Electronic Notes in Theoretical Computer Science 35 (2000)
URL: http://www.elsevier.nl/locate/entcs/volume35.html 20 pages

A fully abstract model for sequential computation ⋆

Michael Marz

Fachbereich Mathematik, Technische Universität Darmstadt
Schloßgartenstraße 7, D-64289 Darmstadt, Germany
e-mail: marz@mathematik.tu-darmstadt.de

1 Introduction

In 1977, G. Plotkin pointed out the problem of finding a fully abstract model for the sequential programming language PCF [16], which had been originally developed by D. Scott [19]. This question turned out to be one of the most enduring problems of semantics. A very nice description of the different approaches to this subject with many references can be found in [5]. In this rather brief overview, we mainly focus on those articles which are related to the work in this paper.

The first fully abstract model for PCF was presented by R. Milner [11]. However, it was purely syntactic and did not provide a semantic description of sequentiality. In 1992, K. Sieber presented a syntax free characterisation of sequentiality for types up to order two and proved a full abstraction result for PCF types up to order three [20]. 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 [14] by applying the concept of *Kripke relations* [6]. A different concept was used by several research groups elaborating fully abstract models for PCF by using *game semantics*, see [2,4,13]. G. McCusker extended this approach to sum types and recursive types [10]. The domain theoretic approach was refined by J. Riecke and A. Sandholm who modified the O'Hearn/Riecke model and presented a fully abstract model for FPC, an extension of call-by-value PCF with sum types and recursive types [18].

After an introduction to some domain theoretic concepts and notations (Section 2) we construct a cartesian closed category *SD* of domains to capture the concept of sequentiality (Section 3). This category is cartesian closed, besides, it has *smash products* making the subcategory $SD_\perp$ with strict functions

⋆ This paper is an extended abstract of a major part of my PhD thesis. It contains all important results of a technical report that I wrote during a one year stay at the University of Birmingham, however, most of the proofs and many details are omitted. For a more detailed version please look at [8].

©2000 Published by Elsevier Science B. V.

symmetric monoidal closed. 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 the convergence problem is sequentially semidecidable. We discuss several properties of *SD* and show how to construct $SD_\perp$ from the subcategory of its SSB-objects.

In Section 4, we have a look at the sequential language SFL, a PCF-like language with a host of important data type constructors (two forms of sums and products, lifting, strict and ordinary function spaces and recursive types). It is universal in the sense that it is close to the mathematical structure of domains, and, hence, can be used as a metalanguage for (non-polymorphic) sequential programming languages. Besides, SFL is powerful enough to deal with call-by-value evaluation strategies as well as with call-by-name strategies.

Finally, we prove that *SD* contains a fully abstract model for SFL.

On the one hand, 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 [17]. In this sense, it is a *category of domains*, and, as *DOM*, provides a domain theoretic framework for describing semantics of computation. On the other hand, *SD* can be used as a framework for a fully abstract model for sequential programming languages like PCF or FPC. Other domain theoretic models for languages with sum types are either not fully abstract (as the Scott model) or not based on a cartesian closed category (as the Riecke/Sandholm model). Both of these models live as subcategories in *SD*.

The language SFL is more expressive than PCF and FPC. As for PCF, the operational semantics is based on observational equivalence which means that, for example, the terms $\lambda x.\Omega$ and Ω cannot operationally be distinguished. This has the advantage that SFL gives a syntactic framework to reason about objects like the function space $\left[(\mathrm{N}_\perp)^n \to \mathrm{N}_\perp\right]$ (where $\mathrm{N}_\perp$ denotes the flat domain of natural numbers). This is neither possible in a call-by-value language where function types are denotated by the lifting of the space of strict functions ($[\![\sigma \to \tau]\!] = \left[[\![\sigma]\!] \circ\!\!\!\to [\![\tau]\!]\right]_\perp$) nor in a *lazy* call-by-name language where the denotation of function types is the lifting of the space of total functions ($[\![\sigma \to \tau]\!] = \left[[\![\sigma]\!] \to [\![\tau]\!]\right]_\perp$). Besides, the η-rule in those languages usually fails which is not the case in SFL. Moreover, since the operational semantics of SFL is close to domain theoretic structures, it is universal in the sense that it can be used as a metalanguage for the semantics of other (non-polymorphic) sequential languages.

In contrast to the game-theoretic account of full abstraction, the present relational approach to full abstraction avoids the intermediate step of an *intensional fully abstract model* (where by the extensional collapse intensional objects may get identified but never will be excluded). 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 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 in which most structural requirements of domain theory are fulfilled with the notable exception of (order) extensionality.

2 Fundamental domain theoretic concepts and notations

This section gives a short summary of the domain theory used in this paper. More details can be found, for instance, in [1] or [17].

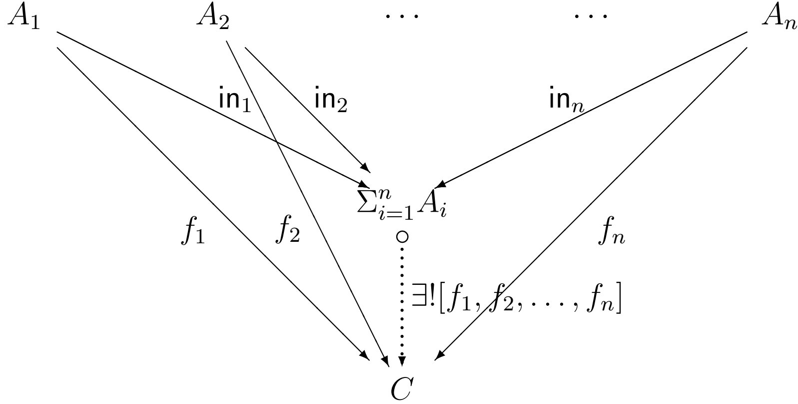
A *domain* A is a partially ordered set that is closed under directed suprema and has a least element $\bot_A$. A function $f\colon A \to B$ is called *continuous* if it is monotone and preserves directed suprema. We write $f\colon A \circ\!\!\to B$ to indicate that a function is *strict*, i.e. it maps $\bot_A$ to $\bot_B$. The domain of all continuous functions between two domains A and B with the pointwise order is denoted by $[A \to B]$, the domain of all strict continuous functions by $[A \circ\!\!\to B]$. We write *DOM* for the category of domains with continuous functions and $DOM_\bot$ for the subcategory with strict continuous functions.

If f is a function from a set w to a domain A and if v is a subset of w, then the function $f \restriction v\colon w \to A$ is defined as $(f \restriction v)(x) = f(x)$ if $x \in v$ and $(f \restriction v)(x) = \bot_A$ otherwise. We use $\breve{\ } x{:}A.e[x]$ (where $e[x]$ is an expression that may contain x) for abstraction on the meta-level.

