Full Abstraction and Universality via Realisability

Michael Marz*
marz@mathematik.tu-darmstadt.de

Alexander Rohr†
rohr@mathematik.tu-darmstadt.de

Thomas Streicher
streicher@mathematik.tu-darmstadt.de

Technische Universität Darmstadt
Fachbereich Mathematik
Schloßgartenstraße 7, D-64289 Darmstadt

Abstract

We construct fully abstract realisability models of PCF. In particular, we prove a variant of the Longley–Phoa Conjecture by showing that the realisability model over an untyped λ–calculus with arithmetic is fully abstract for PCF.

Further we consider the extension of our results to a general sequential functional programming language SFPL giving rise to universal *realisability models for SFPL.*

1. Introduction

Realisability models are known as a flexible tool for organising untyped models of computation into extensional models with a rich type structure where domains appear as sets, see e. g. [5, 8]. For example the Scott model and its effective variant both appear as realisability models over $\mathcal{P}\omega$ and $\mathbb{N}$ with Kleene application, respectively. The model of strongly stable domains has just recently been identified as a realisability model over the partial combinatory algebra (pca) $\mathcal{B}$ of (effective) partial functions on $\mathbb{N}$, independently by J. van Oosten [18] and J. Longley [7].

In this paper we will exhibit the fully abstract Milner model of PCF as a realisability model over the pca A as given by the recursive domain equation $A \cong N \oplus [A{\to}A]_\perp$ in the category *SD* of sequential domains introduced in [9]. In order to overcome the fairly abstract nature of A we restrict A to the sub–pca L on those elements of A appearing as denotations of closed $\mathcal{L}$ terms where $\mathcal{L}$ is an untyped λ–calculus with arithmetic and a unary operation $\mathbf{where}$ telling whether its argument is a number or a function. We show that the realisability model of PCF over L is fully abstract, too. This result extends to all pca's of the form $\mathcal{L}_{/T}$ where T is a congruence on (the closed terms of) $\mathcal{L}$ validated by A and containing the conversion equality $\sim$ of $\mathcal{L}$. We consider this as a proof of a variant of the so called *Longley–Phoa Conjecture* in its original form claiming that realisability models over (suitable) term models of the untyped λ–calculus are fully abstract for PCF. Our variant consists in replacing pure λ–calculus by the somewhat more realistic language $\mathcal{L}$ where natural numbers are built in as basic data instead of coding them by λ–terms.[1] Though we have not been able to prove *universality* of the realisability model over $\mathcal{L}_{/\sim}$, i. e. that all elements of PCF types appear as denotations of closed PCF terms, it is easy to see that all elements of PCF types appear as denotations of closed SFPL terms where SFPL is a general sequential functional programming language of the same expressivity as Marz's SFL introduced in [9]. Finally, we show how to extend our work to an analogous treatment of SFPL giving rise to a universal (and accordingly fully abstract) realisability model of SFPL employing the recursive type $U = U \oplus U \oplus (U \otimes U) \oplus [U {\circ\!\!\to} U]_\perp$ instead of A. A comprehensive presentation of this extension will be found in the PhD Thesis of A. Rohr.

We see our work in the context of recent interest in realisability models for sequential computation [5, 7, 6, 10, 17]. From our point of view realisability models provide a link between purely denotational approaches via 'relational full abstraction' [11, 14, 9] culminating in the category *SD* of sequential domains and operational semantics as given by the untyped programming language $\mathcal{L}$. In [6, 10] one can find related approaches which are based on game semantics, however, we find languages like $\mathcal{L}$ easier to understand. Be-

*Supported by the Daimler-Benz-Stiftung.

†Supported by the Bundesland Hessen.

[1] For an approach to the original Longley–Phoa Conjecture via realisability over a game model of λ–calculus see [10] which, however, fails as the game model under consideration does not contain the natural numbers with $\perp$ as a retract.

sides, they are closer to actual functional programming anyway. Though this preference may be purely subjective (and conservative) our choice of $\mathcal{L}_{/\sim}$ instead of a pca consisting of strategies of a 'universal game' allows for simple and concise proofs based on well-known techniques from categorical logic and realisability semantics avoiding the sometimes heavy machinery of game semantics. Of course, this way our realisability models have a less 'language independent' flavour than the approaches based on game semantics. Nevertheless, the model in *SD* is as extensional and language independent as possible and realisability by $\mathcal{L}$ terms provides a notion of sequential computability for *SD*. It appears to be an interesting problem for further research to find some game semantics for $\mathcal{L}$ where all effective strategies appear as denotations of terms.

Having identified fully abstract or even universal models as full subcategories of realisability models has the further advantage that they fall under the class of models studied in [8]. Therefore, an aim worthwhile for future investigation is to develop an axiomatic treatment of fully abstract or even universal models within Synthetic Domain Theory.

2. The Category *SD* of Sequential Domains

In this section we describe a category *SD* of sequential domains providing a fully abstract model for *sequential functional programming* including the usual type formers and recursively defined types. It subsumes and clarifies previous work on relational full abstraction by O'Hearn, Riecke and Sandholm [11, 14]. A detailed account including proofs can be found in [9].

First we define a notion of a *system of sequential partitions* by isolating a set of closure conditions of those partial partitions of a context Γ that are induced by (denotations of) terms $\Gamma \vdash t : \underline{n}$ (where $\underline{n}$ is the data type consisting of n atoms together with $\perp$).

DEFINITION 1. *A* system of sequential partitions (ssp) *is a pair* $\mathsf{w} = (w, S)$ *where* w *is a finite set (of natural numbers) and* S *is a set of partial partitions of* w*, i. e. a set of partitions of subsets of* w*, satisfying the following closure conditions*

(P1) $\{w\} \in S$ *if* w *is non-empty,*

(P2) *if* $u \in P \in S$ *then* $P - \{u\} \in S$ (Dropping),

(P3) *if* $u, v \in P \in S$ *then* $(P - \{u, v\}) \cup \{u \cup v\} \in S$, (Joining)

(P4) *if* $P, Q \in S$ *and* $u \in P$ *then* $(P - \{u\}) \cup (\{ u \cap v \mid v \in Q\} - \{\varnothing\}) \in S$. (Refinement)

If $\mathsf{w}_1 = (w_1, S_1)$, $\mathsf{w}_2 = (w_2, S_2)$ *are ssp's then an* ssp morphism from w_1 to w_2 *is a function* $\phi \colon w_1 \to w_2$ *satisfying*

$$\{ \phi^{-1}[u] \mid u \in P \} - \{\varnothing\} \in S_1$$

for every $P \in S_2$*. Obviously, ssp morphisms are closed under composition. We write SSP for the ensuing category of ssp's and ssp morphisms.* $\diamond$

Notice that the conditions (P1)–(P4) correspond to the following closure properties of functions denoted by terms of the form $\Gamma \vdash t : \underline{n}$ for some $n \in \mathbb{N}$.

1. We have all constant functions from Γ to some $\underline{n}$.
2. If $\Gamma \vdash t : \underline{n}$ then $\Gamma \vdash$ **if** $t = i$ **then** $\perp$ **else** $t : \underline{n}$.
3. If $\Gamma \vdash t : \underline{n}$ then $\Gamma \vdash$ **if** $t = j$ **then** i **else** $t : \underline{n}$.
4. Definition by case analysis can be nested.

Next we fix a notion of w-ary logical relation on cpo's, i. e. complete partial orders with $\perp$, isolating the closure properties of denotations of terms of a fixed type in a fixed context.

