma \vdash M = N : \tau$ be an SFPL equation-in-context. Then $[\![\Gamma \vdash M : \tau]\!] = [\![\Gamma \vdash N : \tau]\!]$ holds.*

Proof. As for SFL, the main part is to show that $[\![\Gamma \vdash M' : \sigma]\!](d)$ is different from $\perp_{[\![\sigma]\!]}$ (for all $d \in [\![\Gamma]\!]$) if $M'[N_1/x_1, \ldots, N_n/x_n]$ terminates for all well-typed terms N_i where $x_1, \ldots x_n$ are the variables occurring in Γ. Since all crucial lemmas (here Corollary 6.4, Corollary 5.2 and Corollary 6.7) hold for SFPL as well, the proof can be done in the same way as for SFL. The actual verification of the equations is straightforward. $\square$

7 Discussion

The Category SD

In this thesis, we have studied the category SD which is an adequate model for reasoning about sequential computation in the sense that it provides sufficient relational criteria characterising sequential functions. Admittedly, the bunch of relations associated with the domains arising as interpretations of SFL types is fairly large, it has the cardinality of the continuum. Therefore, for an arbitrary continuous function between two sequential domains, it is in general very difficult to check whether it is a morphism in SD. In the light of Loader's undecidability result for finite PCF [Loa96], it is unlikely that a substantial improvement can be achieved as it entails that the additional relational structure has to be infinite.

The category SD is a well-pointed, cpo-enriched, cartesian closed category where various domain theoretic constructions exist and satisfy the universal properties as exhibited in [Plo81]. In this sense, it is a *category of domains* and provides a domain theoretic framework for describing semantics of computation. Moreover, SD even contains the category DOM as a full subcategory. The category SD can be used as a fully abstract model for sequential programming languages encompassing the call-by-name and the call-by-value paradigm. In particular, SD has a model for FPC, which has been argued to be a good criterion for a category to be suitable for semantics [Fio96]. Although SD coincides with the Riecke/Sandholm model [RS97] on FPC type denotations, it not clear whether the subcategory $SSB_\perp$ is equivalent to Riecke's and Sandholm's category. Nevertheless, from my point of view, the presentation of SD captures the notion of sequentiality in a much more elegant way. Other domain theoretic models for sequential languages with sum types are either not fully abstract (as the Scott model) or do not cover call-by-name languages (as the Riecke/Sandholm model).

The Languages SFL and SFPL

The following quotation appears in [Mos90]:

> *A further aim of Denotational Semantics is to promote insight regarding the concepts underlying programming languages. Such insights might help to guide the design of new (and perhaps 'better') programming languages.*

The construction of the language SFL is guided by Denotational Semantics, it is designed in accordance with the structure of the category SD. As for PCF, the operational semantics fixes an observational equivalence where, for example, the terms $\lambda x.\Omega$ and Ω cannot be distinguished. This has the advantage that SFL gives a syntactic framework for reasoning about objects like the function space $\left[(\mathbb{N}_\perp)^n \to \mathbb{N}_\perp\right]$. This is neither possible in a call-by-value language where a function type is interpreted as the lifting of the space of strict functions ($[\![\sigma \to \tau]\!] = \left[[\![\sigma]\!] \circ\!\!\to [\![\tau]\!]\right]_\perp$) nor in call-by-name programming languages such as Haskell where denotations of function types are lifted function spaces ($[\![\sigma \to \tau]\!] = \left[[\![\sigma]\!] \to [\![\tau]\!]\right]_\perp$). Because of this lifting, the interpretations of $\lambda x.\Omega x$ and Ω are different for these languages. Therefore, unlike for SFL, the η-rule $\lambda x.Mx = M$ (where x

does not occur freely in M) usually fails. Although SFL is not likely to be used as a real programming language, it can be used as a metalanguage for the semantics of other (non-polymorphic) sequential languages as its operational semantics is close to domain theoretic structures.