For a domain A, $A_\bot := \{\mathsf{up}_A\, a \mid a \in A\} \cup \{\bot_{A_\bot}\}$ with the induced order denotes the *lifting* of A; the maps $\mathsf{up}_A\colon A \to A_\bot$ and $\mathsf{down}_A\colon A_\bot \circ\!\!\to A$ are defined as $\mathsf{down}_A(\mathsf{up}_A\, x) = x$ and $\mathsf{down}_A(\bot_{A_\bot}) = \bot_A$. We often omit the index when it is clear from the context and we write $\mathbf{2} := \mathbf{1}_\bot$ as an abbreviation for the domain with two elements (*Sierpiński domain*).

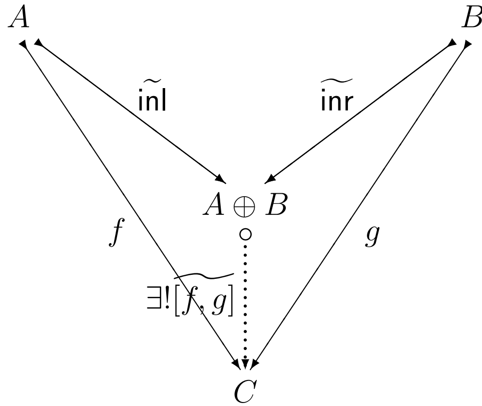
The category *DOM* contains two different kinds of sums. Firstly, the *separated sum* $\Sigma_{i=1}^n A_i = A_1 + \cdots + A_n$ of domains $A_1, \ldots, A_n$ is given by the set $\{\mathsf{in}_i\, a \mid i = 1, \ldots, n\} \cup \{\bot_{\Sigma_{i=1}^n A_i}\}$, the order being defined as $a \sqsubseteq_{\Sigma_{i=1}^n A_i} b$ if, and only if, $a = \bot_{\Sigma_{i=1}^n A_i}$ or $\exists i{=}1, \ldots, n.\ a = \mathsf{in}_i\, a_1,\ b = \mathsf{in}_i\, a_2$ and $a_1 \sqsubseteq_{A_i} a_2$ hold.

It satisfies the following universal property:



That means that, for each n morphisms $f_i \colon A_i \to C$ $(i = 1, \ldots, n)$, there is exactly one strict morphism $[f_1, \ldots, f_n] \colon \Sigma_{i=1}^n A_i \to C$ such that the diagram commutes. (Note that there may be several non-strict maps making the diagram commute.)

The *coalesced sum* $A \oplus B$ of domains A, B is the quotient of $A + B$ where $\mathsf{in}_1 \perp_A$, $\mathsf{in}_2 \perp_B$ and $\perp_{A+B}$ are identified. The elements of $A \oplus B$ are denoted by $\widetilde{\mathsf{inl}}\, a$, $\widetilde{\mathsf{inr}}\, b$ and $\perp_{A \oplus B}$ $(a \in A, b \in B)$. Since the coalesced sum is associative, one can easily construct all finitary coalesced from binary ones. Coalesced sums satisfy a universal property, too, they are categorical sums in the subcategory $DOM_\perp$ of DOM:



In this diagram, the strictness of f and g is crucial. Otherwise, the mediating morphism

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

does not necessarily make the diagram commute. As a consequence, the constructor $\oplus$ is not a functor on DOM. For example,

$$(\breve{\ } x{:}\mathbf{2}.\top_{\mathbf{2}} \circ \breve{\ } x{:}\mathbf{2}.\perp_{\mathbf{2}}) \oplus (\mathsf{id}_{\mathbf{2}} \circ \mathsf{id}_{\mathbf{2}}) = \big((\breve{\ } x{:}\mathbf{2}.\top_{\mathbf{2}}) \oplus \mathsf{id}_{\mathbf{2}}\big)$$

maps $\mathsf{inl}\,\top_{\mathbf{2}}$ to $\mathsf{inl}\,\top_{\mathbf{2}}$ whereas

$$\big((\breve{}x{:}\mathbf{2}.\top_{\mathbf{2}})\oplus\mathsf{id}_{\mathbf{2}}\big)\circ\big((\breve{}x{:}\mathbf{2}.\bot_{\mathbf{2}})\oplus\mathsf{id}_{\mathbf{2}}\big)(\mathsf{inl}\,\top_{\mathbf{2}}) = \big((\breve{}x{:}\mathbf{2}.\top_{\mathbf{2}})\oplus\mathsf{id}_{\mathbf{2}}\big)(\bot_{\mathbf{2}\oplus\mathbf{2}}) = \bot_{\mathbf{2}\oplus\mathbf{2}}$$

holds. Nevertheless, $\oplus$ is functorial on the subcategory $DOM_\bot$.

The *smash product* (also called *tensor product*) $A \otimes B$ of domains A, B 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 $\mathsf{smash}\colon A \times B \to A \otimes B$ which maps $\langle a, b\rangle \in A \times B$ to $(a, b) \in (A \otimes B)$ where $a = \bot_A$ or $b = \bot_B$ already implies $(a, b) = \bot_{A\otimes B}$. Note that the projections $\widetilde{\mathsf{pr}_1}\colon A \otimes B \multimap A$ and $\widetilde{\mathsf{pr}_2}\colon A \otimes B \multimap B$ exist in DOM.

Similar to coalesced sums the constructors $\otimes$ and $\multimap$ are not functorial on DOM, however, they are functors on the subcategory $DOM_\bot$.

3 The category *SD*

In the original Scott model there are continuous functions which are not definable in a sequential functional programming language like PCF. In the category presented in this paper this is not possible. The approach here is to require the morphisms to conserve a special kind of relations. This concept goes back to previous works ([14,18]), however, the model here can be used for a wider variety of languages, and, from a categorical point, satisfies more desirable properties.

We first need a preliminary definition which is a simplification of Riecke's and Sandholm's concept of *computational theories* [18]. 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) $\{w\} \in S$,

(P2) if $\{v_1, \dots, v_{n-1}, v_n\} \in S$ then $\{v_1, \dots, v_{n-1}\} \in S$ *(Dropping)*,

(P3) if $\{v_1, \dots, v_{n-1}, v_n\} \in S$ then $\{v_1, \dots, v_{n-1} \cup v_n\} \in S$ *(Joining)*,

(P4) if $\{v_1, \dots, v_{n-1}, v_n\}, \{w_1, \dots, w_m\} \in S$
then $\{v_1, \dots, v_{n-1}\} \cup \{v_n \cap w_i \mid i = 1, \dots, m,\ v_n \cap w_i \neq \varnothing\} \in S$ *(Refinement)*.

It happens very often that one of the constructed sets of a partial partition turns out to be empty. To avoid notational hassle, we allow $\{v_1, \dots, v_n, \varnothing\}$ as a notion for the partial partition $\{v_1, \dots, v_n\}$. For instance, the last set in condition (P4) can be rewritten as $\{v_1, \dots, v_{n-1}\} \cup \{v_n \cap w_i \mid i = 1, \dots, m\}$.

Sequentiality systems capture the concept of sequentiality in an abstract way: a system of partial partitions for which it is sequentially semidecidable to which class of a partition a given element belongs does necessarily satisfy

conditions (P1) – (P4).