DEFINITION 2. *Let* A *be a cpo and* $\mathsf{w} = (w, S)$ *be an ssp. For* $f \colon w \to A$ *and* $\{u\} \in S$ *let* $f{\restriction}u$ *be the function from* w *to* A *with* $f{\restriction}u(i) = f(i)$ *for* $i \in u$ *and* $f{\restriction}u(i) = \perp$ *otherwise.*

A w*-ary logical relation on* A *is a subset* R *of* A^w *satisfying the following closure conditions*[2]

(R1) $\lambda i{:}w.\, a \in R$ *for all* $a \in A$ (Constant),

(R2) *if* $P \in S$ *and* $a \colon P \to R$ *then* $\mathsf{case}(P, a) := \bigsqcup_{u \in P} a_u{\restriction}u \in R$, (Case)

(R3) R *is closed under directed suprema* (Admissibility). $\diamond$

The second condition expresses closure under *case analysis* as $\mathsf{case}(P, a)(i) = a_u(i)$ whenever $i \in u \in P$ and $\mathsf{case}(P, a)(i) = \perp$ if $i \notin \bigcup P$.

Notice that w-ary logical relations on a cpo A are closed under arbitrary intersections due to the inductive nature of the closure conditions of Definition 2. Accordingly, for any subset $R \subseteq A^w$, there exists a *least* w-ary logical relation on A *containing* R.

Next we define so called 'Kripke logical relations', i. e. logical relations of 'varying arity', similar to the ones originally introduced by A. Jung and J. Tiuryn in [3].

DEFINITION 3. *An* arity $\mathcal{A}$ *is an arbitrary subcategory of SSP. A* Kripke logical relation of arity $\mathcal{A}$ on a cpo A *is a family* $R = (R^{\mathsf{w}})_{\mathsf{w} \in \mathcal{A}}$ *assigning a* w*-ary logical relation* R^{w} *on* A *to every* $\mathsf{w} \in \mathcal{A}$ *and which is stable under re-indexing in the sense that, for all morphisms* $\phi \colon \mathsf{w}_2 \to \mathsf{w}_1$ *in* $\mathcal{A}$, $a \circ \phi \in R^{\mathsf{w}_2}$ *whenever* $a \in R^{\mathsf{w}_1}$. $\diamond$

Now we are ready to define the category *SD* of Sequential Domains.

[2] equivalent to the slightly different ones of [9]

DEFINITION 4. *A* sequential domain *is defined to be a pair $A = \big(|A|, R(A)\big)$ where $|A|$ is a cpo and $R(A)$ is a family of Kripke logical relations on $|A|$ associating with every arity $\mathcal{A}$ a Kripke logical relation $R(A)_{\mathcal{A}}$ on $|A|$ of arity $\mathcal{A}$.*

For sequential domains A and B an SD-morphism from A to B is a continuous function $f\colon |A| \to |B|$ such that, for all arities $\mathcal{A}$ and $\mathsf{w} \in \mathcal{A}$, $f \circ a \in R(B)^{\mathsf{w}}_{\mathcal{A}}$ for all $a \in R(A)^{\mathsf{w}}_{\mathcal{A}}$. Composition in SD is given by ordinary composition of functions. $\diamond$

Notice that for all $A, B \in \mathit{SD}$ the set $\mathit{SD}(A, B)$ is a cpo when ordered pointwise. This way the category *SD* is cpo-enriched and order extensional, i. e. in particular well-pointed. Moreover, *SD* enjoys all good properties required for a category of domains as can be found in [13]. We are going describe some of these constructions without verifying that they meet their characterising universal properties, see [9] for details of this kind.

Cartesian Products. For $A, B \in \mathit{SD}$ their cartesian product $A \times B$ is constructed as follows: $|A \times B| = |A| \times |B|$ and $a \in R(A \times B)^{\mathsf{w}}_{\mathcal{A}}$ if, and only if, $\pi_1 \circ a \in R(A)^{\mathsf{w}}_{\mathcal{A}}$ and $\pi_2 \circ a \in R(B)^{\mathsf{w}}_{\mathcal{A}}$.

Exponentials. For $A, B \in \mathit{SD}$ the function space $[A \to B]$ is constructed as follows: $\big|[A \to B]\big| = \mathit{SD}(A, B)$ ordered pointwise and, for all arities $\mathcal{A}$,

$$f \in R([A \to B])^{\mathsf{w}}_{\mathcal{A}} \quad \text{iff}$$
$$\forall \phi\colon \mathsf{w}' \overset{\mathcal{A}}{\to} \mathsf{w}. \forall a \in R(A)^{\mathsf{w}'}_{\mathcal{A}}.\ \mathsf{eval} \circ \langle f \circ \phi, a\rangle \in R(B)^{\mathsf{w}'}_{\mathcal{A}}$$

where eval denotes the evaluation function defined by $\mathsf{eval}(g, y) = g(y)$. As usual, if $h\colon C \times A \to B$ is a morphism in *SD*, then $\mathsf{curry}(h)\colon C \to [A \to B]$ is given by $\mathsf{curry}(h)(z)(x) = h(z, x)$.

Terminal Object. The object $\mathbf{1}$ is given by $|\,\mathbf{1}\,| = \{\bot\}$ and $R(\mathbf{1})^{\mathsf{w}}_{\mathcal{A}} = \{\bot\}^{w}$ where $\mathsf{w} = (w, S) \in \mathcal{A}$.

Lifting. For a sequential domain A its lifting $A_\bot$ is given by $|A_\bot| = |A|_\bot$ and, for all arities $\mathcal{A}$ and all ssp's $\mathsf{w} = (w, S) \in \mathcal{A}$,

$$R(A_\bot)^{\mathsf{w}}_{\mathcal{A}} := \big\{ (\mathsf{up} \circ a){\restriction}u \ \mid\ a \in R(A)^{\mathsf{w}}_{\mathcal{A}} \wedge \{u\} \in S \big\}$$

where $\mathsf{up}\colon |A| \hookrightarrow |A|_\bot$ is the obvious inclusion.

As usual one may consider the sub-category $\mathit{SD}_\bot$ of *SD* obtained by restricting to strict morphism (i. e. those preserving $\bot$) and keeping all objects.

Strict Function Space. Given sequential domains A and B their strict function space $[A \circ\!\!\to B]$ is obtained from the ordinary function space $[A \to B]$ as follows: $\big|[A \circ\!\!\to B]\big| = \mathit{SD}_\bot(A, B)$, which is a sub-cpo of $\mathit{SD}(A, B)$, and $f \in R([A \circ\!\!\to B])^{\mathsf{w}}_{\mathcal{A}}$ if, and only if, $f \in R([A \to B])^{\mathsf{w}}_{\mathcal{A}}$ and the image of f is contained in $\mathit{SD}_\bot(A, B)$.

Smash Product. For sequential domains A and B their smash product $A \otimes B$ is defined as follows: $|A \otimes B| = |A| \otimes |B|$ and, for arities $\mathcal{A}$ and $\mathsf{w} \in \mathcal{A}$, the w-ary logical relation $R(A \otimes B)^{\mathsf{w}}_{\mathcal{A}}$ is defined as the least such containing the set $\{a \otimes b \mid a \in R(A)^{\mathsf{w}}_{\mathcal{A}} \wedge b \in R(B)^{\mathsf{w}}_{\mathcal{A}}\}$ where $a \otimes b = \otimes \circ \langle a, b\rangle$ and $\otimes\colon |A| \times |B| \to |A| \otimes |B|$ sends pairs whose both components are different from $\bot$ to themselves and all other pairs to $(\bot, \bot)$. Notice that $\otimes$ and $\circ\!\!\to$ endow the sub-category $\mathit{SD}_\bot$ with a symmetric monoidal closed structure.