As an application of the above, SFL can be used to extend a full abstraction result of realisability models for PCF. In general, realisability models provide a flexible tool for organising untyped models of computation into extensional models with a rich type structure. The fully abstract model for PCF in the category SD can be reconstructed as a realisability model over the partial combinatory algebra A which is obtained as the canonical solution of the domain equation $A \cong \mathbb{N}_\perp \oplus [A \to A]_\perp$, see [MRS99] or A. Rohr's forthcoming PhD thesis. The model is built by restricting A to those elements that appear as denotations of closed terms of a simple untyped language $\mathcal{L}$ related to the equation. This proves a variant of the Longley-Phoa Conjecture, claiming in its original form that realisability models over untyped λ-calculus are fully abstract for PCF. It has not been proved yet that the model in [MRS99] is *universal*, i. e. whether all elements of denotations of PCF types are definable in PCF[3], however, it is easy to see that they are definable in SFL. In order to generalise the approach above, replacing A by the solution of the domain equation $U \cong U + U + (U \times U) + [U \to U]$ in $SD_\perp$ gives rise to a universal realisability model of SFL.

Whereas the call-by-name language SFL appears as a restriction of a λ-calculus with products, sums and recursive types to an operationally meaningful fragment, SFPL is rather an extension of a call-by-value functional programming language subsuming the call-by-name paradigm. From a programmer's point of view this may seem to be a more natural approach. We have seen that both languages are equally expressive. It has also been shown that the subcategory SD provides a fully abstract model for SFPL within its subcategory $SD_\perp$.

Games versus Domains

In contrast to the game theoretic account of full abstraction, the present relational approach avoids the intermediate step of an *intensionally fully abstract model*, i. e. a model where all elements appear as (limits of) denotations of terms and all morphisms respect extensional equality. Instead, a sufficient collection of relational constraints is identified whose preservation (together with continuity) characterises sequential functions. In other words, the relational account of full abstraction identifies (relational) invariants of sequential computation and is *a priori* purely extensional in the sense that it avoids factorisation of an intermediary intensional model. This can be considered as an advantage from the point of view of an *extensional theory of computation* as any reference to operational notions is avoided. Accordingly, the relational approach is complementary to the game theoretic approach as the latter analyses the phenomenon of *sequentiality* from an operational point of view in a way such that most structural requirements of Domain Theory are fulfilled. However, (order) extensionality is not a basic property of game models and can usually only be obtained by taking the *extensional collapse*.

[3] This is actually equivalent to the problem whether SFL is a conservative extension of PCF.

Future Work

The category *SD* provides a model for sequential computation that is constructed without any reference to syntax. The proof of the full abstraction theorem, however, uses a particular family of syntactically specified relations. M. Fiore pointed out the problem of finding a purely semantical full abstraction proof for a suitable model. This problem seems to be rather challenging as (until now) there is no evidence why such a construction should exist and what it may look like.

Secondly, it would be desirable to extend the fully abstract model to a polymorphic extension of SFL. This may be helpful for obtaining fully abstract models for ALGOL-like languages, in particular with respect to their concept of local variables and control operators.

Another open problem is the integration of the relational and the game theoretic account of full abstraction. In analogy to [Ehr99], it might be interesting to study the connections between the domain theoretic model for SFL and the extensional collapse of its game model. In particular, it is an open question whether an arbitrary element of a type interpretation in *SD* comes from a strategy in the game model.

References

[AJ94] S. Abramsky and A. Jung. Domain theory. In S. Abramsky, D. M. Gabbay, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 3, pages 1–168. Clarendon Press, 1994.

[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*, pages 1–15. Springer Verlag, 1994. (Full version available from http://theory.doc.ic.ac.uk:80/tfm/papers/MalacariaP/PCFfullabs.dvi.Z).

[BE91] A. Bucciarelli and T. Ehrhard. Extensional embedding of a strongly stable model of PCF. In J. Leach Albert, B. Monien, and M. Rodriguez Artalejo, editors, *Automata, Languages and Programming*, volume 510 of *Lecture Notes in Computer Science*. Springer Verlag, 1991.

[Bek69] H. Bekič. Definable operations in general algebras, and the theory of automata and flowcharts. Technical report, IBM Laboratory, Vienna, 1969. Reprinted in H. Bekič, *Programming Languages and Their Definition*, volume 177 of *Lecture Notes in Computer Science*, pages 30–55. Springer Verlag, 1984.