A sequentiality system S on w can also be presented by a family $(S_n)_{n\in\mathrm{N}}$ of nonempty sets of functions $\varphi\colon w \to \overline{n}_\perp$ (where $\overline{n}_\perp$ stands for the flat domain with n maximal elements $1,\ldots,n$ and a least element $\perp$) naming the equivalence classes of a partial partition in S. Such a family $(S_n)_{n\in\mathrm{N}}$ 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 $\Rightarrow$ $\mu\circ\varphi \in S_m$,

(S2) $\varphi \in S_2, \psi_1 \in S_{m_1}, \psi_2 \in S_{m_2}$ $\Rightarrow$ $\theta\colon w \to \overline{(m_1+m_2)}_\perp \in S_{m_1+m_2}$, where θ is defined as $\theta(x) = \begin{cases} \psi_1(x) & \text{if } \varphi(x)=1 \\ m_1+\psi_2(x) & \text{if } \varphi(x)=2 \\ \perp & \text{if } \varphi(x)=\perp \end{cases}$.

It can be proved that both definitions are equivalent in the sense that there is a one-to-one-correspondence between sequentiality systems on w and sequential function systems on w.

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, by using the second definition one can easily show that every $\mathcal{C}$-indexed family of intersections of relations

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

$(v_1 \subseteq v_2 \subseteq w)$ as originally defined by Kurt Sieber [20] fits in this more general approach, 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 $\mathrm{N}_\perp$ and we restrict ourselves to finite approximations $\overline{n}_\perp$. This does not affect the standard proof of por not being a morphism: $(1,\perp,\perp),(\perp,1,\perp) \in S_2 := R^{\{1,2,3\}}_{\{1,2\},\{1,2,3\}}$ holds, but $\mathsf{por}\big((1,\perp,\perp),(\perp,1,\perp)\big) = (1,1,\perp)$ is not related by S_2.

In order to deal with function types we need families of sequentiality systems over different sets w that are stable under re-indexing. This idea goes back to A. Jung and J. Tiuryn [6] who introduced *Kripke relations* with varying arity in order to characterise definability for the simply typed λ-calculus. The approach here is to define an index category *SPIC* (for sequentiality partition index category) whose objects $\mathsf{w} = (w, S)$ consist of a finite subset $w \subseteq \mathrm{N}$ 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*:

$$\{v_1,\ldots,v_n\} \in S^{\mathsf{w}} \quad\Rightarrow\quad \{\mu^{-1}(v_1),\ldots,\mu^{-1}(v_n)\} \in S^{\mathsf{v}}$$

(analogously, $\varphi \in S^{\mathsf{w}}_n, \quad\Rightarrow\quad \varphi\circ\mu \in S^{\mathsf{v}}_n$ for sequential function systems).

The concept of sequentiality systems induces an extension to *logical relations* on arbitrary domains. We write a $|w|$-ary relation on a domain A as a

function $f\colon w \to A$ where $f(x)$ gives the x-th component of the corresponding tuple.

Definition 3.2 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) R is closed under directed suprema *(Admissibility)*,

(R2) $\forall a{\in}A.\ \breve{}x{:}w.a \in R$ *(Reflexivity)*,

(R3) $f \in R$, $\{v\} \in S \ \Rightarrow\ f \restriction v \in R$ *(Restriction)*,

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

The extension to Kripke relations is the following:

Definition 3.3 Let $\mathcal{C}$ be a subcategory of *SPIC*. A family $R = (R^{\mathsf{w}})_{\mathsf{w} \in \mathsf{Obj}(\mathcal{C})}$ of w-logical relations on a domain A is called $\mathcal{C}$-*Kripke logical relation* on A if it is $\mathcal{C}$-*Kripke monotone*:

$$f \in R^{\mathsf{w}},\ \mu\colon \mathsf{v} \xrightarrow{\mathcal{C}} \mathsf{w} \quad \Rightarrow \quad f \circ \mu \in R^{\mathsf{v}}.$$

If $\mathcal{C}$ is a subcategory of *SPIC* and A a domain, then the set $\mathcal{R}$ of all $\mathcal{C}$-Kripke logical relations on A forms a closure system.

The category *SD* (for sequentiality domains) can now be 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

$$g \in Q^{\mathsf{w}}_{\mathcal{C}} \quad \Rightarrow \quad f \circ g \in R^{\mathsf{w}}_{\mathcal{C}}$$

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

As claimed in the following theorem, this category is closed under the usual domain theoretic closure properties, i.e. it is cartesian closed and contains the different kinds of products, sums and functions satisfying the universal domain theoretic properties as exhibited in [17]. Hence, it can be described by a language similar to the simply typed λ-calculus whose operational semantics is defined in accordance with domain theoretic structures. For instance, product types are interpreted as cartesian products rather than by smash products (like in call-by-value languages) or by lifted cartesian products (like in lazy call-by-name languages). Such a language will be examined in the next section.

Theorem 3.4 *The category SD is cpo-enriched, well-pointed and cartesian closed. It contains smash products, liftings, separated sums and coalesced sums satisfying the universal properties as explained in the previous section. Moreover, the smash products make the subcategory* $SD_{\perp}$ *with strict functions*

symmetric monoidal closed. 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 the constructions $A \times B$, $A \otimes B$ etc. are defined as

$$\mathbf{1}^{\mathsf{w}}_{\mathcal{C}} := \{\breve{}x{:}w.\top\},$$
$$(Q \times R)^{\mathsf{w}}_{\mathcal{C}} := \{\langle g, h\rangle \colon w \to A \times B \mid g \in Q^{\mathsf{w}}_{\mathcal{C}}, h \in R^{\mathsf{w}}_{\mathcal{C}}\},$$
$$(Q \otimes R)^{\mathsf{w}}_{\mathcal{C}} := \mathsf{cl}^{\mathsf{w}}\big(\{(g, h) \colon w \to A \otimes B \mid g \in Q^{\mathsf{w}}_{\mathcal{C}}, h \in R^{\mathsf{w}}_{\mathcal{C}}\}\big),$$
$$(Q_\perp)^{\mathsf{w}}_{\mathcal{C}} := \big\{f \colon w \to A_\perp \mid \big\{\{x \in w \mid f(x) \in A\}\big\} \in S, \mathsf{down} \circ f \in Q^{\mathsf{w}}_{\mathcal{C}}\big\},$$
$$(\Sigma^n_{i=1} Q_i)^{\mathsf{w}}_{\mathcal{C}} := \Big\{f \colon w \to \Sigma^n_{i=1} A_i \mid \exists\varphi{\in}S_n.\exists g_i{\in}(Q_i)^{\mathsf{w}}_{\mathcal{C}}.\ f(x) = \begin{cases} \mathsf{in}_i \circ g_i(x) & \text{if } \varphi x = i \\ \perp & \text{if } \varphi x = \perp \end{cases}\Big\},$$
$$(Q \oplus R)^{\mathsf{w}}_{\mathcal{C}} := \Big\{f \mid \exists\varphi{\in}S_2.\exists g{\in}Q^{\mathsf{w}}_{\mathcal{C}}.\exists h{\in}R^{\mathsf{w}}_{\mathcal{C}}.\ f(x) = \begin{cases} \widetilde{\mathsf{inl}} \circ g(x) & \text{if } \varphi(x) = 1 \\ \widetilde{\mathsf{inr}} \circ h(x) & \text{if } \varphi(x) = 2 \\ \perp & \text{if } \varphi(x) = \perp \end{cases}\Big\},$$
$$[Q \to R]^{\mathsf{w}}_{\mathcal{C}} := \big\{f \colon w \to [A \to B] \mid \forall\mu \colon \mathsf{v} \xrightarrow{\mathcal{C}} \mathsf{w}.\forall g{\in}Q^{\mathsf{v}}_{\mathcal{C}}.\ \breve{}x{:}v.\big(f(\mu x)\big)(gx) \in R^{\mathsf{v}}_{\mathcal{C}}\big\},$$
$$[Q \circ\!\!\to R]^{\mathsf{w}}_{\mathcal{C}} := \big\{f \colon w \to [A \circ\!\!\to B] \mid \forall\mu \colon \mathsf{v} \xrightarrow{\mathcal{C}} \mathsf{w}.\forall g{\in}Q^{\mathsf{v}}_{\mathcal{C}}.\ \breve{}x{:}v.\big(f(\mu x)\big)(gx) \in R^{\mathsf{v}}_{\mathcal{C}}\big\}$$