Coalesced Sum. For sequential domains A and B their coalesced sum $A \oplus B$ is given as follows: $|A \oplus B| = |A| \oplus |B|$ and, for arities $\mathcal{A}$ and $\mathsf{w} \in \mathcal{A}$, the w-ary logical relation $R(A \oplus B)^{\mathsf{w}}_{\mathcal{A}}$ is the least such containing all elements of the form $\mathsf{inl} \circ a$ or $\mathsf{inr} \circ b$ for $a \in R(A)^{\mathsf{w}}_{\mathcal{A}}$ and $b \in R(B)^{\mathsf{w}}_{\mathcal{A}}$, respectively, where $\mathsf{inl}\colon A \hookrightarrow A \oplus B$ and $\mathsf{inr}\colon B \hookrightarrow A \oplus B$ are the obvious inclusions into the left and right summand, respectively.

Recursive Domain Equations. These are solved as usual by the inverse limit construction. If $\langle e_n\colon A_n \to A_{n+1}, p_n\colon A_{n+1} \to A_n\rangle$ is a sequence of embedding/projection pairs in *SD*, then its bilimit A_∞ is obtained as follows. The cpo $|A_\infty|$ is defined to be the set $\big\{a \in \prod_{n \in \mathbb{N}} |A_n| \ \big|\ \forall n{\in}\mathbb{N}.\, a_n = p_n(a_{n+1})\big\}$ endowed with the pointwise ordering. For arities $\mathcal{A}$ and ssp's $\mathsf{w} = (w, S) \in \mathcal{A}$ the w-ary logical relation $R(A_\infty)^{\mathsf{w}}_{\mathcal{A}}$ consists of all $a \in |A_\infty|^{w}$ with $\pi_n \circ a \in R(A_n)^{\mathsf{w}}_{\mathcal{A}}$ for all $n \in \mathbb{N}$.

Natural Numbers. The data type $N \cong \mathbf{1}_\bot \oplus N$ of natural numbers is a particular instance of a recursive type. Up to isomorphism, however, it can be described more explicitly as follows. The underlying cpo $|N|$ is the flat cpo $\mathbb{N} \cup \{\bot\}$ where all elements in $\mathbb{N}$ are maximal and $\bot$ is the least element of N. For arities $\mathcal{A}$ and ssp's $\mathsf{w} = (w, S) \in \mathcal{A}$ the w-ary logical relation $R(N)^{\mathsf{w}}_{\mathcal{A}}$ consists of all $a \in N^{w}$ such that the induced partial partition of w is contained in S, more precisely, $\{a^{-1}[\{n\}] \mid n \in \mathbb{N}\} - \{\varnothing\} \in S$.

The main result of [9] is that the category SD provides a fully abstract model for the sequential functional language SFL. This language subsumes languages as PCF and FPC for which SD is fully abstract as well. An analogous result holds for the later introduced programming language SFPL of equal expressivity. In particular it has been shown that *all* elements of (SD-interpretations of) non-recursive types are definable within SFL. Similarly for SFPL as it is of the same expressivity as SFL. Thus, all compact elements of recursive types are definable within SFL as all recursive types appear as inverse limits of non-recursive types.

For subsequent use we observe that the category *SD* has splitting of idempotents, i. e., for every $r \in \mathit{SD}(A,A)$ with $r \circ r = r$, there are *SD* morphisms $e\colon r(A) \to A$ and $p\colon A \to r(A)$ with $p \circ e = \mathsf{id}_{r(A)}$ and $r = e \circ p$. The sequential domain $r(A)$ is constructed as follows: its underlying cpo is the sub-cpo of $|A|$ on the image of r and $a \in R\big(r(A)\big)^{\mathsf{w}}_{\mathcal{A}}$ iff $a \in R(A)^{\mathsf{w}}_{\mathcal{A}}$ and $a = r{\circ}a$. The morphism e is given by inclusion whereas p is defined by $p(a) = r(a)$.

3. A Fully Abstract Realisability Model of PCF

The aim of this section is to demonstrate that the fully abstract model of PCF in *SD* is isomorphic to the realisability model[3] of PCF over the partial combinatory algebra A as given by the canonical solution of the domain equation

$$A \cong N \oplus [A{\to}A]_\perp$$

in *SD*. Obviously, both N and $[A{\to}A]$ appear as retracts of A via *SD* morphisms $e_{\mathrm{num}}\colon N \hookrightarrow A$ and $p_{\mathrm{num}}\colon A \to N$ with $p_{\mathrm{num}} \circ e_{\mathrm{num}} = \mathsf{id}_N$ and $e_{\mathrm{fun}}\colon [A{\to}A] \hookrightarrow A$ and $p_{\mathrm{fun}}\colon A \to [A{\to}A]$ with $p_{\mathrm{fun}} \circ e_{\mathrm{fun}} = \mathsf{id}_{[A\to A]}$, respectively. Thus, A is a *reflexive* object in the cartesian closed category *SD* and, therefore, a model of untyped λ-calculus as known from [15]. More explicitly, the binary function $ab := p_{\mathrm{fun}}(a)(b)$ endows A with the structure of a total combinatory algebra. Notice also that $A \times A$ is a retract of A as e.g. for the *SD* morphism $e_{\mathrm{prod}}\colon A \times A \to A$ with $e_{\mathrm{prod}}(a,b) = e_{\mathrm{fun}}(\lambda x{:}A.\,\mathsf{if}\ x{=}0\ \mathsf{then}\ a\ \mathsf{else}\ b)$ one can find an *SD* morphism p_{prod} with $p_{\mathrm{prod}} \circ e_{\mathrm{prod}} = \mathsf{id}_{A\times A}$.

In the sequel, $\mathsf{Mod}(A)$ denotes the category of *modest sets* over the combinatory algebra A. Objects of $\mathsf{Mod}(A)$ are pairs $X = (|X|, \Vdash_X)$ where $|X|$ is a set and $\Vdash_X \subseteq A \times |X|$ is the graph of a partial surjection from A to $|X|$. Usually, we simply write X for $|X|$ and $\Vdash$ for $\Vdash_X$ whenever X is clear from the context. For $x \in X$ we often use the notation $\|x\|$ to denote the set $\{a \in A \mid a \Vdash_X x\}$ of realisers of x, which is non-empty as $\Vdash_X$ is assumed to be surjective. Notice that, for different elements x and $y \in X$, the sets of realisers $\|x\|$ and $\|y\|$ are disjoint due to the assumption that $\Vdash_X$ is a functional relation from X to A.

For modest sets X and Y an element $a \in A$ is said to *realise* a function $f\colon |X| \to |Y|$ iff $ab \Vdash_Y f(x)$ whenever $b \Vdash_X x$, in which case we write $a \Vdash f$. A morphism $f\colon X \to Y$ in $\mathsf{Mod}(A)$ is, by definition, a function $f\colon |X| \to |Y|$ with $a \Vdash f$ for some $a \in A$. Composition of morphisms in $\mathsf{Mod}(A)$ is given by ordinary composition of functions. Obviously, $\mathsf{Mod}(A)(X,Y)$ becomes a modest set when endowed with the relation $\Vdash$ just defined. We often write $a \Vdash_{[X\to Y]} f$ for $a \Vdash f$ to make explicit the dependency on X and Y, besides, $[X{\to}Y]$ stands for the pair $(\mathsf{Mod}(A)(X,Y), \Vdash_{[X\to Y]})$ which turns out to be the exponential of Y by X in the cartesian closed category $\mathsf{Mod}(A)$.

