View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Edinburgh Research Explorer

THE UNIVERSITY of EDINBURGH

Edinburgh Research Explorer

Matching typed and untyped readability (Extended abstract)

Citation for published version:
Longley, J 2000, 'Matching typed and untyped readability (Extended abstract)' Electronic Notes in Theoretical Computer Science, vol. 35, pp. 109-132. DOI: 10.1016/S1571-0661(05)80734-0

Digital Object Identifier (DOI):
10.1016/S1571-0661(05)80734-0

Link:
Link to publication record in Edinburgh Research Explorer

Document Version:
Publisher's PDF, also known as Version of record

Published In:
Electronic Notes in Theoretical Computer Science

General rights
Copyright for the publications made accessible via the Edinburgh Research Explorer is retained by the author(s) and / or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights.

Take down policy
The University of Edinburgh has made every reasonable effort to ensure that Edinburgh Research Explorer content complies with UK legislation. If you believe that the public display of this file breaches copyright please contact openaccess@ed.ac.uk providing details, and we will remove access to the work immediately and investigate your claim.

Download date: 05. Apr. 2019

Electronic Notes in Theoretical Computer Science 35 (2000)
URL: http://www.elsevier.nl/locate/entcs/volume35.html 24 pages

Matching typed and untyped realizability (Extended abstract)

John Longley

Laboratory for the Foundations of Computer Science, JCMB
The King's Buildings, Mayfield Road
Edinburgh EH9 3JZ, U.K.
e-mail: jrl@dcs.ed.ac.uk

Abstract

Realizability interpretations of logics are given by saying what it means for computational objects of some kind to *realize* logical formulae. The computational objects in question might be drawn from an untyped universe of computation, such as a partial combinatory algebra, or they might be typed objects such as terms of a PCF-style programming language. In some instances, one can show that a particular untyped realizability interpretation matches a particular typed one, in the sense that they give the same set of realizable formulae. In this case, we have a very good fit indeed between the typed language and the untyped realizability model—we refer to this condition as *(constructive) logical full abstraction*.

We give some examples of this situation for a variety of extensions of PCF. Of particular interest are some models that are logically fully abstract for typed languages including *non-functional* features. Our results establish connections between what is computable in various programming languages, and what is true inside various realizability toposes. We consider some examples of logical formulae to illustrate these ideas, in particular their application to exact real-number computability.

The present article summarizes the material I presented at the Domains IV workshop, plus a few subsequent developments; it is really an extended abstract for a projected journal paper. No proofs are included in the present version.

0 Introduction

It is well-known that realizability models provide a good supply of denotational models for a range of functional programming languages. In the most familiar situation, one starts with a partial combinatory algebra A, and constructs the category $\mathbf{M}od(A)$ of *modest sets* over A (or equivalently the category $\mathbf{P}ER(A)$ of partial equivalence relations on A). Since many familiar PCAs (e.g. $K_1, \mathcal{P}\omega_{re}, K_{2re}, \Lambda^0/T$ for any λ-theory T) consist of effective objects of

©2000 Published by Elsevier Science B. V. Open access under CC BY-NC-ND license.

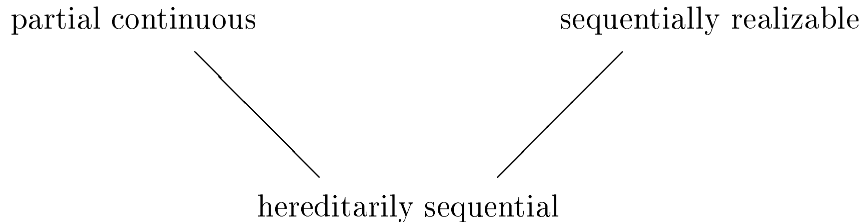
some kind, the corresponding categories have a notion of computability built into them: all the morphisms are computable in some sense.

Interestingly, different PCAs embody different notions of computability. For example, we can often pick out an object of $\mathbf{M}od(A)$ playing the role of $N_\perp$ (usually this is obvious), and then consider the finite types in $\mathbf{M}od(A)$ generated from $N_\perp$ by exponentiation. Taking global elements of these objects (i.e. applying the functor $\mathrm{Hom}(1, -)$), we obtain a *finite type structure*, which we can think of as the class of "computable" finite-type partial functionals relative to A. An interesting question is which PCAs give rise to which finite type structures.

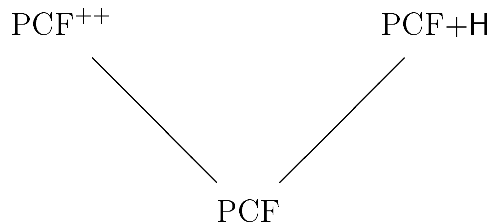
At present, it seems that there are essentially three different finite type structures that occur widely in nature, each existing in a "full continuous" and an "effective" flavour. All six of these type structures have a number of different characterizations, and have some claim to being mathematically natural objects of study. The three full type structures are:

- The *partial continuous* functionals: that is, the finite type structure arising from the familiar Scott domain model [30].
- The *hereditarily sequential* functionals of Nickau [20]: this coincides with the finite type structure arising from the fully abstract game models for PCF due to Abramsky, Hyland *et al* [1,8].
- The *strongly stable* functionals of Bucciarelli and Ehrhard [3]: these coincide with the *sequentially realizable* functionals of Longley [15].

Intuitively, the type structure of hereditarily sequential functionals is smaller than the other two (more precisely, it is a subquotient of either of the others):



Each of these type structures has a natural effective analogue. Rather remarkably, in each case one can find a programming language (with a decidable set of terms and an effective operational semantics) which defines precisely the functionals in the effective type structure:



One can therefore characterize these effective type structures as the closed term models for the respective programming languages. Here PCF^{++} is the extension of PCF with parallel-or and exists operators as studied in [25]. For the functional H, see [15].

There are known examples of PCAs giving rise to each of these six type structures:

- The partial continuous functionals arise from many "continuous PCAs" such as the Scott graph model $\mathcal{P}\omega$ [29], the D_∞ models (in complete lattices or CPOs) [28], Plotkin's universal domain T^ω [26], and Kleene's second model K_2 [11].
- The effective partial continuous functionals (corresponding to PCF^{++}) arise from the effective analogues of each of the above PCAs, as well as from Kleene's first model K_1 [10] (the best-known example).
- The hereditarily sequential functionals arise from some PCAs recently constructed by Abramsky (see [14]), as well as from PCAs obtained by solving various recursive domain equations in known fully abstract models of PCF, such as categories of games or sequential domains (see [19]).
- The effective hereditarily sequential functionals (i.e. the PCF-definable functionals) arise from the effective analogues of any of these. Moreover, the *Longley-Phoa Conjecture* asserts that this type structure also arises from the term model Λ^0/T for any semi-sensible λ-theory T (see e.g. [13]).
- The sequentially realizable (SR) functionals arise from van Oosten's combinatory algebra $\mathcal{B}$ [23], and from the equivalent combinatory algebra $\mathcal{A}$ constructed by Abramsky (see [14]). They also arise from the combinatory algebra $\mathcal{B}_2$ described in [15].
- The effective SR functionals arise from the effective analogues of any of these.