for every subcategory $\mathcal{C}$ of the index category SPIC and every object $\mathsf{w} = (w, S) \in \mathsf{Obj}(\mathcal{C})$ where $\mathsf{cl}^{\mathsf{w}}(R)$ is the least $|w|$-ary admissible relation containing R. (We write $\mathbf{1}$ for the singleton domain as well as for the family of its Kripke relations.) This implies that the relations on $\mathbf{2} := \mathbf{1}_\perp$ are given by $\mathbf{2}^{\mathsf{w}}_{\mathcal{C}} = (\mathbf{1}_\perp)^{\mathsf{w}}_{\mathcal{C}} = S_1$.

As in *DOM*, coalesced sum, smash product and strict function space constructors are not functors on *SD*, but 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 $DOM_\perp$. This allows to interpret recursive types as bilimits of expanding sequences in *SD*.

For some objects in *SD* the question of whether an arbitrary element is not the bottom element is sequentially semidecidable. This is formalised in the following definition:

Definition 3.5 An object $(A, \mathcal{Q})$ in *SD* is called *SSB-object* (for sequentially separable bottom) if there is a morphism $\chi_A \colon (A, \mathcal{Q}) \to (\mathbf{2}, \mathbf{2})$ 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*.

The question arises what objects in *SD* are SSB-objects. The following lemma shows which constructors preserve the SSB property:

Proposition 3.6 (i) *If $(A, \mathcal{Q})$ is an arbitrary object, then $(A_\perp, \mathcal{Q}_\perp)$ is an SSB-object.*

(ii) *If $(A, \mathcal{Q})$ and $(B, \mathcal{R})$ are SSB-objects, then so are $(A \otimes B, \mathcal{Q} \otimes \mathcal{R})$ and $(A \oplus B, \mathcal{Q} \oplus \mathcal{R})$.*

(iii) *If $(A_1, Q_1), \ldots, (A_, \mathcal{Q}_n)$ are arbitrary objects, then $(\Sigma_{i=1}^n A_i, \Sigma_{i=1}^n \mathcal{Q}_i)$ is an SSB-object.*

Note that the cartesian product of SSB-objects does not necessarily satisfy the SSB property. For example, $(\mathbf{2} \times \mathbf{2}, \mathbf{2} \times \mathbf{2})$ is not an SSB-object. This can be shown by using Sieber's relation from page 6 again: the triples $(\top_\mathbf{2}, \perp_\mathbf{2}, \perp_\mathbf{2})$ and $(\perp_\mathbf{2}, \top_\mathbf{2}, \perp_\mathbf{2})$ are related by $S_1 := R^{\{1,2,3\}}_{\{1,2\},\{1,2,3\}}$, but

$$\big(\chi_{\mathbf{2}\times\mathbf{2}}(\top_\mathbf{2}, \perp_\mathbf{2}), \chi_{\mathbf{2}\times\mathbf{2}}(\perp_\mathbf{2}, \top_\mathbf{2}), \chi_{\mathbf{2}\times\mathbf{2}}(\perp_\mathbf{2}, \perp_\mathbf{2})\big) = (\top_\mathbf{2}, \top_\mathbf{2}, \perp_\mathbf{2})$$

is not. Hence, $\chi_{\mathbf{2}\times\mathbf{2}}$ is not a morphism in *SD*, and, therefore, $(\mathbf{2} \times \mathbf{2}, \mathbf{2} \times \mathbf{2})$ is not an SSB-object.

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

Proposition 3.7 *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:*

$$(Q \otimes R)^{\mathsf{w}}_{\mathcal{C}} = \{(g, h) \colon w \to A \otimes B \mid g \in Q^{\mathsf{w}}_{\mathcal{C}}, h \in R^{\mathsf{w}}_{\mathcal{C}}\},$$

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

This proposition does not hold if we omit the SSB property. A modification of an example in [15] can be used as an example where the conclusion of Proposition 3.7 is wrong for non-SSB-objects. Besides, smash products on SSB-objects have projections $\widetilde{\mathsf{pr}_1} \colon (A \otimes B, \mathcal{Q} \otimes \mathcal{R}) \to (A, \mathcal{Q})$ and $\widetilde{\mathsf{pr}_2} \colon (A \otimes B, \mathcal{Q} \otimes \mathcal{R}) \to (B, \mathcal{R})$ which is not the case for non-SSB-objects.

Furthermore, SSB-objects can also be used to make the relation between the model of Riecke/Sandholm and the category *SD* explicit: the subcategory $SSB_\perp$ of SSB-objects with strict functions is equivalent to the category that J. Riecke and A. Sandholm used as a fully abstract model for FPC in [18]. In other words, the category *SD* contains a subcategory that is a fully abstract model for FPC.

Finally, the category $SD_\perp$ can be constructed from its SSB-objects. As Alex Simpson pointed out, it is the Eilenberg-Moore category of algebras in SSB_{tot} (where SSB_{tot} is the category of SSB-objects with strict, total functions, i.e. strict functions reflecting the bottom element) with respect to the lifting monad.

4 The language SFL

In this section we discuss the language SFL (for sequential functional language) which can be considered as a universal metalanguage for sequential computations. The requirement of sequentiality makes it impossible to define all type constructors in the usual way without any constraints. For instance, a sequential evaluation of a $\widetilde{\mathbf{case}}$-term for a coalesced sum type requires a sequential strategy giving a result if a given input term terminates.