For the particular combinatory algebra A (and more generally for all reflexive objects A of a well-pointed ccc) there is a remarkable full sub-ccc of $\mathsf{Mod}(A)$ of so called *A-retracts* encompassing all interpretations of PCF types in $\mathsf{Mod}(A)$.

DEFINITION 5. *For every retraction r of A let $r[A]$ be the modest set with underlying set $\big|r[A]\big| = \{r(a) \mid a \in A\}$ and $a \Vdash_{r[A]} x$ iff $r(a) = x$. A modest set X is called* A-retract *if, and only if, $X \cong r[A]$ for some idempotent $r \in \mathit{SD}(A,A)$.* ◇

As $\mathbf{1}$ and $A \times A$ are retracts of A it follows that A-retracts are closed under finite products. The next lemma says that A-retracts are also closed under exponentiation. Thus, A-retracts form a sub-ccc of $\mathsf{Mod}(A)$.

LEMMA 6. *Let $r_X, r_Y \in \mathit{SD}(A,A)$ be idempotent and e_X, p_X and e_Y, p_Y be splittings of r_X and r_Y, respectively. Let $e_{X\to Y}\colon [X{\to}Y] \to A$ and $p_{X\to Y}\colon A \to [X{\to}Y]$ be defined as $e_{X\to Y} = e_{\mathrm{fun}} \circ [p_X{\to}e_Y]$ and $p_{X\to Y} = [e_X{\to}p_Y] \circ p_{\mathrm{fun}}$ in SD.*

Then the exponential $\big[r_X[A] \to r_Y[A]\big]$ in $\mathsf{Mod}(A)$ is canonically isomorphic to the A-retract E with $|E| = [X{\to}Y]$ and $a \Vdash_E f$ iff $p_{X\to Y}(a) = f$. Thus, A-retracts are closed under exponentials.

Proof. W.l.o.g. assume that e_X and e_Y are inclusions. Then $X = \big|r_X[A]\big|$ and $a \Vdash_{r_X[A]} f$ iff $p_X(a) = f$ and similarly for Y.

Recall that the underlying set of the $\mathsf{Mod}(A)$-exponential $\big[r_X[A] \to r_Y[A]\big]$ consists of all set-theoretic functions $f\colon X \to Y$ which are realisable in the sense that, for some $a \in A$, we have $p_Y \circ p_{\mathrm{fun}}(a) = f \circ p_X$ for which we write $a \Vdash f$ as usual.

The canonical isomorphism between $\big[r_X[A] \to r_Y[A]\big]$ and E is constructed as follows. Whenever $a \Vdash f$ then $f \in [X{\to}Y]$ and $a \Vdash_E f$ because of

$$\begin{aligned} f = f \circ p_X \circ e_X &= p_Y \circ p_{\mathrm{fun}}(a) \circ e_X \\ &= p_{X\to Y}(a) \in \mathit{SD}(X,Y) \quad . \end{aligned}$$

On the other hand, if $a \Vdash_E f$, i. e. $f = p_{X\to Y}(a)$, then $e_{\mathrm{fun}}\big(p_{\mathrm{fun}}(a) \circ r_X\big) \Vdash f$ since we have $p_Y \circ p_{\mathrm{fun}}(a) \circ r_X = p_Y \circ p_{\mathrm{fun}}(a) \circ e_X \circ p_X = f \circ p_X$.

Finally, observe that $\big[r_X[A] \to r_Y[A]\big]$ is an A-retract because $e_{X\to Y} \circ p_{X\to Y} = e_{\mathrm{fun}} \circ [r_X{\to}r_Y] \circ p_{\mathrm{fun}}$ is an idempotent in $\mathit{SD}(A,A)$. ☐

Now we will show that the model of PCF in $\mathsf{Mod}(A)$ where base type ι is interpreted as $r_{\mathrm{num}}[A]$ is isomorphic to

[3] For more detailed information on realisability models see e.g. [16, 5].

the model of PCF in *SD* where ι is interpreted as N. Interpretations in $\mathsf{Mod}(A)$ and *SD* are referred to by $[\![\_]\!]_{\mathsf{Mod}(A)}$ and $[\![\_]\!]_{SD}$, respectively. Nevertheless, we often write simply σ for $[\![\sigma]\!]_{\mathsf{Mod}(A)}$. Every such σ appears in a canonical way as a retract of A via an embedding $e_\sigma : \sigma \to A$ and a projection $p_\sigma : A \to \sigma$. These are inductively defined as follows: for base type ι we have $e_\iota = e_{\mathrm{num}}$ and $p_\iota = p_{\mathrm{num}}$ and for function types we have $e_{\sigma\to\tau} = e_{\mathrm{fun}} \circ [p_\sigma \to e_\tau]$ and $p_{\sigma\to\tau} = [e_\sigma \to p_\tau] \circ p_{\mathrm{fun}}$.

THEOREM 7. *For an appropriate choice of exponentials we have* $\left|[\![\sigma]\!]_{\mathsf{Mod}(A)}\right| = [\![\sigma]\!]_{SD}$ *and* $a \Vdash_{[\![\sigma]\!]_{\mathsf{Mod}(A)}} d$ *iff* $p_\sigma(a) = d$ *for every PCF type* σ.

Thus, the model of PCF in $\mathsf{Mod}(A)$ *is isomorphic to the model of PCF in SD and, therefore, the model of PCF in* $\mathsf{Mod}(A)$ *is fully abstract.*

Proof. The proof is by induction on the structure of PCF types. For base type ι the claim is valid by definition. If $\left|[\![\sigma_i]\!]_{\mathsf{Mod}(A)}\right| = [\![\sigma_i]\!]_{SD}$ and $a \Vdash_{[\![\sigma_i]\!]_{\mathsf{Mod}(A)}} d$ iff $p_{\sigma_i}(a) = d$ for $i = 1, 2$, then due to Lemma 6 we can choose the exponential $[\![\sigma_1 \to \sigma_2]\!]_{\mathsf{Mod}(A)} = \left[[\![\sigma_1]\!]_{\mathsf{Mod}(A)} \to [\![\sigma_2]\!]_{\mathsf{Mod}(A)}\right]$ such that $\left|\left[[\![\sigma_1]\!]_{\mathsf{Mod}(A)} \to [\![\sigma_2]\!]_{\mathsf{Mod}(A)}\right]\right| = \left[[\![\sigma_1]\!]_{SD} \to [\![\sigma_2]\!]_{SD}\right]$ and $a \Vdash_{[\![\sigma_1\to\sigma_2]\!]_{\mathsf{Mod}(A)}} d$ iff $p_{\sigma_1\to\sigma_2}(a) = d$. $\square$

Notice that the results of this section can be obtained from weaker assumptions as well. In particular, one need not assume that $A \cong N \oplus [A\to A]_\perp$. It suffices that both N and $[A\to A]$ are *SD*–retracts of A.

Our key lemma 6 essentially says that the Karoubi envelope of the pca A, i. e. the full sub-category of *SD* on A–retracts, constitutes a sub–ccc of $\mathsf{Mod}(A)$. This, probably, is sort of folklore among categorically minded logicians. However, we have preferred to make this insight explicit for the sake of self–containedness.

4. Proof of a Variant of the Longley–Phoa Conjecture

Although we have shown that the Milner model of PCF, i. e. the cpo-enriched order extensional fully abstract model of PCF, is (isomorphic to) the realisability model over A, one still might complain about the fairly abstract nature of A as the very idea of realisability models is to build 'denotational' models from 'operational' universes of algorithms as given by pca's. However, using the fact that $\mathsf{Mod}(A)$ hosts the Milner model of PCF we will construct a fully abstract realisability model of PCF over a combinatory algebra L which appears as an appropriate quotient of an appropriate untyped functional programming language $\mathcal{L}$. This we consider as a proof of a variant of the so-called Longley–Phoa Conjecture, see [5], claiming that realisability models over untyped λ-calculus give rise to fully abstract models of PCF. The only difference is that λ-calculus is replaced by a core functional programming language $\mathcal{L}$ which we consider as even more 'realistic' than the pure λ-calculus since it avoids the coding of natural numbers e. g. by 'Church numerals'.