All these PCAs yield realizability models that are fully abstract for the appropriate programming languages, and moreover, the effective ones even yield models that are *universal* (that is, every element of the model of appropriate type is denotable by a term of the language). Universality is already a strong criterion for goodness of fit between a language and a model; but since we have a choice of universal models for each of our three languages, it is natural to ask how they differ one from another, and whether some are "better" than others in some sense. That is, can we find a stronger "goodness of fit" criterion than universality?

The purpose of the present paper is to introduce and study one such criterion, namely *(constructive) logical full abstraction*. This criterion asserts that the logic of realizability embodied by the PCA agrees with a notion of realizability derived from the programming language itself. We will see that this criterion does indeed introduce useful distinctions between PCAs that realize the same type structure, and will give examples of logically fully abstract

models for each of our languages. Moreover, we will show that some of the above PCAs actually provide models that are logically fully abstract for *non-functional* extensions of PCF (in a sense we shall define). Finally, we will look at some examples of logical formulae that show up the differences between the various realizability interpretations, to illustrate how logical formulae can be used to express information about what is and is not computable in various kinds of programming language.

The notion of logical full abstraction (LFA) was first sketched in Chapter 8 of the author's Ph.D. thesis [13], in both a *classical* and a (stronger) *constructive* version. The classical notion of LFA was further studied in [17]; the purpose of the present paper is to study the constructive notion in more detail.

1 Preliminary definitions

1.1 Realizability models

We first summarize some definitions concerning realizability models and fix some notation. The reader may consult [13] for more details and further background information.

Definition 1.1 [PCA] A *partial combinatory algebra* (PCA) consists of a set A together with a partial binary operation $\cdot : A \times A \rightharpoonup A$ (called *application*, and treated as left-associative) such that there exist elements $k, s \in A$ satisfying

$$k \cdot x \cdot y \ = \ x, \qquad s \cdot x \cdot y \cdot z \ \succeq \ x \cdot z \cdot (y \cdot z)$$

for all $x, y, z \in A$.

Here the symbol $\succeq$ means "if the RHS is defined, so is the LHS and they are equal". The above definition is thus slightly more liberal than the more usual definition of PCA, but all the relevant theory goes through unaffected. We often abbreviate $a \cdot b$ by ab, and write i for skk (note that $ix = x$ for all $x \in A$).

In any PCA, one can define a *pairing* operation $\langle -, - \rangle$, e.g. by $\langle x, y \rangle = s(si(kx))(ky)$. The corresponding *projections* are defined by $\mathit{fst} = k$ and $\mathit{snd} = ki$; note that $\mathit{fst}\langle x, y \rangle = x$ and $\mathit{snd}\langle x, y \rangle = y$.

Definition 1.2 [Modest sets] Let A be a PCA.

(i) A *modest set* X over A consists of an underlying set $|X|$, and for each $x \in |X|$ a non-empty set $\|x\| \subseteq A$ of *realizers* for x, such that if $a \in \|x\|$ and $a \in \|x'\|$ then $x = x'$. We sometimes write $x \in X$ in place of $x \in |X|$.

(ii) A *morphism* $f : X \to Y$ between modest sets is a function $f : |X| \to |Y|$ for which there exists $r \in A$ such that for all $x \in |X|$ and $a \in \|x\|$ we have $r \cdot a \in \|f(x)\|$. In this situation we say that r *tracks* f. We write $\mathbf{M}od(A)$ for the category of modest sets over A.

The category $\mathbf{M}od(A)$ is cartesian-closed. Given modest sets X, Y, the exponential Y^X is constructed as follows: $|Y^X|$ is the set of morphisms $f : X \to Y$; and $\|f\|$ is the set of elements $r \in A$ that track f.

$\mathbf{M}od(A)$ also has a natural number object N. For any non-trivial PCA A, this may be constructed as follows: let $|N|$ be the set N of natural numbers, and let $\|n\|$ be the singleton set $\{\overline{n}\}$, where $\overline{n}$ is the *Curry numeral* for n:

$$\overline{0} \;=\; \langle ki, i\rangle, \qquad \overline{n+1} \;=\; \langle k, \overline{n}\rangle$$

It is easy to see that $\mathbf{M}od(A)$ is equivalent to the well-known category $\mathbf{P}ER(A)$ of partial equivalence relations on A. In fact, $\mathbf{M}od(A)$ embeds as a full sub-CCC in the (standard) *realizability topos* $\mathbf{R}T(A)$, though the latter is more complicated to construct and we shall not need it here.

In order to interpret languages such as PCF in $\mathbf{M}od(A)$, we want an object to play the role of $N_\perp$. We can obtain such an object if we have the following piece of extra structure on our PCA:

Definition 1.3 Let A be a PCA. A *non-termination set* in A is a non-empty set $E \subseteq A$ such that, for all $a, b \in A$, if $a \in E$ then $sab \in E$. Any non-termination set E gives rise to a *lift operation* $-_\perp$ on objects of $\mathbf{M}od(A)$ as follows: let $|X_\perp| = |X| \sqcup \{\perp\}$; for $x \in |X|$, let $\|x\|_{X_\perp} = \{\langle a, b\rangle \mid ai = i, bi \in \|x\|_X\}$; and let $\|\perp\|_{X_\perp} = \{\langle a, b\rangle \mid a \in E,\, b \in A\}$.

The lift operation $-_\perp$ in fact extends to a monad on $\mathbf{M}od(A)$, but here all we really need is the object $N_\perp$. The notion of non-termination set here replaces the notion of *divergence* from [13,18]; indeed, in the familiar cases, every divergence gives rise to a non-termination set and *vice versa.* However, the definition of non-termination set is both cleaner and slightly more robust (but less intuitive!).

Let us say that a *choice of natural number domain* (or *choice of $N_\perp$*) in a cartesian-closed category $\mathcal{C}$ is simply an object $N_\perp$ of $\mathcal{C}$ with a canonical identification $|N_\perp| \cong \mathrm{N} \sqcup \{\perp\}$. The natural number object in $\mathbf{M}od(A)$ together with a non-termination set gives rise to a choice of natural number domain, though we may on occasion be interested in choices of $N_\perp$ not of this form. Technically, the choice of natural number domain is part of the data for a realizability model; however, in many cases of interest there is only one natural candidate for $N_\perp$ that stands out, and so we shall not always bother to mention it explicitly.

We can now interpret the *finite types* in any realizability model. The finite types are built up from a single ground type ι via the (right-associative) binary type constructors $\times$ and $\to$.

Definition 1.4 [Finite type structure] An *(extensional, partial) finite type structure (FTS)* T consists of a set T^σ for each finite type σ such that $T^\iota = \mathrm{N} \sqcup \{\perp\}$ and $T^{\sigma\times\tau} = T^\sigma \times T^\tau$, together with "application" functions $\cdot_{\sigma\tau} : T^{\sigma\to\tau} \times T^\sigma \to T^\tau$ such that, for any $f, g \in T^{\sigma\to\tau}$, if $f \cdot x = g \cdot x$ for all $x \in T^\sigma$ then $f = g$.