The approach in this paper to deal with this problem is to use two different kinds of SFL types. A similar approach was studied by A. Meyer and S. Cosmadakis in [9], although they do not deal with smash products. Arbitrary types are denoted by the letters $\tau, \tau_1, \tau_2, \ldots$. Types whose termination is sequentially semidecidable are called *computational types* or *SSB-types*, they are denoted by $\sigma, \sigma_1, \sigma_2, \ldots$. It turns out that those types are interpreted as SSB-objects in *SD*. The grammar for the types is given by

$$\sigma \quad ::= \quad \alpha \mid \tau_\perp \mid \sigma \otimes \sigma \mid \Sigma_{i=1}^n \tau \mid \sigma \oplus \sigma \mid \mu\alpha.\sigma$$

$$\tau \quad ::= \quad \sigma \mid \tau \times \tau \mid \tau \to \tau \mid \sigma \circ\!\!\to \tau \mid \mu\alpha.\tau$$

where α ranges over the set of all type variables. SFL terms are given by the grammar

$$\begin{aligned} M ::= {} & x \mid \langle M, M\rangle \mid \mathbf{pr_1}\ M \mid \mathbf{pr_2}\ M \mid (M, M) \mid \widetilde{\mathbf{pr_1}}\, M \mid \widetilde{\mathbf{pr_2}}\, M \mid \mathbf{in}_i^{\tau_1,\ldots,\tau_n} M \mid \\ & \mathbf{case}^{\tau,\ldots,\tau,\tau} M \,\mathbf{of}\ \mathbf{in}_i\, x \Rightarrow M, \ldots, M \mid \widetilde{\mathbf{inl}}^{\sigma,\sigma} M \mid \widetilde{\mathbf{inr}}^{\sigma,\sigma} M \mid \\ & \widetilde{\mathbf{case}}^{\sigma,\sigma,\tau} M \,\mathbf{of}\ \mathbf{inl}\, x \Rightarrow M; \mathbf{inr}\, x \Rightarrow M \mid \mathbf{up}\, M \mid \mathbf{down}\, M \mid \lambda x{:}\tau.M \mid \\ & MM \mid \widetilde{\lambda} x{:}\sigma.M \mid M^{\sim}M \mid \mathbf{fold}^{\mu\alpha.\tau} M \mid \mathbf{unfold}^{\mu\alpha.\tau} M \end{aligned}$$

where x ranges over the set of all variables. Type annotations are often omitted when they are clear from the context. Terms corresponding to types with circled connectives have tildes, they are evaluated by a call-by-value strategy.

In the following typing rules we assume that all occurring types are closed.

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

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

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

$$\frac{\Gamma \vdash M : \tau_1}{\Gamma \vdash \mathbf{in}_1^{\tau_1,\ldots,\tau_n} M : \Sigma_{i=1}^n \tau_i} \quad \cdots \quad \frac{\Gamma \vdash M : \tau_n}{\Gamma \vdash \mathbf{in}_n^{\tau_1,\ldots,\tau_n} M : \Sigma_{i=1}^n \tau_i}$$

$$\frac{\Gamma \vdash M : \Sigma_{i=1}^{n}\tau_i \qquad \Gamma, x{:}\tau_i \vdash M_i : \tau \qquad \cdots \qquad \Gamma, x{:}\tau_n \vdash M_n : \tau}{\Gamma \vdash \mathbf{case}^{\tau_1,\ldots,\tau_n,\tau} M \,\mathbf{of}\ \mathbf{in}_i\, x \Rightarrow M_1, \ldots, M_n : \tau}$$

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

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

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

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

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

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

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

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

The evaluation strategy for SFL is given by:

$$\frac{}{\langle M_1, M_2\rangle \Downarrow \langle M_1, M_2\rangle}$$

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

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

$$\frac{}{\mathbf{in}_1\, M \Downarrow \mathbf{in}_1\, M} \quad \cdots \quad \frac{}{\mathbf{in}_n\, M \Downarrow \mathbf{in}_n\, M}$$

$$\frac{M \Downarrow \mathbf{in}_j\, N \qquad M_j[N/x] \Downarrow V}{\mathbf{case}\, M\, \mathbf{of}\ \mathbf{in}_i\, x \Rightarrow M_1, \ldots, M_n \Downarrow V} \qquad \text{(for all } j = 1, \ldots, n)$$

$$\frac{M \Downarrow V}{\widetilde{\mathbf{inl}}\, M \Downarrow \widetilde{\mathbf{inl}}\, V} \qquad \frac{M \Downarrow \widetilde{\mathbf{inl}}\, V_1 \qquad M_1[V_1/x] \Downarrow V_2}{\widetilde{\mathbf{case}}\, M\, \mathbf{of}\ \mathbf{inl}\, x \Rightarrow M_1\, ;\mathbf{inr}\, x \Rightarrow M_2 \Downarrow V_2}$$

$$\frac{M \Downarrow V}{\widetilde{\mathbf{inr}}\, M \Downarrow \widetilde{\mathbf{inr}}\, V} \qquad \frac{M \Downarrow \widetilde{\mathbf{inr}}\, V_1 \qquad M_2[V_1/x] \Downarrow V_2}{\widetilde{\mathbf{case}}\, M\, \mathbf{of}\ \mathbf{inl}\, x \Rightarrow M_1\, ;\mathbf{inr}\, x \Rightarrow M_2 \Downarrow V_2}$$

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

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

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

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

The syntax of SFL is powerful enough to express some more features that we have not discussed, yet. Of course, we can define a type $\mathbf{void} := \mu\alpha.\alpha$ not containing any value and a type $\mathbf{unit} := \mathbf{void}_\bot$. Moreover, we can define a fixed point combinator $\mathbf{Y}_\tau$ of type $(\tau \to \tau) \to \tau$ and an always diverging term $\Omega_\tau := \mathbf{Y}_\tau(\lambda x{:}\tau.x)$ for every SFL type τ. This enables us to define the type of natural numbers, lists etc. We use $* := \mathbf{up}\,\Omega_{\mathbf{void}}$ as an abbreviation for the (up to equivalence) only value of type **unit**.

Although the denotations $[\![\Sigma_{i=1}^n \tau_i]\!]$ and $[\![(\tau_1)_\bot \oplus \cdots \oplus (\tau_1)_\bot]\!]$ as well as $[\![\tau_1 \to \tau_2]\!]$ and $[\![(\tau_1)_\bot \circ\!\!\to \tau_2]\!]$ are isomorphic in *SD* (see 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, giving the same evaluation rules. For instance, if we set $\tau_1 \to \tau_2 := (\tau_1)_\bot \circ\!\!\to \tau_2$, $\lambda x{:}\tau.M := \widetilde{\lambda} y{:}\tau_\bot.M[\mathbf{down}\, x/x]$ and $MN := M\tilde{\ }(\mathbf{up}\, N)$, the term $(\lambda x{:}\tau.\mathbf{up}\, x)\Omega_\tau$ of type $\tau_\bot$ would evaluate to $\mathbf{up}\,(\mathbf{down}\,(\mathbf{up}\,\Omega_\tau))$ rather than to $\mathbf{up}\,\Omega_\tau$. The same phenomenon occurs for terms of a separated sum type defined by using coalesced sums types.