Our language $\mathcal{L}$ is 'the' programming language associated with the recursive type $A = N \oplus [A\to A]_\perp$, namely untyped call-by-name λ-calculus with arithmetic and a further unary operation $\mathbf{where}$. More explicitly, we have a constant $\underline{n}$ for every $n \in \mathbb{N}$, untyped versions of the arithmetic PCF operations **succ**, **pred** and the conditional $\mathbf{cond}$. The unary operation $\mathbf{where}$ tests whether its argument is a number or a function as given by the following rules:

$$\frac{M \Downarrow \underline{n}}{\mathbf{where}\, M \Downarrow \underline{0}} \qquad \frac{M \Downarrow \lambda x.N}{\mathbf{where}\, M \Downarrow \underline{1}}$$

For an exhaustive study of a similar language for a slightly different recursive type see [12]. The interpretation of $\mathcal{L}$ in A is straightforward and $[\![M]\!]_A$ stands for the denotation of a closed $\mathcal{L}$-term M in A.

Notice that $\mathcal{L}$-definable elements of A are closed under application and functional abstraction and, therefore, constitute a sub–pca of A.

DEFINITION 8. *Let L be the sub–pca of A on $\mathcal{L}$–definable elements of A, i. e. the denotations of closed $\mathcal{L}$ terms.*

Furthermore, for every modest set $X \in \mathsf{Mod}(A)$*, let* $X_L \in \mathsf{Mod}(L)$ *be the modest set with* $|X_L| = \{x \in |X| \mid \exists a {\in} L.\ a \Vdash_X x\}$ *and* $a \Vdash_{X_L} x$ *iff* $a \Vdash_X x$*. Then X_L is called the* L-restriction *of X. For PCF types σ we write σ_L for the L-restriction of* $[\![\sigma]\!]_{\mathsf{Mod}(A)}$. $\diamond$

LEMMA 9. *For every PCF type σ and for every closed PCF term t of type σ there is an $\mathcal{L}$ term M satisfying* $p_\sigma([\![M]\!]_A) = [\![t]\!]_{SD}$*. Accordingly, the PCF–definable elements of σ, in particular the compact ones, are all contained in* $\sigma_L = p_\sigma[L]$.

Proof. (Sketch) By induction on the structure of the judgement $\Gamma \vdash t : \sigma$ one easily shows that its interpretation $[\![\Gamma \vdash t]\!]_{SD} : [\![\Gamma]\!]_{SD} \to [\![\sigma]\!]_{SD}$ is realised by an element of L (using the isomorphism of Theorem 7 between the model of PCF in *SD* and the model of PCF in $\mathsf{Mod}(A)$). Thus, the interpretation of PCF in $\mathsf{Mod}(A)$ provides a translation of PCF terms to $\mathcal{L}$ terms—essentially a 'stripping'—whose restriction to closed terms t yields an $\mathcal{L}$ term denoting a realiser for $[\![t]\!]_{SD}$ in L. $\square$

By an easy induction on σ one observes that the retraction $r_\sigma = e_\sigma \circ p_\sigma$ arises as the denotation of an appropriate $\mathcal{L}$ term R_σ.

LEMMA 10. *For every PCF type σ the retraction $r_\sigma = e_\sigma \circ p_\sigma$ is $\mathcal{L}$–definable by a closed term R_σ. Accordingly,* $\sigma_L = r_\sigma[L]$ *and* $d \in p_\sigma[L]$ *iff* $e_\sigma(d) \in L$.

Proof. Obviously, the retraction $r_\iota = e_{\mathrm{num}} \circ p_{\mathrm{num}}$ is $\mathcal{L}$–definable. If $r_\sigma = e_\sigma \circ p_\sigma$ and $r_\tau = e_\tau \circ p_\tau$ are

$\mathcal{L}$–definable, then the retraction $r_{\sigma\to\tau} = e_{\sigma\to\tau} \circ p_{\sigma\to\tau} = e_{\text{fun}} \circ [p_\sigma \to e_\tau] \circ [e_\sigma \to p_\tau] \circ r_{\text{fun}} = e_{\text{fun}} \circ [r_\sigma \to r_\tau] \circ r_{\text{fun}}$ is $\mathcal{L}$–definable, too. Thus, by induction all retractions r_σ are $\mathcal{L}$–definable.

If $d \in p_\sigma[L]$, i.e. $d = p_\sigma(a)$ for some $a \in L$, then $e_\sigma(d) = e_\sigma(p_\sigma(a)) = r_\sigma(a) \in L$. On the other hand, $e_\sigma(d) \in L$ entails $d = p_\sigma\big(e_\sigma(d)\big) \in p_\sigma[L] = \sigma_L$. $\square$

In the next lemma we show that the L-realisable morphisms from σ_L to τ_L are in natural 1-1-correspondence with $(\sigma\to\tau)_L$.

LEMMA 11. *Let σ and τ be PCF types.*

(1) *Let $a \in L$ and $f\colon \sigma_L \to \tau_L$ be an arbitrary set-theoretic function. If $a \Vdash f$, i.e. $(f \circ p_\sigma)(b) = \big(p_\tau \circ p_{\text{fun}}(a)\big)(b)$ for all $b \in L$, then $p_{\sigma\to\tau}(a)$ is the unique $\overline{f} \in SD(\sigma,\tau)$ with $f = \overline{f}{\restriction}L$.*

(2) *If $\overline{f} \in (\sigma\to\tau)_L$, i.e. $\overline{f} = p_{\sigma\to\tau}(a)$ for some $a \in L$, then, for $f = \overline{f}{\restriction}\sigma_L$, we have $f\colon \sigma_L \to \tau_L$ and $e_{\text{fun}}\big(p_{\text{fun}}(a) \circ r_\sigma\big) \Vdash f$.*

Thus, $[\sigma_L \to \tau_L]$ is canonically isomorphic to $(\sigma\to\tau)_L$ in $\mathsf{Mod}(L)$.

Proof. For (1) suppose that $a \Vdash f$, i.e. $(f \circ p_\sigma){\restriction}L = \big(p_\tau \circ p_{\text{fun}}(a)\big){\restriction}L$. Then, for $d \in L$, we have $f(d) = (f \circ p_\sigma \circ e_\sigma)(d) = \big(p_\tau \circ p_{\text{fun}}(a) \circ e_\sigma\big)(d) = p_{\sigma\to\tau}(a)(d)$ as $e_\sigma(d) \in L$ by Lemma 10. Uniqueness of $p_{\sigma\to\tau}$ follows from the fact that σ_L contains all compact elements of $[\![\sigma]\!]_{SD}$ by Lemma 9.