In any cartesian-closed category $\mathcal{C}$ equipped with a choice of $N_\perp$, we have an interpretation $[\![-]\!]$ of the finite types defined by

$$[\![\iota]\!] = N_\perp, \qquad [\![\sigma \times \tau]\!] = [\![\sigma]\!] \times [\![\tau]\!], \qquad [\![\sigma \to \tau]\!] = [\![\tau]\!]^{[\![\sigma]\!]}.$$

We hence obtain a finite type structure $T(\mathcal{C}, N_\perp)$, where $T(\mathcal{C})^\sigma = |[\![\sigma]\!]|$, and the application operations are given by the evaluation morphisms in $\mathcal{C}$. In the case $\mathcal{C} = \mathbf{M}od(A)$, we write this simply as $T(A, N_\perp)$, or $T(A, E)$ if the choice of $N_\perp$ arises from the non-termination set E. More loosely, we may write it as $T(A)$ and refer to it as the *FTS over* A.

1.2 Typed programming languages

Next we introduce some general notions concerning typed programming languages. By a *language* $\mathcal{L}$ let us mean a family of sets $\mathcal{L}_\sigma$ of *terms* of type σ, with the following closure properties:

- if $M \in \mathcal{L}_{\sigma\times\tau}$ then $\mathsf{fst}_{\sigma\tau} M \in \mathcal{L}_\sigma$ and $\mathsf{snd}_{\sigma\tau} M \in \mathcal{L}_\tau$,
- if $M \in \mathcal{L}_{\sigma\to\tau}$ and $N \in \mathcal{L}_\sigma$ then $MN \in \mathcal{L}_\tau$.

We suppose that each term M has a set of *free variables* $\mathrm{FV}(M)$, such that $\mathrm{FV}(\mathsf{fst}_{\sigma\tau} M) = \mathrm{FV}(\mathsf{snd}_{\sigma\tau} M) = \mathrm{FV}(M)$ and $\mathrm{FV}(MN) = \mathrm{FV}(M) \cup \mathrm{FV}(N)$; we write $\mathcal{L}^0_\sigma$ for the set $\{M \in \mathcal{L}_\sigma \mid \mathrm{FV}(M) = \emptyset\}$ of *closed* terms of type σ. If Γ is a finite non-repetitive list of variables in which all the free variables of M appear, we may say M is a *term in context* Γ. We also assume we have a notion of *substitution* for terms of $\mathcal{L}$, interacting with free variables in the expected way. Finally we suppose we are given an *evaluation* function $\mathrm{Eval}_{\mathcal{L}} : \mathcal{L}^0_\iota \to \mathrm{N}_\perp$.

A *translation* θ from $\mathcal{L}$ to $\mathcal{L}'$ consists of a family of functions $\theta_\sigma : \mathcal{L}_\sigma \to \mathcal{L}'_\sigma$ that preserve projections, application, and free variables, and such that for $M \in \mathcal{L}^0_\iota$ we have $\mathrm{Eval}_{\mathcal{L}'}(\theta_\iota M) = \mathrm{Eval}_{\mathcal{L}}(M)$. If such a translation exists, we may think of $\mathcal{L}$ as a sublanguage of $\mathcal{L}'$.

For any language $\mathcal{L}$, we can obtain a partial equivalence relation $\approx_\sigma$ on each $\mathcal{L}^0_\sigma$ as follows:

- $M \approx_\iota N$ iff $\mathrm{Eval}_{\mathcal{L}}(M) = \mathrm{Eval}_{\mathcal{L}}(N)$,
- $M \approx_{\sigma\times\tau} N$ iff $\mathsf{fst}_{\sigma\tau} M \approx_\sigma \mathsf{fst}_{\sigma\tau} N$ and $\mathsf{snd}_{\sigma\tau} M \approx_\tau \mathsf{snd}_{\sigma\tau} N$,
- $M \approx_{\sigma\to\tau} N$ iff $MP \approx_\tau NQ$ whenever $P \approx_\sigma Q$.

We may extend $\approx_\sigma$ to open terms as follows: if M, N are terms in context $x_1^{\sigma_1}, \ldots, x_r^{\sigma_r}$, then $M \approx_\sigma N$ iff for all closed terms $P_1, \ldots, P_r, Q_1, \ldots, Q_r$ such that $P_i \approx_{\sigma_i} Q_i$ for each i, we have $M[\vec{P}/\vec{x}] \approx_\sigma N[\vec{Q}/\vec{x}]$.

We say a term $M : \sigma$ is *functional* if $M \approx_\sigma M$; we say a language is *functional* if all its terms are functional. For any language $\mathcal{L}$, the sublanguage consisting of functional terms is a functional language, which we may call the *functional core* (or *Gandy hull*) of $\mathcal{L}$.

Given a functional language $\mathcal{L}$ and a cartesian-closed category $\mathcal{C}$ with choice of $N_\perp$, an *interpretation* of $\mathcal{L}$ in $(\mathcal{C}, N_\perp)$ assigns to every term $M \in \mathcal{L}_\tau$

in every context $\Gamma = x_1^{\sigma_1}, \ldots, x_r^{\sigma_r}$ a morphism $[\![M]\!]_\Gamma : [\![\sigma_1]\!] \times [\![\sigma_r]\!] \to [\![\tau]\!]$ in such a way that composition reflects substitution. Such an interpretation is *adequate* if for all $M \in \mathcal{L}_\iota^0$ we have $[\![M]\!] = \mathrm{Eval}(M)$; it is *universal* if for any morphism $f : [\![\sigma_1]\!] \times [\![\sigma_r]\!] \to [\![\tau]\!]$ there is a term $M \in \mathcal{L}_\tau$ in context $\Gamma = x_1^{\sigma_1}, \ldots, x_r^{\sigma_r}$ such that $[\![M]\!]_\Gamma = f$. If there is an adequate interpretation of $\mathcal{L}$ in $(\mathcal{C}, N_\perp)$ we say that $(\mathcal{C}, N_\perp)$ is a *model* of $\mathcal{L}$.

In the case of a realizability model $\mathbf{M}od(A)$, we will without comment identify morphisms $1 \to [\![\sigma]\!]$ with elements of $[\![\sigma]\!]$. Furthermore, if ν is a valuation assigning to the variables $x_i^{\sigma_i} \in \Gamma$ an element $\nu(x) \in [\![\sigma_i]\!]$, and $M : \sigma$ is a term in context Γ, we will write $[\![M]\!]^\nu$ for the element $[\![M]\!]_\Gamma(\nu(x_1), \ldots, \nu(x_r))$ of $[\![\sigma]\!]$.