Defining the usual operational equivalence for SFL by convergence as for FPC is meaningless, since e.g. $\langle \Omega_{\mathbf{unit}}, \Omega_{\mathbf{unit}} \rangle$ and $\Omega_{\mathbf{unit}\times\mathbf{unit}}$ can be distin-

guished by the trivial context ($\langle \Omega_{\mathbf{unit}}, \Omega_{\mathbf{unit}} \rangle \Downarrow$, but $\Omega_{\mathbf{unit} \times \mathbf{unit}} \not\Downarrow$) (where $M \Downarrow$ indicates that there is a value V with $M \Downarrow V$) although both of them are denotated by $\bot_{\mathbf{2} \times \mathbf{2}}$, see next section. Therefore, we use an observational equivalence similar to PCF:

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 $C[\cdot]$. Note that any computational type would do in place of **unit**, giving an equivalent definition.

The termination of terms of a computational type is operationally observable: 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. Therefore, because values of a smash product type are pairs of values in both components, the request for a smash product type with non-computational components has to be rejected. The usage of a type as a component of a smash product type for which termination of its terms is not sequentially semidecidable 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 to define closed terms $\widetilde{\mathbf{case}}\,(\widetilde{\mathbf{inl}}\,M)\,\mathbf{of}\ \mathbf{inl}\,x \Rightarrow *\,; \mathbf{inr}\,x \Rightarrow *$ (for coalesced sums), $\widetilde{\mathbf{pr_1}}\,(*, M)$ (for smash products) and $(\widetilde{\lambda x{:}\tau.*})\tilde{\ }M$ (for strict functions) of type **unit** that terminate if, and only if, M of type τ does so as well. Since this is generally not sequentially observable, those terms cannot exist in a purely sequential language.

It is an interesting observation that the introduction of smash product types for non-SSB-types does not work for operational reasons, whereas the relations on smash products in the category *SD* cannot be defined directly from the relation of its components.

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 $\lambda x{:}\tau_1.\Omega_{\tau_2}$ are observationally equivalent in SFL which is not the case for call-by-value or lazy call-by-name languages. As we will see later, this fits with the denotational semantics of 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 $\lambda x{:}\tau_1.\Omega_{\tau_2}$ are equal. However, we can encode terms of other sequential languages in SFL, for example, call-by-value abstraction and application refer to $\mathbf{up}\,(\widetilde{\lambda x{:}\sigma.M})$ and $(\mathbf{down}\,M)\tilde{\ }N$ whereas the corresponding terms for a lazy call-by-name language are $\mathbf{up}\,(\lambda x{:}\tau.M)$ and $(\mathbf{down}\,M)N$. The denotational semantics of those functions is the lifting of the function spaces in *SD* (strict functions for call-by-value and normal ones for call-by-name style). The advantage of the more general approach in this paper is that the language SFL is close to the categorical meaning of products

and functions.

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 $x_1, \ldots, x_n$.

- $\Gamma \vdash \mathbf{pr_1}\,\langle M_1, M_2\rangle = M_1 : \tau$,
- $\Gamma \vdash \mathbf{pr_2}\,\langle M_1, M_2\rangle = M_2 : \tau$,
- if $M_2[N_1/x_1, \ldots, N_n/x_n]{\Downarrow}$ for all well-typed terms $N_1, \ldots, N_n$ then $\Gamma \vdash \widetilde{\mathbf{pr_1}}\,(M_1, M_2) = M_1 : \sigma$,
- if $M_1[N_1/x_1, \ldots, N_n/x_n]{\Downarrow}$ for all well-typed terms $N_1, \ldots, N_n$ then $\Gamma \vdash \widetilde{\mathbf{pr_2}}\,(M_1, M_2) = M_2 : \sigma$,
- $\Gamma \vdash \big(\mathbf{case}\,(\mathbf{in}_j\, M)\,\mathbf{of}\ \mathbf{in}_i\, x \Rightarrow M_1, \ldots, M_n\big) = M_j[M/x] : \tau$ for all $j = 1, \ldots, n$,
- if $M_1[N_1/x_1, \ldots, N_n/x_n]{\Downarrow}$ for all well-typed terms $N_1, \ldots, N_n$ then $\Gamma \vdash \big(\widetilde{\mathbf{case}}\,(\widetilde{\mathbf{inl}}\, M_1)\,\mathbf{of}\ \mathbf{inl}\, x \Rightarrow M_2\,;\mathbf{inr}\, x \Rightarrow M_3\big) = M_2[M_1/x] : \tau$,
- if $M_1[N_1/x_1, \ldots, N_n/x_n]{\Downarrow}$ for all well-typed terms $N_1, \ldots, N_n$ then $\Gamma \vdash \big(\widetilde{\mathbf{case}}\,(\widetilde{\mathbf{inr}}\, M_1)\,\mathbf{of}\ \mathbf{inl}\, x \Rightarrow M_2\,;\mathbf{inr}\, x \Rightarrow M_3\big) = M_3[M_1/x] : \tau$,
- $\Gamma \vdash \mathbf{down}\,(\mathbf{up}\, M) = M : \tau$,
- $\Gamma \vdash (\lambda x{:}\tau_1.M)N = M[N/x] : \tau_2$,
- if $x \notin FV(M)$ then $\Gamma \vdash \lambda x{:}\tau_1.Mx = M : \tau_1 \to \tau_2$,
- if $N[N_1/x_1, \ldots, N_n/x_n]{\Downarrow}$ for all well-typed terms $N_1, \ldots, N_n$ then $\Gamma \vdash (\widetilde{\lambda} x{:}\sigma.M)\,\tilde{}\,N = M[N/x] : \tau$,
- if $x \notin FV(M)$ then $\Gamma \vdash \widetilde{\lambda} x{:}\sigma.M\,\tilde{}\,x = M : \sigma \multimap \tau$,
- $\Gamma \vdash \mathbf{unfold}\,(\mathbf{fold}\, M) = M : \tau$,
- $\Gamma \vdash \mathbf{fold}\,(\mathbf{unfold}\, M) = M : \tau$,
- $\Gamma \vdash M = M : \tau$,
- 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 $\Gamma \vdash M_1 = N_1 : \tau_1, \ldots, \Gamma \vdash M_n = N_n : \tau_n$ then $\Gamma \vdash C[M_1, \ldots, M_n] = C[N_1, \ldots, N_n] : \tau$ (for every context $C[\cdot, \ldots, \cdot]$).

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 equation system is by no means complete (which is actually impossible due to Gödel's Incompleteness Theorem). However, as we will see in the next section, it is sound with respect to the operational and denotational semantics of SFL.