[Ber79] G. Berry. *Modèles Complètement Adéquats et Stables des Lambda-calculs typés.* Thèse de Doctorat d'Etat, Université Paris VII, 1979.

[Ber90] U. Berger. *Totale Objekte und Mengen in der Bereichstheorie.* Dissertation, Universität München, 1990.

[CCF94] R. Cartwright, P.-L. Curien, and M. Felleisen. Fully abstract semantics for observably sequential languages. *Information and Computation*, 111:297–401, 1994.

[Cro93] R. Crole. *Categories for Types.* Cambridge Mathematical Textbooks. Cambridge University Press, 1993.

[Cur86] P.-L. Curien. *Categorical Combinators, Sequential Algorithms and Functional Programming.* Pitman, 1986.

[Cur93] P.-L. Curien. *Categorical Combinators, Sequential Algorithms and Functional Programming.* Progress in Theoretical Computer Science. Birkhäuser, second edition, 1993.

[Ehr99] T. Ehrhard. A relative PCF-definability result for strongly stable functions and some corollaries. *Information and Computation*, 152(1):111–137, July 1999.

[Ers75] Y. L. Ershov. Theorie der Numerierungen II. *Zeitschrift für mathematische Logik*, 21:473–584, 1975.

[Fio96] M. Fiore. *Axiomatic Domain Theory in Categories of Partial Maps.* Distinguished Dissertations in Computer Science. Cambridge University Press, 1996.

[Fre91] P. J. Freyd. Algebraically complete categories. In A. Carboni, M. C. Pedicchio, and G. Rosolini, editors, *Category Theory, Proc. Int. Conf. in Como, Italy, July 1990*, volume 1488 of *Lecture Notes in Mathematics*, pages 95–104. Springer Verlag, 1991.

[GS90] C. Gunter and D. S. Scott. Semantic domains. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B: Formal Models and Semantics, chapter 12, pages 633–674. Elsevier Science Publishers B.V., 1990.

[Gun92] C. Gunter. *Semantics of Programming Languages. Structures and Techniques.* Foundations of Computing. MIT Press, 1992.

[HO95] J. M. E. Hyland and C.-H. L. Ong. On full abstraction for PCF. Manuscript available from http://theory.doc.ic.ac.uk:80/tfm/papers/OngCLH/pcf.ps.gz, to appear in *Information and Computation*, 1995.

[JFM+96] A. Jung, M. Fiore, E. Moggi, P. O'Hearn, J. Riecke, G. Rosolini, and I. Stark. Domains and denotational semantics: History, accomplishments, and open problems. *Bulletin of the European Association for Theoretical Compter Science*, pages 227–256, June 1996.

[JT93] A. Jung and J. Tiuryn. A new characterization of lambda definability. In M. Bezem and J. F. Groote, editors, *Typed Lambda Calculi and Applications*, volume 664 of *Lecture Notes in Computer Science*, pages 245–257. Springer Verlag, 1993.

[Lev99] P. B. Levy. Call-by-push-value: A subsuming paradigm (extended abstract). In J.-Y. Girard, editor, *Typed Lambda-Calculi and Applications*, volume 1581 of *Lecture Notes in Computer Science*. Springer-Verlag, April 1999.

[Loa96] R. Loader. Finitary PCF is not decidable. Unpublished manuscript available from http://hypatia.dcs.qmw.ac.uk, July 1996.

[Mac71] S. Mac Lane. *Categories for the Working Mathematician*. Springer Verlag, Berlin, 1971.

[MC88] A. R. Meyer and S. S. Cosmadakis. Semantical paradigms: Notes for an invited lecture. In *Third Annual Symposium on Logic in Computer Science*, pages 236–253. IEEE Computer Society Press, 1988.

[McC96] G. McCusker. Games and full abstraction for FPC. In *Eleventh Annual IEEE Symposium on Logic in Computer Science*, pages 174–183. IEEE Computer Society Press, 1996.

[Mil77] R. Milner. Fully abstract models of typed lambda-calculi. *Theoretical Computer Science*, 4:1–22, 1977.

[Mit96] J. C. Mitchell. *Foundations for Programming Languages*. Foundations of Computing. The MIT Press, Cambridge, Massachusetts, 1996.