2 Untyped and typed realizability

Let $\mathcal{L}$ be any functional language such that $0, 1 \in \mathcal{L}_\iota^0$. We will consider the class $\mathbf{J}(\mathcal{L})$ of logical formulae given by the following grammar:

$$\phi \ ::= \ M =_\sigma N \mid P\downarrow \mid \phi_1 \wedge \phi_2 \mid \phi_1 \Rightarrow \phi_2 \mid \exists x^\sigma.\phi_1 \mid \forall x^\sigma.\phi_1$$

where $M, N : \sigma$ and $P : \iota$ range over terms of $\mathcal{L}$, and x^σ ranges over variables of $\mathcal{L}$. Intuitively we have an equality predicate at each type σ, and a termination predicate at ground type; we will usually omit the subscript in equality formulae. We will write *true*, *false* for the formulae $0 = 0, 0 = 1$ respectively, and $\neg\phi$ as sugar for $\phi \Rightarrow$ *false*. Note that we have omitted disjunction from the logic (see below); however, we may express disjunctions by translating $\phi_1 \vee \phi_2$ to

$$\exists n^\iota.\, n\downarrow \wedge (n = 0 \Rightarrow \phi_1) \ \wedge \ ((\neg n = 0) \Rightarrow \phi_2).$$

2.1 Untyped realizability

We recall the standard notion of untyped realizability for formulae of $\mathbf{J}(\mathcal{L})$. Suppose A is a PCA and E a non-termination set such that $\mathcal{L}$ has an adequate interpretation $[\![-]\!]$ in $\mathbf{M}od(A)$ as above. Then we may define a relation $a\ \mathbf{r}^\nu\ \phi$ (read "a realizes ϕ under ν") between elements $a \in A$, valuations ν and formulae $\phi \in \mathbf{J}(\mathcal{L})$ whose free variables are in ν as follows:

- If $[\![M]\!]^\nu = [\![N]\!]^\nu$, then $a\ \mathbf{r}^\nu\ M = N$ for any $a \in A$.
- If $[\![P]\!]^\nu \in \mathrm{N}$, then $a\ \mathbf{r}^\nu\ P\downarrow$ for any $a \in A$.
- If $\mathit{fst}\, a\ \mathbf{r}^\nu\ \phi$ and $\mathit{snd}\, a\ \mathbf{r}^\nu\ \psi$, then $a\ \mathbf{r}^\nu\ \phi \wedge \psi$.
- If $ab\ \mathbf{r}^\nu\ \psi$ whenever $b\ \mathbf{r}^\nu\ \psi$, then $a\ \mathbf{r}^\nu\ \phi \Rightarrow \psi$.
- If, for some $e \in [\![\sigma]\!]$, $\mathit{fst}\, a \in ||e||$ and $\mathit{snd}\, a\ \mathbf{r}^{\nu(x^\sigma \mapsto e)}\ \phi$, then $a\ \mathbf{r}^\nu\ \exists x^\sigma.\phi$.
- If, for all $e \in [\![\sigma]\!]$, we have $ab\ \mathbf{r}^{\nu(x^\sigma \mapsto e)}\ \phi$ whenever $b \in ||e||$, then $a\ \mathbf{r}^\nu\ \forall x^\sigma.\phi$.
- That's all.

We write just $a\ \mathbf{r}\ \phi$ if a realizes ϕ under the empty valuation. If there exists $a \in A$ such that $a\ \mathbf{r}\ \phi$, we write $(A, E) \models \phi$ (or just $A \models \phi$), and say that ϕ *is*

realizable in A. This notion of realizability is exactly the one arising from the *internal logic* of $\mathbf{M}od(A)$ (or of $\mathbf{R}T(A)$); indeed, one can give an equivalent definition of the relation $\models$ by exploiting the categorical structure of $\mathbf{M}od(A)$ (see [13, page 262]). However, the concrete definition in terms of realizers is perhaps easier to grasp, and is better suited to our present purposes.

It is interesting to note that, for the double-negation fragment of $\mathbf{J}(\mathcal{L})$ (i.e. the image of the Gödel double-negation translation $\phi \mapsto \phi^\circ$), the above interpretation agrees with a simple *classical* interpretation of logic in the finite type structure $T(A)$. That is, we have $A \models \phi^\circ$ iff $T(A) \models \phi$ (see [13, Chapter 8] for the easy definition of satisfaction in $T(A)$). Semantically, this corresponds to the fact that passing from $\mathbf{M}od(A)$ (or $\mathbf{R}T(A)$) to the FTS corresponds to taking global elements; and the global elements functor $\mathrm{Hom}(1, -) : \mathbf{R}T(A) \to \mathbf{S}et$ is exactly the reflection from $\mathbf{R}T(A)$ to its double-negation sheaf subtopos. What this means is that if two realizability models yield the same FTS, then the corresponding relations $\models$ agree on the double-negation fragment of $\mathbf{J}(\mathcal{L})$. (In fact, the converse also holds in the cases of interest: see [17].) However, they may well disagree on the rest of $\mathbf{J}(\mathcal{L})$: for example, the PCAs K_1 and $\mathcal{P}\omega_{re}$ give the same FTS but yield quite different realizability interpretations (see below). To summarize, the FTS only embodies information about the double-negation fragment of the internal logic.

It may be argued that this classical fragment of the logic is enough for many practical purposes, e.g. for reasoning about programs (see [13, Chapter 9]). However, it is still natural to ask whether we can find a use for the finer distinctions between models given by their internal logic. This is the purpose of the present paper.

Several variants of the above definitions are possible. In particular, one can define the Kreisel-style *modified realizability* relation $a \ \mathbf{m}r \ \phi$, giving rise to the satisfaction relation $A \models_m \phi$, though we will not give the details here. (For background on modified realizability see [24]).

2.2 Typed realizability

The above gives an interpretation for formulae of $\mathbf{J}(\mathcal{L})$ relative to a particular model $\mathbf{M}od(A)$, which we think of as a "semantic" model for $\mathcal{L}$. We now present an alternative, more "syntactic" notion of realizability, defined purely in terms of the typed programming language and without reference to any particular model. (Our hope is that such an interpretation could be grasped relatively easily by a programmer without any background in denotational semantics.)

The new definition of realizability is closely parallel to the one above, except that realizers are now terms of the typed programming language itself rather than elements of an untyped structure. Let $\mathcal{L}$ be any language, and $\mathcal{L}'$ its functional core. In order to obtain a pleasant logic, *terms* will be drawn only from $\mathcal{L}'$, so that e.g. the extensionality rule holds. (This will

ensure that our logic agrees with the usual internal logic of finite types in certain toposes.) However, *realizers* for formulae are allowed to be possibly non-functional programs, drawn from the whole of $\mathcal{L}$.

Formally, we define a relation $M\ \mathbf{R}\ \phi$ between closed terms M of $\mathcal{L}$ and closed formulae ϕ of $\mathbf{J}(\mathcal{L}')$ inductively as follows:

- If $N \approx_\sigma N'$, then $M\ \mathbf{R}\ (N =_\sigma N')$ for any $M \in \mathcal{L}^0_\iota$.
- If $P : \iota$ terminates, then $M\ \mathbf{R}\ (P\downarrow)$ for any $M \in \mathcal{L}^0_\iota$.
- If $\mathsf{fst}_{\sigma\tau} M\ \mathbf{R}\ \phi$ and $\mathsf{snd}_{\sigma\tau} M\ \mathbf{R}\ \psi$, then $M\ \mathbf{R}\ \phi \wedge \psi$.
- If $MN\ \mathbf{R}\ \psi$ whenever $N\ \mathbf{R}\ \phi$, then $M\ \mathbf{R}\ \phi \Rightarrow \psi$.
- If $\mathsf{fst}_{\sigma\tau} M : \sigma$ and $\mathsf{snd}_{\sigma\tau} M\ \mathbf{R}\ \phi[M/x^\sigma]$, then $M\ \mathbf{R}\ \exists x^\sigma.\phi$.
- If $MN\ \mathbf{R}\ \phi[N/x^\sigma]$ whenever $N : \sigma$, then $M\ \mathbf{R}\ \forall x^\sigma.\phi$.
- That's all.