5 The interpretation of SFL in *SD*

The semantics for finite SFL types (i.e. types that do not contain type variables or recursive types except for **void**) is the one which is suggested by the choice

of symbols for the connectives, i.e. $[\![\tau_\bot]\!] = [\![\tau]\!]_\bot$, $[\![\sigma_1 \otimes \sigma_2]\!] = [\![\sigma_1]\!] \otimes [\![\sigma_2]\!]$, $[\![\sigma_1 \oplus \sigma_2]\!] = [\![\sigma_1]\!] \oplus [\![\sigma_2]\!]$, $[\![\tau_1 \times \tau_2]\!] = [\![\tau_1]\!] \times [\![\tau_2]\!]$ etc. Since the Kripke relations on type denotations are uniquely determined by the structure of its type, we do not distinguish between the denotation $[\![\tau]\!]$ of an SFL type τ and its underlying domain. We also write $[\![\tau]\!]_{\mathcal{C}}$ for the $\mathcal{C}$-Kripke relation corresponding to $[\![\tau]\!]$.

A simple recursive type $\mu\alpha.\tau$ is interpreted as $\mathsf{fix}(F_\tau)$ where $F_\tau\colon SD_\bot^{op} \times SD_\bot \to SD_\bot$ is the mixed variant functor on $SD_\bot$ that is determined by the structure of τ. The $\mathcal{C}$-Kripke relations on such a bilimits $\mathsf{fix}(F_\tau)$ are componentwise defined. In particular, the type **void** $= \mu\alpha.\alpha$ is interpreted as the terminal object $(\mathbf{1}, \mathbf{1})$. This construction works since all type constructing functors are locally continuous and the embeddings and projections are strict and uniform.

For nested recursive types like $\mu\alpha.\mu\beta.\tau$, we need to consider parameterised functors. Even if $\mu\alpha.\mu\beta.\tau$ is a closed type, τ may contain two free type variables α and β. Therefore, the corresponding functor F_τ has four arguments for negative and positive occurrences of each type. In the general case, τ might even contain more free type variables, so we cannot restrict ourselves to functors with a certain number of arguments. However, it turns out that not more than four parameters need to be considered at the same time, thus, we focus on functors with four arguments and keep in mind that there may be some more. Suppose $F_\tau\colon SD_\bot^{op} \times SD_\bot \times SD_\bot^{op} \times SD_\bot \to SD_\bot$ is a functor corresponding to an SFL type τ. As for the category of domains, there is a locally continuous functor $F_{\mu\beta.\tau}\colon SD_\bot^{op} \times SD_\bot \to SD_\bot$ such that

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

holds for each pair of objects D^-, D^+ in $SD_\bot$. (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.

Note that the denotation $[\![\sigma]\!]$ of a computational SFL type σ is always an SSB-object.

Terms in SFL are denotated as terms-in-contexts with respect to their operational behaviour. If M is a closed SFL term of type τ, we use $[\![M]\!]$ as an abbreviation for $[\![\varnothing \vdash M : \tau]\!](\bot_{\mathbf{1}})$. In particular, the denotation of a fixed point combinator $\mathbf{Y}_\tau$ computes the least fixed point of a given function $f\colon [\![\tau]\!] \to [\![\tau]\!]$ in SD and the term Ω_τ is interpreted as the bottom element $\bot_{[\![\tau]\!]}$.

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

Proposition 5.1 *Let M be a closed SFL term with $[\![M]\!] \neq \bot$. Then M evaluates to a value.*

Using the Substitution Lemma, which can be easily proved by induction, it is another straightforward induction to verify that $[\![M]\!] = [\![V]\!]$ is true if $M \Downarrow V$ holds. On the other hand, Proposition 5.1 implies that the denotation of a

closed term M of a computational type σ is different from $\bot_\sigma$ if M evaluates to a value. Hence, the following adequacy result follows:

Theorem 5.2 ***(Computational Adequacy)*** *Let M be a closed SFL term of type* **unit***. Then M terminates if, and only if, $[\![M]\!] = \top$ holds.*

For the full abstraction result we define one particular subcategory $\mathcal{C}$ of *SPIC*. It is essentially the same definition as in [14] and [18]. An object $[\tau_1, \ldots, \tau_n]$ of $\mathcal{C}$ consist of the product of the underlying sets of interpretations of finite SFL types $\tau_1, \ldots, \tau_n$ and a sequentiality system defined as

$$S_n = \{[\![\Gamma \vdash M : n \odot \mathbf{unit}]\!] \mid \Gamma \vdash M : n \odot \mathbf{unit} \text{ is an SFL term-in-context}\}$$

(where $\Gamma = x_1{:}\tau_1, \ldots, x_m{:}\tau_m$, $0 \odot \mathbf{unit} = \mathbf{void}$, $n \odot \mathbf{unit} = \overbrace{\mathbf{unit} \oplus \cdots \oplus \mathbf{unit}}^{n\text{-times}}$). It can be easily shown that this is indeed a sequentiality system on $[\tau_1, \ldots, \tau_n]$. The morphisms in $\mathcal{C}$ are projections $\pi\colon ([\tau_1, \ldots, \tau_{n+k}], S) \to ([\tau_1, \ldots, \tau_n], S')$. Again, it is not difficult to verify that this definition makes $\mathcal{C}$ a subcategory of *SPIC*.

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

Lemma 5.3 *Let $\mathcal{C}$ be the category from above and let $f \in [\![\tau]\!]$ be an element of the denotation of a finite SFL type. Then f is related by $[\![\tau]\!]^{\varnothing}_{\mathcal{C}}$ if, and only if, there is a closed SFL term F with $[\![F]\!] = f$.*

Proof. The idea is to show the following more general claim by induction of the structure of the type τ.

> Let $\mathsf{w} = (w, S)$ with $w = [\varrho_1, \ldots, \varrho_n]$ be an object of $\mathcal{C}$, $\Gamma = x_1{:}\varrho_1, \ldots, x_n{:}\varrho_n$ a context and $f\colon [\varrho_1, \ldots, \varrho_n] \to [\![\tau]\!]$ a function. Then $f \in [\![\tau]\!]^{\mathsf{w}}_{\mathcal{C}}$ holds if, and only if, there is an SFL term F with $[\![\Gamma \vdash F : \tau]\!] = f$.

This proves the lemma by choosing $w = \varnothing$. □ □

As a consequence of Lemma 5.3, it can be shown that every element in the denotation of a finite SFL type is definable. Besides, since projections and embeddings of a bilimit can be defined in SFL as well, this yields definabilitity of every compact element in the denotation of an arbitrary SFL type. This fact provides the most important tool to prove the main result of this section:

Theorem 5.4 ***(Full abstraction)*** *Let M, N be closed terms of the same type. Then:*

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

Proof. The "⇒"-part is a consequence of Theorem 5.2. For the "⇐"-direction we need to generalise the claim in order to deal with free type variables:

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

The proof for this is a straigthforward induction using the fact that every type denotation is an algebraic domain. Thus, $M \sqsubseteq_{SFL} N$ implies $[\![M]\!] \sqsubseteq [\![N]\!]$ for all closed terms M, N of a closed type τ. □ □

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

Corollary 5.5 ***(Soundness)*** *Let $\Gamma \vdash M = N : \tau$ be an SFL equation-in-context. Then $[\![\Gamma \vdash M : \tau]\!] = [\![\Gamma \vdash N : \tau]\!]$ holds.*

6 Discussion

The category *SD* presented in this paper provides an adequate model for reasoning about sequential computation in the sense that it provides sufficient relational criteria characterising sequential continuous functions. Admittedly, the bunch of relations associated with the domains arising as interpretations of SFL types is fairly large (cardinality of the continuum). That means that it is not possible to verify whether a given continuous function between two objects in *SD* is an *SD*-morphism. In particular, the exponentials in *SD* are not decidable. However, in the light of Loader's undecidability result for finite PCF [7], it seems to be unlikely that a substantial improvement can be achieved as it entails that the additional relational structure has to be infinite.