For (2) suppose $a \in L$ and write $\overline{f} = p_{\sigma\to\tau}(a)$. Then we have for all $b \in L$ that $p_{\sigma\to\tau}(a)\big(p_\sigma(b)\big) = \big(p_\tau \circ p_{\text{fun}}(a) \circ e_\sigma\big)\big(p_\sigma(b)\big) = \big(p_\tau \circ p_{\text{fun}}(a) \circ e_\sigma \circ p_\sigma\big)(b)$. Thus, $\overline{f}\big(p_\sigma(b)\big) = p_\tau\big(\big(p_{\text{fun}}(a) \circ r_\sigma\big)(b)\big)$ for all $b \in L$, i.e. $e_{\text{fun}}\big(p_{\text{fun}}(a) \circ r_\sigma\big) \Vdash f = \overline{f}{\restriction}\sigma_L$ as desired. $\square$

THEOREM 12. *The model of PCF in $\mathsf{Mod}(L)$ is fully abstract when ι is interpreted as $([\![\iota]\!]_{\mathsf{Mod}(A)})_L$. Moreover, for an appropriate choice of exponentials we have $[\![\sigma]\!]_{\mathsf{Mod}(L)} = \sigma_L$ for every PCF type σ.*

Proof. By induction on the structure of PCF types we get $[\![\sigma]\!]_{\mathsf{Mod}(L)} = \sigma_L$ for every PCF type σ when choosing the exponentials as in Lemma 11. $\square$

Notice that the combinatory algebra L is isomorphic to $\mathcal{L}_{/Th(A)}$ where $Th(A)$ is the congruence on the closed terms of $\mathcal{L}$ induced by interpretation in A. An analogue of Theorem 12 holds for all term models $\mathcal{L}_{/T}$ where T is a sufficiently well-behaved congruence on $\mathcal{L}$. We refer by $\sim_{\mathcal{L}}$ or simply $\sim$ to the obvious 'conversion' relation for $\mathcal{L}$ which is a congruence containing the β-rule and all δ-rules for the operations and constants of $\mathcal{L}$ as needed for computation (i.e. strong enough to derive $M \sim \underline{n}$ whenever $[\![M]\!]_A = n \in \mathbb{N}$).

THEOREM 13. *Suppose T is a congruence on $\mathcal{L}$ with $T \subseteq Th(A)$ and containing $\sim_{\mathcal{L}}$. Then the PCF model in $\mathsf{Mod}(\mathcal{L}_{/T})$ is isomorphic to the PCF model in $\mathsf{Mod}(L) = \mathsf{Mod}(\mathcal{L}_{/Th(A)})$ and, therefore, fully abstract.*

More precisely, we have that for an appropriate choice of exponentials the underlying set of $[\![\sigma]\!]_{\mathsf{Mod}(L_{/T})}$ is σ_L and $[M]_T \Vdash_{\mathsf{Mod}(L_{/T})} d$ iff $p_\sigma([\![M]\!]_A) = d$.

In particular, the model of PCF in $\mathsf{Mod}(\mathcal{L}_{/\sim})$ is fully abstract.

Proof. The claim follows immediately from an analogue of Lemma 11.

The latter holds as it can be shown easily by induction on the structure of PCF types that $[M]_T \Vdash_{\mathsf{Mod}(\mathcal{L}_{/T})} d$ if, and only if, $p_\sigma([\![M]\!]_A) = d$ because $(M,N) \in T$ entails $[\![M]\!]_A = [\![N]\!]_A$ due to the assumption $T \subseteq Th(A)$. $\square$

5. Universality

The question arises whether the realisability models of the previous section are *universal* for PCF, meaning that all elements of PCF types appear as denotations of PCF terms. We have not been able to prove universality for our models of the form $\mathsf{Mod}(\mathcal{L}_{/T})$. Although, by Lemma 9, we know that denotations of closed PCF terms are realised by $\mathcal{L}$–definable elements of A the reverse direction seems to be much more difficult as for every closed $\mathcal{L}$ term M and every PCF type σ one would have to exhibit a PCF term t of type σ such that $p_\sigma\big([\![M]\!]_A\big) = [\![t]\!]_{SD}$. Nevertheless, universality of $\mathsf{Mod}(\mathcal{L}_{/T})$ w.r.t. PCF is not unlikely as S. Abramsky and J. Longley have announced in [6] that they can prove universality w.r.t. PCF for a realisability model over a pca $\mathcal{A}_{wb,\mathit{eff}}$ consisting of effective 'well-bracketed' strategies of a certain 'universal' game. They claim that they can transform any strategy realising an object of $[\![\sigma]\!]_{\mathsf{Mod}(\mathcal{A}_{wb,\mathit{eff}})}$ into an effective strategy for the game $[\![\sigma]\!]_{\mathrm{AJM}}$, i.e. the interpretation of σ in the game model of PCF as described in [1].

Notice, however, that all the L–realisable elements of $[\![\sigma]\!]_{SD}$ can be denoted by terms of SFPL, a general sequential functional language whose types are given by

$$A ::= \mathbf{0} \mid \sigma_\perp \mid A \otimes A \mid A \oplus A \mid X \mid \mu X.\, A$$
$$\sigma ::= A \mid \sigma \times \sigma \mid A \multimap \sigma \mid \mu X.\, \sigma$$

where types of the first class are called *computational*. For every (closed) computational type A its interpretation $[\![A]\!]$ in *SD* has the property that '$\perp$ is sequentially separable' in the sense that there exists an SFPL definable *SD* morphism $\mathsf{sep}\colon [\![A]\!] \to \mathbf{1}_\perp$ with $\mathsf{sep}(x) = \perp$ iff $x = \perp$. The language SFPL consists of typed call-by-value λ-calculus together with the usual constructors and eliminators associated with the various type formers[4]. The distinction be-

[4]For every type a fixed point operator can be defined using recursive types as in [2].

tween computational and general types is motivated by the fact that for computational types the syntactic and semantic notions of value coincide, i. e. for closed terms $M{:}A$ the term M reduces to some syntactic value V iff $[\![M]\!] \neq \bot$. Therefore, one can provide the eliminators for $A \oplus B$ and $A \otimes B$ with an operational semantics, which is impossible if, for A or B, $\bot$ is not sequentially separable. Notice, however, that the restrictions of the SFPL type system are not severe as for example we can define $\sigma \to \tau$ as $\sigma_\bot \multimap \tau$ and $\sigma + \tau$ as $\sigma_\bot \oplus \tau_\bot$. For a detailed presentation of SFPL see Tables 1–4 below.

Whereas Marz's language SFL (see [9]) appears as a restriction of λ-calculus with recursive types to an operationally meaningful fragment, SFPL is rather an extension of a call-by-value functional programming language (as e. g. FPC) subsuming the call-by-name paradigm.

In contrast to SFL, SFPL contexts are of the form $\Gamma \equiv x_1{:}A_1, \ldots, x_n{:}A_n$ where $A_1, \ldots, A_n$ are required to be computational types, the interpretation of Γ

Table 1. Types of the language SFPL

Computational types

$A ::= \mathbf{0} \mid \sigma_\bot \mid A \otimes A \mid A \oplus A \mid X \mid \mu X.\, A$

General types

$\sigma ::= A \mid \sigma \times \sigma \mid A \multimap \sigma \mid \mu X.\, \sigma$

SFPL environments

$\Gamma \equiv x_1{:}A_1, \ldots, x_n{:}A_n$ where $A_1, \ldots, A_n$ are computational types

Table 2. Terms of the language SFPL

General terms

$$\begin{aligned} M ::= {} & \Omega \mid x \mid \lambda x{:}A.\, M \mid M(M) \mid \mathbf{inl}(M) \mid \mathbf{inr}(M) \mid \\ & \mathbf{case}\; M \;\mathbf{of}\; \mathbf{inl}(x) \Rightarrow M \;[\!]\; \mathbf{inr}(x) \Rightarrow M \mid \\ & (M, M) \mid \mathbf{let}\; M \;\mathbf{be}\; (x, x) \;\mathbf{in}\; M \mid \\ & \langle M, M \rangle \mid \mathbf{pr_1}(M) \mid \mathbf{pr_2}(M) \mid \\ & \mathbf{thunk}(M) \mid \mathbf{force}(M) \mid \\ & \mathbf{fold}(M) \mid \mathbf{unfold}(M) \end{aligned}$$