If there exists M such that $M\ \mathbf{R}\ \phi$, we write $\mathcal{L} \models \phi$ and say that ϕ *is realizable in* $\mathcal{L}$. Note that any realizers for ϕ must be of a type $\tau(\phi)$ that can easily be read off from the structure of ϕ; we may think of $\tau(\phi)$ as the type of "potential realizers" for ϕ. (We can now see difficulty with disjunction: we would like the type of realizers for $\phi \vee \psi$ to be a disjoint sum type, but such types are not honest computational datatypes since e.g. they do not have a bottom element. There may be a way round this, but for now it seems simplest to leave out disjunction altogether.)

Note that if $\mathcal{L}$ is itself functional, then $\mathcal{L}' = \mathcal{L}$ and the relations $\approx_\sigma$ coincide with *observational equivalence*; the definition of typed realizability thus admits a slightly simpler reading in this case. Examples of this special case will be considered in Section 3; other examples involving non-functional languages will be considered in Section 4.

Having given untyped and typed realizability interpretations for $\mathbf{J}(\mathcal{L})$, it is natural to ask when they agree:

Definition 2.1 Let $\mathcal{L}$ be a language with functional core $\mathcal{L}'$ and A be a PCA such that $\mathbf{M}od(A)$ (with some choice of $N_\perp$) is a model for $\mathcal{L}'$. We say this model is *(constructively) logically fully abstract (LFA)* for $\mathcal{L}$ if, for all closed $\phi \in \mathbf{J}(\mathcal{L}')$, we have $A \models \phi$ iff $\mathcal{L} \models \phi$.

3 LFA models for functional languages

We now give some examples of LFA models for purely functional languages. The following result (partly folklore) describes a commonly occurring situation in which logical full abstraction holds.

Proposition 3.1 *Suppose $\mathcal{C}$ is a CCC giving a universal model for $\mathcal{L}$ (for some choice of object $N_\perp \in \mathcal{C}$), and suppose U is a universal object of $\mathcal{C}$. Let A be the combinatory algebra with underlying set* $\mathrm{Hom}(1, U)$ *obtained from some choice of retraction $U^U \lhd U$.*

(i) If $\mathcal{C}$ is well-pointed, then there is a full cartesian-closed embedding $I : \mathcal{C} \to \mathbf{M}od(A)$ into the projective *objects of $\mathbf{M}od(A)$.*

(ii) More generally, if $\mathcal{C}$ has a well-pointed cartesian-closed quotient $\mathcal{C}/\approx$, then there is a full cartesian-closed embedding $I : \mathcal{C}/\approx \to \mathbf{M}od(A)$.

In either case, the induced interpretation of $\mathcal{L}$ in $\mathbf{M}od(A)$ (with natural number domain $I(N_\perp)$) is constructively LFA.

In fact, in the above situation, the *modified* realizability interpretation of $\mathbf{J}(\mathcal{L})$ over A is also LFA. In addition, it seems likely that a large supply of LFA models can be obtained using the notion of *extensional* realizability (see [22]), though we have not yet explored this.

The above proposition represents a very pleasant situation and provides a cheap source of examples of LFA models; we will use it below to obtain LFA models of each of the three functional languages mentioned in the Introduction. There are also other LFA models not of this form, but one has to work harder to prove logical full abstraction. (Of course, this might suggest that the results thus obtained are correspondingly more interesting!)

3.1 PCF and its extensions

First we recall the definition of call-by-name PCF. We do this not because there is any shortage of definitions of PCF in the literature, but to provide a basis for some of the less familiar extensions to PCF that we shall define in the next section.

The types of PCF are the finite types defined above. For each type σ we have an infinite supply of variables of type σ, ranged over by $x^\sigma, y^\sigma, z^\sigma$. We also have the following collection of constants:

$$0,1,2,\ldots : \iota, \qquad \mathsf{if} : \iota \to \iota \to \iota \to \iota,$$

$$\mathsf{succ}, \mathsf{pred} : \iota \to \iota, \qquad \mathsf{Y}_\sigma : (\sigma \to \sigma) \to \sigma,$$

$$\mathsf{fst}_{\sigma\tau} : (\sigma \times \tau) \to \sigma, \quad \mathsf{snd}_{\sigma\tau} : (\sigma \times \tau) \to \tau.$$

The terms of PCF are built up from variables and constants as usual in the simply-typed λ-calculus:

- if $M : \tau$, then $(\lambda x^\sigma.M) : \sigma \to \tau$;
- if $M : \sigma$ and $N : \tau$, then $\langle M, N\rangle : \sigma \times \tau$;
- if $M : \sigma \to \tau$ and $N : \sigma$, then $(MN) : \tau$.

The *evaluation contexts* $E[-]$ of PCF are defined inductively as follows: the identity context $-$ is an evaluation context; and if $E[-]$ is an evaluation context then so are $\mathsf{succ}\,E[-]$, $\mathsf{pred}\,E[-]$, $\mathsf{if}\,E[-]$, $\mathsf{fst}_{\sigma\tau}E[-]$, $\mathsf{snd}_{\sigma\tau}E[-]$ and $E[-]N$ whenever these are well-typed. One then defines a one-step reduction relation $\to$ on closed terms of the same type inductively as follows (here n ranges over the *numerals* $0, 1, 2, \ldots$):

- $(\lambda x^\sigma.M)N \to M[N/x^\sigma]$;

- $\mathsf{succ}\, n \to (n+1)$, $\mathsf{pred}\,(n+1) \to n$, $\mathsf{pred}\, 0 \to 0$, $\mathsf{if}\, 0 \to (\lambda xy.x)$, $\mathsf{if}\,(n+1) \to (\lambda xy.y)$, $\mathsf{Y}_\sigma M \to M(\mathsf{Y}_\sigma M)$, $\mathsf{fst}_{\sigma\tau}\langle M, N\rangle \to M$, $\mathsf{snd}_{\sigma\tau}\langle M, N\rangle \to N$;
- if $M \to M'$ and $E[-]$ is an evaluation context such that $E[M]$ is well-typed, then $E[M] \to E[M']$.

We write $\to^*$ for the reflexive-transitive closure of $\to$. We say that a closed term $M : \iota$ *terminates* if $M \to^* n$ for some (necessarily unique) numeral n; in this case, we set $\mathrm{Eval}(M) = n$. If M does not terminate, then by convention we take $\mathrm{Eval}(M) = \bot$.

The language PCF^{++} is defined in the same way as PCF except that we include two additional constants

$$\mathsf{parallel\text{-}or} \;:\; \iota \to \iota \to \iota, \qquad \mathsf{exists} \;:\; (\iota \to \iota) \to \iota.$$

We will also consider the extension of PCF with a single constant

$$\mathsf{H} \;:\; ((\iota \to \iota) \to \iota) \;\to\; ((\iota \to \iota) \to \iota) \;\to\; \iota.$$

The above function Eval can be extended to yield an operationally defined evaluation relation for PCF^{++} [25], or for PCF+H [15], though we will not give the details here.