Recently, the fully abstract model for PCF in the category *SD* was reconstructed as a realisability model over the partial combinatory algebra A which is obtained as the canonical solution of the domain equation $A \cong N \oplus [A \to A]_\perp$ in *SD* (see [12]). In particular, for every PCF type σ its interpretation in $Mod(A)$ is isomorphic to the one given by the canonical retraction of A to its denotation in *SD*. This result can be refined to a solution of (a variant of) the Longley-Phoa conjecture in the following way. Let $\mathcal{L}$ be the language associated to the domain equation for A, i.e. untyped λ–calculus with arithmetic, and let L be the sub-pca of A consisting of those elements of A that can be denoted by (closed) terms of $\mathcal{L}$. It turns out that the PCF–model in $Mod(L)$ is obtained as a restriction of the one in $Mod(A)$ by restricting realisers to L and the elements of the underlying set to those which are realised by elements of L. Notice that L is a term model as it is isomorphic to $\mathcal{L}_{Th(A)}$. One can show that the choice of $Th(A)$ is irrelevant, i.e. instead of L one might take $\mathcal{L}/_T$ for any theory T contained in $Th(A)$ and containing the obvious conversion rules for $\mathcal{L}$ and it still holds that the PCF–model in $Mod(\mathcal{L}/_T)$ is isomorphic to the one in $Mod(L)$.

Among the various open questions to be investigated in the future we just focus on the following ones. It is desirable to find a fully abstract model for a polymorphic extension of SFL. In particular this would give rise to a fully abstract model for the polymorphic lambda calculus over a base type allowing recursive definitions of objects. Furthermore, such a polymorphic extension of SFL seems to be needed for obtaining fully abstract models of ALGOL-like

languages in particular with respect to their concept of local variables.

Another open problem is the integration of the relational and the game-theoretic account of full abstraction. It might be interesting to give a direct proof that the relational model of SFL is the extensional collapse of its game model (in analogy to [3]).

Acknowledgements

I would like to thank the members of the *Arbeitsgruppe 14* of the *Fachbereich Mathematik* at the *Technische Universität Darmstadt* and the *'Theory Group'* of the *School of Computer Science* at the *University of Birmingham* for the stimulating atmosphere in their departments and for their interest in this work. Moreover, I thank Martín Escardó, Marcello Fiore, Martin Hofmann, Mathias Kegelmann, Peter O'Hearn, Gordon Plotkin and Dirk Thierbach for helpful discussions. Very special thanks go to my supervisors Achim Jung (in Birmingham) and Thomas Streicher (in Darmstadt) for many long discussions with plenty of interesting ideas and useful hints.

Besides, I thank Dieter Spreen for organising the nice workshop in Remagen-Rolandseck and for giving me the opportunity to present my work. After discussing the topic with other participants I was able to simplify some of the results.

References

[1] Abramsky, S. 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, Clarendon Press, 1994, pages 1–168.

[2] 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, Springer Verlag, 1994, pages 1–15.

[3] Ehrhard, T., *A relative PCF-definability result for strongly stable functions and some corollaries,* February, 1997 Available from http://hypatia.dcs.qmw.ac.uk/cgi-bin/sarah?q=ehrhard.

[4] Hyland, J. M. E. and C.-H. L. Ong, *On full abstraction for PCF,* 1995, Manuscript (available from `ftp://ftp.comlab.ox.ac.uk/pub/Documents/techpapers/Luke.Ong/pcf.ps.gz`).

[5] Jung, A., 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, June 1996, pp. 227–256.

[6] Jung, A. and J. Tiuryn, *A new characterization of lambda definability,* in: M. Bezem and J. F. Groote, editors, *Typed Lambda Calculi and Applications*, Lecture Notes in Computer Science **664** (1993), pp. 245–257.

[7] Loader, R., *Finitary PCF is not decidable,* July 1996, Unpublished manuscript available from `http://hypatia.dcs.qmw.ac.uk`.

[8] Marz, M., *A fully abstract model for sequential computation,* Technical Report CSR-98-6, University of Birmingham, September 1998, Available from `ftp://ftp.cs.bham.ac.uk/pub/tech-reports/1998/CSR-98-06.ps.gz`.

[9] Meyer, A. R. and S. S. Cosmadakis, *Semantical paradigms: Notes for an invited lecture,* in: *3rd Symposium on Logic in Computer Science*, IEEE Computer Society Press, 1988, pp. 236–253.

[10] McCusker, G., *Games and full abstraction for FPC,* in: *Eleventh Annual IEEE Symposium on Logic in Computer Science*, 1996, pages 174–183.

[11] Milner, R., *Fully abstract models of typed lambda-calculi,* Theoretical Computer Science **4** (1977), pp. 1–22.

[12] Marz, M., A. Rohr and T. Streicher, *Full abstraction via realisability,* Submitted to LICS 99, 1998. Available from `http://www.mathematik.tu-darm- stadt.de/~streicher/FAR/far.ps.gz`.

[13] Nickau, H., *Hereditarily sequential functionals,* in: *Proceedings,* Symposium on Logical Foundations of Computer Science: Logic at St. Petersburg, Lecture Notes in Computer Science, Springer Verlag, 1994.

[14] O'Hearn, P. W. and J. G. Riecke, *Kripke logical relations and PCF,* Information and Computation **120** (1995), pp. 107–116.

[15] O'Hearn, P. W. and J. C. Reynolds, *From Algol to polymorphic linear lambda calculus,* Lectures at Isaac Newton Institute for Mathematical Sciences, Cambridge, 1995.

[16] Plotkin, G. D., *LCF considered as a programming language,* Theoretical Computer Science **5** (1977), pp. 223–255.

[17] Plotkin, G. D., "Post-graduate Lecture Notes in Advanced Domain Theory (Incorporating the 'Pisa Notes'," Dept. of Computer Science, Univ. of Edinburgh, 1981.

[18] Riecke, J. and A. Sandholm, *A relational account of call-by-value sequentiality,* in: Twelfth Annual IEEE Symposium on Logic in Computer Science, LICS'97, 1997, pp. 258–267.

[19] Scott, D. S., *A type-theoretical alternative to ISWIM, CUCH, OWHY,* Theoretical Computer Science **121** (1993), pp. 411–440. Reprint of a manuscript written in 1969.

[20] Sieber, K., *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, LMS Lecture Note Series **177** (1992), Cambridge University Press, pp. 258–269.