Value terms

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

Terminated program terms

$T ::= V \mid \lambda x{:}A.\, M \mid \langle M, M \rangle \mid \mathbf{fold}(T)$

Table 3. Typing rules for SFPL

$$\frac{}{\Gamma \vdash \Omega : \sigma} \qquad \frac{}{\Gamma, x{:}A, \Delta \vdash x : A} \qquad \frac{\Gamma, x{:}A \vdash M : \sigma}{\Gamma \vdash \lambda x{:}A.\, M : A \multimap \sigma} \qquad \frac{\Gamma \vdash M : A \multimap \sigma \quad \Gamma \vdash N : A}{\Gamma \vdash M(N) : \sigma} \qquad \frac{\Gamma \vdash M : A_1}{\Gamma \vdash \mathbf{inl}(M) : A_1 \oplus A_2}$$

$$\frac{\Gamma \vdash M : A_2}{\Gamma \vdash \mathbf{inr}(M) : A_1 \oplus A_2} \qquad \frac{\Gamma \vdash M : A_1 \oplus A_2 \quad \Gamma, x_1{:}A_1 \vdash N_1 : \sigma \quad \Gamma, x_2{:}A_2 \vdash N_2 : \sigma}{\Gamma \vdash \mathbf{case}\; M \;\mathbf{of}\; \mathbf{inl}(x) \Rightarrow N_1 \;[\!]\; \mathbf{inr}(x) \Rightarrow N_2 : \sigma} \qquad \frac{\Gamma \vdash M_1 : A_1 \quad \Gamma \vdash M_2 : A_2}{\Gamma \vdash (M_1, M_2) : A_1 \otimes A_2}$$

$$\frac{\Gamma \vdash M : A_1 \otimes A_2 \quad \Gamma, x_1{:}A_1, x_2{:}A_2 \vdash N : \sigma}{\Gamma \vdash \mathbf{let}\; M \;\mathbf{be}\; (x_1, x_2) \;\mathbf{in}\; N : \sigma} \qquad \frac{\Gamma \vdash M_1 : \sigma_1 \quad \Gamma \vdash M_2 : \sigma_2}{\Gamma \vdash \langle M_1, M_2 \rangle : \sigma_1 \times \sigma_2} \qquad \frac{\Gamma \vdash M : \sigma_1 \times \sigma_2}{\Gamma \vdash \mathbf{pr_1}(M) : \sigma_1} \qquad \frac{\Gamma \vdash M : \sigma_1 \times \sigma_2}{\Gamma \vdash \mathbf{pr_2}(M) : \sigma_2}$$

$$\frac{\Gamma \vdash M : \sigma}{\Gamma \vdash \mathbf{thunk}(M) : \sigma_\bot} \qquad \frac{\Gamma \vdash M : \sigma_\bot}{\Gamma \vdash \mathbf{force}(M) : \sigma} \qquad \frac{\Gamma \vdash M : \sigma[\mu X.\, \sigma / X]}{\Gamma \vdash \mathbf{fold}(M) : \mu X.\, \sigma} \qquad \frac{\Gamma \vdash M : \mu X.\, \sigma}{\Gamma \vdash \mathbf{unfold}(M) : \sigma[\mu X.\, \sigma / X]}$$

Table 4. Operational semantics for SFPL

$$\frac{}{\lambda x{:}A.\, M \Downarrow \lambda x{:}A.\, M} \qquad \frac{M \Downarrow \lambda x{:}A.\, M' \quad N \Downarrow V \quad M'[V/x] \Downarrow T}{M(N) \Downarrow T} \qquad \frac{M \Downarrow V}{\mathbf{inl}(M) \Downarrow \mathbf{inl}(V)} \qquad \frac{M \Downarrow V}{\mathbf{inr}(M) \Downarrow \mathbf{inr}(V)}$$

$$\frac{M \Downarrow \mathbf{inl}(V) \quad N_1[V/x] \Downarrow T}{\mathbf{case}\; M \;\mathbf{of}\; \mathbf{inl}(x) \Rightarrow N_1 \;[\!]\; \mathbf{inr}(x) \Rightarrow N_2 \Downarrow T} \qquad \frac{M \Downarrow \mathbf{inr}(V) \quad N_2[V/x] \Downarrow T}{\mathbf{case}\; M \;\mathbf{of}\; \mathbf{inl}(x) \Rightarrow N_1 \;[\!]\; \mathbf{inr}(x) \Rightarrow N_2 \Downarrow T} \qquad \frac{M_1 \Downarrow V_1 \quad M_2 \Downarrow V_2}{(M_1, M_2) \Downarrow (V_1, V_2)}$$

$$\frac{M \Downarrow (V_1, V_2) \quad N[V_1, V_2 / x_1, x_2] \Downarrow T}{\mathbf{let}\; M \;\mathbf{be}\; (x_1, x_2) \;\mathbf{in}\; N \Downarrow T} \qquad \frac{}{\langle M_1, M_2 \rangle \Downarrow \langle M_1, M_2 \rangle} \qquad \frac{M \Downarrow \langle M_1, M_2 \rangle \quad M_1 \Downarrow T}{\mathbf{pr_1}(M) \Downarrow T} \qquad \frac{M \Downarrow \langle M_1, M_2 \rangle \quad M_2 \Downarrow T}{\mathbf{pr_2}(M) \Downarrow T}$$

$$\frac{}{\mathbf{thunk}(M) \Downarrow \mathbf{thunk}(M)} \qquad \frac{M \Downarrow \mathbf{thunk}(N) \quad N \Downarrow T}{\mathbf{force}(M) \Downarrow T} \qquad \frac{M \Downarrow T}{\mathbf{fold}(M) \Downarrow \mathbf{fold}(T)} \qquad \frac{M \Downarrow \mathbf{fold}(T)}{\mathbf{unfold}(M) \Downarrow T}$$

being the tensor product $[\![A_1]\!] \otimes \cdots \otimes [\![A_n]\!]$. Moreover, the interpretation of a judgement $\Gamma \vdash M : A$ gives rise to a *strict* function $[\![A_1]\!] \otimes \cdots \otimes [\![A_n]\!] \multimap [\![A]\!]$.

Both languages, SFL and SFPL, have the same underlying type system and are equally expressive in the sense that they define the same global elements of closed types. A language related to SFPL, but more general, has been introduced recently by P. Levy in [4]. His intention was to integrate call-by-value and call-by-name into a single language even using arbitrary computational monads rather than just the lifting monad as in SFPL.

As $A \cong N \oplus [A \to A]_\perp$ appears as a (recursive) type of SFPL the language $\mathcal{L}$ (of the previous section) can be understood as a sub-language of SFPL. Moreover, the embeddings e_σ and projections p_σ can quite straightforwardly be expressed as SFPL programs E_σ and P_σ, respectively. Thus, for every closed $\mathcal{L}$ term M, the element $p_\sigma\big([\![M]\!]_A\big)$ appears as denotation of the SFPL term $P_\sigma(M)$. So the question of universality of $\mathsf{Mod}(\mathcal{L}_{/\sim})$ is equivalent to the conservativity of SFPL over PCF, i. e. that, for every SFPL term of PCF type σ, there is an observationally equivalent PCF term. This seems to be a question about functional programming independent from denotational semantics.