It is shown in [18] that any realizability model is a model of PCF provided it satisfies a *completeness axiom*, which appears to hold in most of the naturally occurring examples. Moreover, some natural realizability models are also models of PCF^{++} or of PCF+H (see below).

3.2 Examples of LFA models

We now give some examples of LFA models for each of our three languages.

- For PCF^{++}: Recall from [13] that the PCA K_1 (equipped with the non-termination set $\{n \mid n \cdot 0 \uparrow\}$) gives rise to a universal model of PCF^{++}. Let $\mathcal{C}$ be the full subcategory of $\mathbf{M}od(K_1)$ consisting of the retracts of the finite types. Then $U = 2_\bot^N$ is a universal object in $\mathcal{C}$ (by the "effective universality" of T^ω—see [26]), and the corresponding combinatory algebra A is exactly T^ω_{re}. Since we are in the situation of Proposition 3.1(i), the model $\mathbf{M}od(T^\omega_{re})$ is LFA for PCF^{++} (as is the corresponding modified realizability model).

  The PCA T^ω_{re} is closely related to the Scott graph model $\mathcal{P}\omega_{re}$. Interestingly, the standard realizability model on $\mathcal{P}\omega_{re}$ is not quite LFA for PCF^{++}: a counterexample (discussed in [13, page 263]) is the formula

  $$\forall x^\iota.\forall y^\iota.\ \neg(x\downarrow \wedge\, y\downarrow) \;\Rightarrow\; \exists n^\iota.(x\downarrow \Rightarrow n = 0) \wedge (y\downarrow \Rightarrow n = 1),$$

  which is realizable in $\mathcal{P}\omega_{re}$ but not in PCF^{++}. However, it appears that the *modified* realizability model over $\mathcal{P}\omega_{re}$ *is* LFA, although it is not an instance of Proposition 3.1.

  Note in passing that $\mathbf{M}od(K_1)$, although a universal model of PCF^{++}, comes nowhere near being LFA for PCF^{++}. For instance, Church's thesis

is realizable in K_1 but not in PCF^{++}:

$$\forall f^{\iota\to\iota}.\exists e^{\iota}.\forall n^{\iota}.\ \text{“}f(n) = e\cdot n\text{”}.$$

- For PCF+H: By analogy with the above, recall from [15] that the effective van Oosten algebra $\mathcal{B}_{re}$ gives rise to a universal model for PCF+H. Let $\mathcal{C}$ be the full subcategory of $\mathbf{M}od(\mathcal{B}_{re})$ consisting of retracts of finite types. It is shown in [15] that the object $U = N_\perp^{(N_\perp^N)}$ is universal in $\mathcal{C}$, and it gives rise to the combinatory algebra $\mathcal{B}_{2re}$. Again we are in the situation of Proposition 3.1(i), and so the standard and modified realizability models over $\mathcal{B}_{2re}$ are both LFA for PCF+H.

  However, neither the standard nor the modified realizability model over $\mathcal{B}_{re}$ is LFA for PCF+H.

- For PCF: The following construction has recently been given by Marz, Rohr and Streicher [19]. Let U be the canonical solution to the domain equation

$$U \;\cong\; U \oplus U \oplus (U \otimes U) \oplus (U \circ\!\!\rightarrow U)_\perp$$

  in a category $\mathcal{S}$ of *sequential domains* (a fully abstract model of PCF). Then all the PCF types (and also U^U), are syntactically definable retracts of U in the untyped λ-calculus $\mathcal{L}$ corresponding to the above domain equation. Let L_U be the PCA of definable elements of U (this is a term model for $\mathcal{L}$). By taking $\mathcal{C}$ to be the category of definable retracts of U and definable morphisms between them, we see by Proposition 3.1(i) that the realizability model over L_U is LFA for PCF. (In particular it is universal—this establishes a variant of the Longley-Phoa conjecture.)

  Similar results can be obtained by starting from a suitable intensional category $\mathcal{G}$ of games and innocent strategies. However, unlike $\mathcal{S}$, the category $\mathcal{G}$ is not well-pointed, so we are in the situation of Proposition 3.1(ii). The combinatory algebras thus obtained from $\mathcal{S}$ and $\mathcal{G}$ are very closely related: it seems likely that the former is a quotient of the latter.

  It is also plausible that the λ-term model Λ^0/T for any semi-sensible theory T yields an LFA model of PCF (this is a stronger claim than the Longley-Phoa conjecture). We have not yet considered whether any of Abramsky's recent constructions of combinatory algebras give LFA models for PCF.

3.3 A characterization of LFA models

In [17] a characterization of *classically* LFA realizability models was given: a realizability model $\mathbf{M}od(A)$ is LFA for $\mathcal{L}$ iff it is universal for $\mathcal{L}$. (This was proved in [17] for the languages PCF and PCF^{++}, but the same proof can be adapted to work for PCF+H.) We now give a simple characterization of constructively LFA models for these languages in a similar spirit.

Since all three of our languages $\mathcal{L}$ are functional, it is easy to see that all closed instances of the following schemata (the *axiom of choice* and the *independence of premiss* principle) are typed-realizable in each of them (for

any finite types σ, τ):

$$\text{AC:} \quad (\forall x^\sigma.\exists y^\tau.\phi[x,y]) \;\Rightarrow\; (\exists f^{\sigma\to\tau}.\forall x^\sigma.\phi[x,fx])$$

$$\text{IP:} \quad \forall x^\sigma.\, ((\neg\phi[x]) \Rightarrow \exists y^\tau.\psi[x,y]) \;\Rightarrow\; \exists y^\tau.((\neg\phi[x]) \Rightarrow \psi[x,y])$$

So in any PCA A which yields an LFA model of $\mathcal{L}$, these principles must be realizable. In addition, we have already seen that any LFA model must be classically LFA and hence universal. In fact, these conditions together suffice for logical full abstraction:

Theorem 3.2 *Let $\mathcal{L}$ be one of our three purely functional languages. A realizability model $(\mathbf{M}od(A), N_\perp)$ is constructively LFA for $\mathcal{L}$ iff it is a universal model for $\mathcal{L}$ and all closed instances of AC and IP are realizable in A.*

4 LFA models for non-functional languages

We now show how the notions of typed realizability and logical full abstraction can be extended to certain "impure" (i.e. non-functional) extensions of PCF. In doing so, we shall find a new use for some of the PCAs discarded above.

4.1 Conditions for logical full abstraction

We first give some general conditions which suffice for logical full abstraction. Intuitively, a model $(\mathbf{M}od(A), N_\perp)$ is LFA for a language $\mathcal{L}$ if the typed language $\mathcal{L}$ and the untyped structure A can be "simulated" sufficiently well in each other.