[Mos90] P. D. Mosses. Denotational semantics. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B: Formal Models and Semantics, chapter 11, pages 575–632. Elsevier Science Publishers B.V., 1990.

[MRS99] M. Marz, A. Rohr, and T. Streicher. Full abstraction and universality via realisability. In *Fourteenth Annual IEEE Symposium on Logic in Computer Science*, pages 174–182. IEEE Computer Society Press, 1999.

[Mul87] K. Mulmuley. *Full Abstraction and Semantic Equivalence*. The MIT Press, 1987.

[Nic94] H. Nickau. Hereditarily sequential functionals. In *Proceedings of the Symposium on Logical Foundations of Computer Science: Logic at St. Petersburg*, Lecture Notes in Computer Science. Springer Verlag, 1994.

[Ong95] C.-H. L. Ong. Correspondence between operational and denotational semantics: the full abstraction problem for PCF. In S. Abramsky, D. M. Gabbay, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 4, pages 269–356. cp, 1995.

[OR95a] P. W. O'Hearn and J. C. Reynolds. From Algol to polymorphic linear lambda calculus. Lectures at Isaac Newton Institute for Mathematical Sciences, Cambridge, 1995.

[OR95b] P. W. O'Hearn and J. G. Riecke. Kripke logical relations and PCF. *Information and Computation*, 120(1):107–116, 1995.

[OT93] P. W. O'Hearn and R. D. Tennent. Relational parametricity and local variables. In *Conference Record 20th Symposium on Principles of Programming Languages*, pages 171–184, 1993.

[Pit96] A. M. Pitts. Relational properties of domains. *Information and Computation*, 127:66–90, 1996.

[Plo73] G. D. Plotkin. Lambda-definability and logical relations. Memorandum SAI-RM-4, University of Edinburgh, October 1973.

[Plo77] G. D. Plotkin. LCF considered as a programming language. *Theoretical Computer Science*, 5:223–255, 1977.

[Plo80] G. D. Plotkin. Lambda-definability in the full type hierarchy. In J. P. Seldin and J. R. Hindley, editors, *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 363–373. Academic Press, London, 1980.

[Plo81] G. D. Plotkin. Post-graduate lecture notes in advanced domain theory (incorporating the "Pisa Notes"). Dept. of Computer Science, Univ. of Edinburgh, 1981.

[Plo85] G. D. Plotkin. Types and partial functions. Notes for Lectures at CSLI, Stanford University, 1985.

[RS97] J. G. Riecke and A. Sandholm. A relational account of call-by-value sequentiality. In *Twelfth Annual IEEE Symposium on Logic in Computer Science, LICS'97*, pages 258–267, 1997.

[Saz75] V. Yu. Sazonov. Sequentially and partially computable functionals. In C. Böhm, editor, *Proc. Symp. λ-Calculus and Computer Science theory*, volume 37 of *Lecture Notes in Computer Science*, pages 312–318. Springer Verlag, 1975.

[Saz76] V. Yu. Sazonov. O Wyrasimosti Funkzionalov w jasyke D. Scotta LCF. *Algebra i Logika*, 15(3):308–330, 1976. Translated into English in *Algebra and Logic.*

[Sco69] D. S. Scott. A type theoretical alternative to ISWIM, CUCH, OWHY. Manuscript, University of Oxford, 1969.

[Sco93] D. S. Scott. A type theoretical alternative to ISWIM, CUCH, OWHY. *Theoretical Computer Science*, 121:411–440, 1993. Reprint of a manuscript written in 1969.

[Sie92] K. Sieber. Reasoning about sequential functions via logical relations. In M. P. Fourman, P. T. Johnstone, and A. M. Pitts, editors, *Proc. LMS Symposium on Applications of Categories in Computer Science, Durham 1991*, volume 177 of *LMS Lecture Note Series*, pages 258–269. Cambridge University Press, 1992.

[Smy83] M. B. Smyth. The largest cartesian closed category of domains. *Theoretical Computer Science*, 27:109–119, 1983.

[SP82] M. B. Smyth and G. D. Plotkin. The category-theoretic solution of recursive domain equations. *SIAM J. Computing*, 11(4):761–783, 1982.

[Win93] G. Winskel. *The Formal Semantics of Programming Languages. An Introduction.* MIT Press, 1993.