However, for the language SFPL itself one obtains a *universal* realisability model $\mathsf{Mod}(\mathcal{L}_{/\sim})$ when instantiating $\mathcal{L}$ by an untyped version of SFPL, namely the programming language associated with the domain equation

$$U \cong U \oplus U \oplus (U \otimes U) \oplus [U \multimap U]_\perp$$

and $\sim$ by the obvious conversion relation for $\mathcal{L}$. Notice that $\mathcal{L}$ constitutes the purely functional core of an untyped sequential call–by–value programming language (‘untyped FPC’). Whereas the pca $A \cong N \oplus [A \to A]_\perp$ is not a type of PCF, our new pca U appears as a type of SFPL, the programming language under consideration, and, therefore, $\mathcal{L}$ appears as a sub-language of SFPL. One shows that $\mathbf{0}$, $U_\perp$, $U \oplus U$, $U \otimes U$, $U \times U$ and $U \multimap U$ all appear as $\mathcal{L}$–definable retracts of U. This can be achieved for a lot of variations of the domain equation for U, as e. g.

$$U \cong \mathbf{1}_\perp \oplus [U \multimap U]_\perp$$

which is the simplest recursive type containing all other SFPL types as definable retracts. Its associated language is lazy[5] call-by-value λ-calculus with an error element and the facility of handling it.

Accordingly, for closed SFPL types σ, we have embeddings $e_\sigma : \sigma \to U$ and projections $p_\sigma : U \to \sigma$ which can be denoted by closed SFPL terms E_σ and P_σ, respectively, and satisfy $p_\sigma \circ e_\sigma = \mathsf{id}_\sigma$. From this it is now straightforward to prove analogues of the relevant theorems and lemmas of the previous sections using almost literally the same arguments exploiting that *SD* provides a fully abstract model for SFPL. This follows from full abstraction of *SD* w.r.t. SFL as shown in [9] since both languages are equally expressive. In particular, we get that for every (closed) SFPL type σ its denotation $[\![\sigma]\!]_{\mathsf{Mod}(\mathcal{L}_{/\sim})}$ is isomorphic to the modest set $(\sigma_L, \Vdash_\sigma)$ where $[M]_\sim \Vdash_\sigma d$ iff $p_\sigma([\![M]\!]_U) = d$. Now universality of $\mathsf{Mod}(\mathcal{L}_{/\sim})$ for SFPL is immediate as, for every closed $\mathcal{L}$ term M, the element $p_\sigma([\![M]\!]_U)$ appears as denotation of the SFPL term $P_\sigma(M)$ implicitly using that $\mathcal{L}$ is a subset of SFPL.

[5] By *lazy* we mean that function types are lifted, i. e. functional abstractions are terminated program terms.

6. Discussion and Further Results

A natural question whose answer we think to be positive is whether the congruence $Th(A)$ coincides with observational equivalence for $\mathcal{L}$. For this it suffices to show that all compact elements of A are $\mathcal{L}$–definable. A tentative proof for this would look as follows. First identify a subset $\mathcal{P}$ of SFPL such that (1) one can denote in $\mathcal{P}$ all compact elements of the finite type approximations A_n of A arising from its inverse limit construction and (2) denotations of $\mathcal{P}$ terms in A can be denoted already by $\mathcal{L}$ programs. A candidate for such a $\mathcal{P}$ would be an extension of PCF by the type former $N \oplus (\_)_\perp$. Now if e is a compact element in A, i. e. a compact element in some A_n, then due to (1) and (2) there is an $\mathcal{L}$–program M denoting e. Thus, we get $e = [\![R_n(M)]\!]_A$ where R_n is the $\mathcal{L}$ term denoting the canonical retraction of A onto A_n.

The question arises whether there may be a more syntactic proof of universality of $\mathsf{Mod}(\mathcal{L}_{/\sim})$ for SFPL avoiding any reference to *SD*. This would amount to proving by induction on the structure of SFPL types σ that $[\![\sigma]\!]_{\mathsf{Mod}(\mathcal{L}_{/\sim})}$ is isomorphic to $(T_\sigma, \Vdash_\sigma)$ where T_σ is the set of closed SFPL terms of type σ modulo observational equivalence and $[M]_\sim \Vdash_\sigma [t]$ iff $P_\sigma(M)$ and t are observationally equal. However, the validity of this claim is already a consequence of the considerations above as U is a fully abstract model for $\mathcal{L}$ which follows from the full abstraction result for *SD* w.r.t. SFL established in [9] and, hence, for SFPL. Thus, a purely syntactic proof would amount to work inside the term model for SFPL modulo observational equivalence. There the required solution of the recursive type equation for U surely exists for trivial reasons, namely as closed terms of type U modulo observational equivalence. Nevertheless, we think working in *SD* due to its abstract nature is more comfortable than working in the term model of SFPL.

Acknowledgements

We thank Hanno Nickau for explaining and discussing the results of [10]. Our common efforts in clarifying the pattern behind the argumentation in *loc. cit.* were the starting point and motivation for the work reported in this paper. Furthermore, we thank John Longley for detailed comments and discussion on a draft version of our work, in particular for insisting on the issue of universality.

References

[1] 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. (A full version of this paper is available from http://theory.doc.ic.ac.uk:80/tfm/papers/MalacariaP/PCFfullabs.dvi.Z).

[2] C. Gunter. *Semantics of Programming Languages. Structures and Techniques*. Foundations of Computing. MIT Press, 1992.

[3] 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.

[4] P. B. Levy. Call-by-push-value: A subsuming paradigm (extended abstract). In J.-Y. Girard, editor, *Typed Lambda-Calculi and Applications*. Springer-Verlag, April 1999. To appear.

[5] J. Longley. *Realizability Toposes and Language Semantics*. PhD thesis, University of Edinburgh, 1994.

[6] J. Longley. Realizability models for sequential computation. Unfinished draft, available from http://www.dcs.ed.ac.uk/home/jrl/pisa.ps.gz, September 1998.

[7] J. Longley. The sequentially realizable functionals. Unpublished draft, available from http://www.dcs.ed.ac.uk/home/jrl/sequential.dvi.gz, November 1998.

[8] J. R. Longley and A. E. Simpson. A uniform approach to domain theory in realizability models. *Mathematical Structures in Computer Science*, 7:469–506, 1997.

[9] M. Marz. 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.

[10] H. Nickau and C.-H. L. Ong. Full completeness, effectively almost-everywhere copycat strategies and the longley conjecture. Unpublished draft, April 1998.

[11] P. W. O'Hearn and J. G. Riecke. Kripke logical relations and PCF. *Information and Computation*, 120(1):107–116, 1995.

[12] A. M. Pitts. Some notes on inductive and co-inductive techniques in the semantics of functional programs. Notes Series BRICS-NS-94-5, BRICS, Department of Computer Science, University of Aarhus, Dec. 1994. vi+135 pp, draft version.

[13] G. D. Plotkin. Post-graduate lecture notes in advanced domain theory (incorporating the "Pisa Notes"). Dept. of Computer Science, Univ. of Edinburgh, 1981.

[14] J. 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.

[15] D. S. Scott. Relating theories of lambda calculus. In J. R. Hindley and J. P. Seldin, editors, *To H. B. Curry: Essays in Combinatory Logic, Lambda Calculus and Formalism*, pages 403–450. Academic Press, 1980.

[16] T. Streicher. *Semantics of Type Theory*. Progress in Theoretical Computer Science. Birkhäuser, 1991.

[17] T. Streicher. Full abstraction via realisability. Unpublished manuscript, available from http://www.mathematik.tu-darmstadt.de/~streicher/far.ps.gz, August 1998.

[18] J. van Oosten. A combinatorry algebra for sequential functionals of finite type. Preprint 996, Department of Mathematics, Utrecht University, 1997.