Firstly, define a *compilation* of $\mathcal{L}$ to A (w.r.t. $N_\perp$) to be a mapping γ from closed terms of $\mathcal{L}$ to elements of A such that

- $\gamma(MN) = \gamma(M) \cdot \gamma(N)$;
- if $M \in \mathcal{L}'^0_\iota$ then $\gamma(M)$ is a realizer for $\mathrm{Eval}(M)$ in $N_\perp$.

Secondly, if $(\mathbf{M}od(A), N_\perp)$ is a model of a functional language $\mathcal{L}'$, define a *simulation* of A in $\mathcal{L}'$ to consist of a type α, a mapping $\xi : A \to [\![\, \alpha \,]\!]$, and a term $\mathsf{apply} : \alpha \times \alpha \to \alpha$ of $\mathcal{L}'$ such that

- $\xi(a \cdot b) = [\![\, \mathsf{apply} \,]\!](\xi(a), \xi(b))$ whenever $a \cdot b$ is defined;
- there exist $u, v \in A$ such that for all $a \in A$, $u \cdot a \in \|\xi(a)\|$ and if $b \in \|\xi(a)\|$ then $v \cdot b = a$.

The following theorem now gives some sufficient conditions for logical full abstraction. It can be viewed as a generalization of Proposition 3.1.

Theorem 4.1 *Suppose $\mathcal{L}$ is a language with functional core $\mathcal{L}'$, $(\mathbf{M}od(A), N_\perp)$ is a realizability model for $\mathcal{L}'$, and moreover*

(i) *There is a compilation γ of $\mathcal{L}$ to A w.r.t. $N_\perp$.*

(ii) *There is a simulation $(\alpha, \xi, \mathsf{apply})$ of A in $\mathcal{L}'$.*

(iii) *For each type* σ *there is a term* $\mathsf{realizer}_\sigma : \sigma \to \alpha$ *of* $\mathcal{L}$ *such that for any* $M \in \mathcal{L}^0_\sigma$ *we have* $\mathsf{realizer}_\sigma M \in \mathcal{L}'$ *and* $[\![\, \mathsf{realizer}_\sigma M \,]\!] = \xi(\gamma(M))$.

Then $(\mathbf{M}od(A), N_\perp)$ *is logically fully abstract for* $\mathcal{L}$.

These conditions appear rather cumbersome, but they are very useful for establishing particular instances of logical full abstraction. We now present three examples of non-functional languages and corresponding LFA models for them.

*4.2 PCF+*quote

Firstly, we extend PCF with a Lisp-style quote operator. We define the language PCF+quote in the same way as PCF except that we include a family of constants $\mathsf{quote}_\sigma : \sigma \to \iota$. Evaluation contexts for PCF+quote are defined exactly as for PCF. We then take $\lceil - \rceil$ to be some effective Gödel-numbering of terms of PCF+quote, and include in the definition of one-step reduction all well-typed instances of

$$\mathsf{quote}_\sigma M \;\to\; \lceil M \rceil .$$

One might also consider adding Lisp-style eval operators with the property that $\mathsf{eval}_\sigma \lceil M \rceil \approx M$, but in fact there is no need: such operators can be defined in PCF+quote. (The construction is not trivial, but it is a simple adaptation of the construction of the PCF *enumerators* E^σ in [17].)

It is easy to see that the operators parallel-or and exists can be implemented in PCF+quote. More formally, there exist closed terms

$$\mathsf{parallel\text{-}or} \;:\; \iota \to \iota \to \iota, \qquad \mathsf{exists} \;:\; (\iota \to \iota) \to \iota$$

of PCF+quote inducing a translation of PCF^{++} into PCF+quote. commutes with evaluation for closed terms of ground type.

The functional core of PCF+quote is extensionally equivalent to PCF^{++}, which has an interpretation in $\mathbf{M}od(K_1)$ (with $N_\perp$ given as usual by the non-termination set $\{n \mid n \cdot 0 \uparrow\}$). Moreover, the three conditions of Theorem 4.1 are easily verified (note that the operations quote_σ give rise almost immediately to suitable terms $\mathsf{realizer}_\sigma$). Hence:

Theorem 4.2 *The model* $(\mathbf{M}od(K_1), N_\perp)$ *is LFA for* PCF+quote.

Thus, realizability over PCF+quote yields exactly the logic of finite types over $N_\perp$ in Hyland's *effective topos* [7].

*4.3 PCF+*timeout

Secondly, we consider (essentially) the language PCF+T recently introduced by Escardó in [6]. Our presentation of this language will be superficially different from that in [6], but it is easy to show that the two presentations are equivalent.

The idea is to add an operator $\mathsf{timeout}$ which will try to evaluate an expression of ground type for a prescribed length of "time". For simplicity, we will define the *time* taken to evaluate $P : \iota$ to be the number of recursion unfoldings (i.e. the number of reduction steps $\mathsf{Y}_\sigma M \to M(\mathsf{Y}_\sigma M)$) involved in the reduction of P (this will be finite if P terminates, and infinite otherwise). This appears to be a reasonable way to measure time, because the fragment of PCF without Y is normalizing, and so Y is in some sense the only thing that introduces the possibility of computations of arbitrary length. The operator $\mathsf{timeout} : \iota \to \iota \to \iota$ will then have the property that

$$\mathsf{timeout}\, P\, k \to^* 0 \qquad \text{if } P \text{ does not terminate within time } k;$$

$$\mathsf{timeout}\, P\, k \to^* n+1 \text{ if } P \text{ evaluates to } n \text{ within time } k.$$

Formally, the syntax of PCF+$\mathsf{timeout}$ is defined as for PCF but with the additional constant $\mathsf{timeout} : \iota \to \iota \to \iota$. The evaluation contexts of PCF+$\mathsf{timeout}$ are defined as for PCF with the additional clause: if $E[-] : \iota$ is an evaluation context then so are $\mathsf{timeout}\, P\, E[-]$ and $\mathsf{timeout}\, E[-]\, k$ (where k is a numeral). We now define a family of reduction relations $M \to^k M'$, with the intuitive meaning that "M reduces to M' in (exactly) time k". The relations $\to^k$ are defined simultaneously by induction as follows:

- $(\lambda x^\sigma.M)N \to^0 M[N/x^\sigma]$;
- $\mathsf{succ}\, n \to^0 (n+1)$, $\mathsf{pred}\,(n+1) \to^0 n$, $\mathsf{pred}\, 0 \to^0 0$, $\mathsf{if}\, 0\, NP \to^0 N$, $\mathsf{if}\,(n+1)NP \to^0 P$, $\mathsf{fst}_{\sigma\tau}\langle M, N\rangle \to^0 M$, $\mathsf{snd}_{\sigma\tau}\langle M, N\rangle \to^0 N$;
- $\mathsf{Y}_\sigma M \to^1 M(\mathsf{Y}_\sigma M)$;
- if $M \to^k n$ and $k \le l$, then $\mathsf{timeout}\, M\, l \to^k n+1$;
- if $M \to^k E[\mathsf{Y}_\sigma M']$ then $\mathsf{timeout}\, M\, k \to^k 0$;
- if $M \to^k M'$ and $E[-]$ is an evaluation context such that $E[M]$ is well-typed, then $E[M] \to^k E[M']$;
- if $M \to^k M'$ and $M' \to^l M''$, then $M \to^{k+l} M''$.

We now define $\to^*$ to be the reflexive closure of the union of the relations $\to^k$; the notion of termination and evaluation operation for PCF+$\mathsf{timeout}$ are then defined as usual. It is easy to see how a suitable operator $\mathsf{timeout}$ can be defined in PCF+quote, and so we have a translation from PCF+$\mathsf{timeout}$ to PCF+quote. Likewise, it is easy to see how we can use $\mathsf{timeout}$ to interleave computations and thus define $\mathsf{parallel\text{-}or}$ and exists operations, so we have a translation of PCF^{++} into PCF+$\mathsf{timeout}$. (Much less obviously, there is also a translation of PCF+H into PCF+$\mathsf{timeout}$!)

It turns out that PCF+$\mathsf{timeout}$ has an LFA model given by Kleene's second model K_{2re}. (The idea that this should be the case arose in a discussion with Martín Escardó.) Recall that this PCA has as its underlying set the set of *total* recursive functions $\mathrm{N} \to \mathrm{N}$, and application is defined as follows. Let $\langle -, - \rangle$ and $[\ldots]$ be effective codings for pairs and finite sequences of natural numbers respectively. Given $f, g \in K_2$ and $n \in \mathrm{N}$, let $j(f, g, n)$ be the unique

number j, if one exists, such that $f\langle n, [g(0), \ldots, g(i-1)]\rangle = 0$ for all $i < j$ but $f\langle n, [g(0), \ldots, g(j-1)]\rangle > 0$. (We think of 0 here as a request for more information about g, and any other number $m+1$ as signalling the result m. This is similar to the intuition behind the specification of the timeout operation.) We now define $f \cdot g$ to be the function $\lambda n.\, f(j(f,g,n)) - 1$ if this is total, and undefined otherwise. We also have an associated operation $| : K_{2re} \times K_{2re} \to \mathrm{N}_\perp$ given by $f \mid g = f(j(f,g,0)) - 1$ (taking this to be $\perp$ if $j(f,g,0)$ is undefined).

Some care is needed over the choice of $N_\perp$ in $\mathbf{M}od(K_{2re})$: the obvious choice arising from non-termination in the PCA is the wrong one, since termination in K_{2re} is not a Σ_1-predicate! The right choice of $N_\perp$ is defined by

$$\|m\| \;=\; \{f \mid f \mid \mathbf{0} = m\}, \qquad \|\perp\| \;=\; \{f \mid f \mid \mathbf{0} = \perp\}$$

where $\mathbf{0} = \lambda n.0$. If $f \mid \mathbf{0} = m$, we may think of $j(f, \mathbf{0}, 0)$ as a measure of the time taken for the computation of f to occur.

The functional core of PCF+timeout is again equivalent to PCF^{++}, which has an interpretation in $\mathbf{M}od(K_{2re}, N_\perp)$. (Proofs of these claims will appear in the full version of this paper.) Moreover:

Theorem 4.3 *The model* $(\mathbf{M}od(K_{2re}), N_\perp)$ *is LFA for* PCF+timeout.

Again this is proved by verifying the three conditions of Theorem 4.1. For condition 1, some care is needed to find a compilation which exactly preserves the "time" taken to evaluate ground type terms. Condition 2 is straightforward, using the type $\alpha = \iota \to \iota$. Condition 3 requires some crafty programming using timeout; the key lemma is the following special case:

Lemma 4.4 *There is a closed term* $\| : \|(\alpha \to \iota) \to \alpha$ *in* PCF+timeout *such that, for any closed term* $M : \alpha \to \iota$ *in the functional core of* PCF+timeout*,* $[\![\, \|M\| \,]\!]$ *corresponds to an element* $f \in K_{2re}$ *such that* $[\![\, M \,]\!] = \lambda g. f \mid g$.

*4.4 PCF+*catch

Finally, we consider a family of sequential programming languages which, in some sense, all embody the same computational power: PCF+catch [5,4], PCF+call/cc, μPCF [21], and a certain fragment of Standard ML admitting local uses of exceptions and references. It seems that these languages all admit good translations into each other, though we will not make this precise here (see [12] for an indication of the state-of-the-art). For simplicity, we will choose the language PCF+catch (essentially the language SPCF of [4] without errors) as representative of this family of languages, but we believe that the result below would apply equally well to any of them.

The syntax of PCF+catch is defined as for PCF but with additional constants

$$\mathsf{catch}_k : (\overbrace{\iota \to \cdots \to \iota}^{k} \to \iota) \to \iota$$

for $k \geq 0$. The evaluation contexts of PCF+catch are defined as for PCF with the following additional clause: if $E[-]$ is an evaluation context then so is $\mathsf{catch}_k(\lambda x_0 \ldots x_{m-1}.E[-])$ whenever $0 \leq m \leq k$. The one-step reduction relation is defined as for PCF with the following additional clauses:

- $\mathsf{catch}_k(\lambda x_0 \ldots x_{m-1}.E[x_i]) \rightarrow i$ whenever $E[-]$ is an evaluation context and x_i is free in $E[x_i]$;
- $\mathsf{catch}_k(\lambda x_0 \ldots x_{m-1}.n) \rightarrow m + n$;
- $\mathsf{catch}_1(\mathsf{succ}) \rightarrow 0$, $\mathsf{catch}_1(\mathsf{pred}) \rightarrow 0$, $\mathsf{catch}_3(\mathsf{if}) \rightarrow 0$.

It follows from the universality of PCF+catch for effective sequential algorithms (see [9]) that the functional H is definable in PCF+catch (see [15]). Thus we have a translation of PCF+H into PCF+catch. (Indeed, the functional core of PCF+catch is equivalent to PCF+H.) It is also easy to see that PCF+catch can be translated into PCF+quote.

A corresponding model is given by van Oosten's $\mathcal{B}_{re}$, with the evident choice of $N_\perp$ arising from the non-termination set $\{\lambda n.\perp\}$:

Theorem 4.5 *The model $(\mathbf{M}od(\mathcal{B}_{re}), N_\perp)$ is LFA for* PCF+catch.

Once again, the proof uses Theorem 4.1. For condition 1, the necessary compilation is given essentially by the interpretation of PCF+catch in sequential algorithms; condition 2 is easy; and condition 3 involves some cunning programming with catch. (The key lemma is analogous to Lemma 4.4 above.)

4.5 Summary

The situation we have described so far is summarized by Figure 1, which shows the six languages we have considered and the PCAs that give LFA models for them. The arrows here represent translations between the programming languages; it seems that no other translations are possible beyond those indicated. Note that not all these translations respect the functional core: e.g. the functional core of PCF+catch corresponds to PCF+H while that of PCF+quote corresponds to PCF^{++}. (This illustrates the non-functorial nature of the "extensional collapse" construction.)

Although here we have concentrated on the connections between particular languages and particular PCAs, we believe the translations are also of interest. We view the above picture as representing various notions of computability, ordered according to their computational power, or (if one prefers) their degree of intensionality. Indeed, it is no accident that for each of the above translations there is a corresponding *applicative morphism* between the respective PCAs (cf. [13]). We hope to study these translations more fully in a later paper.

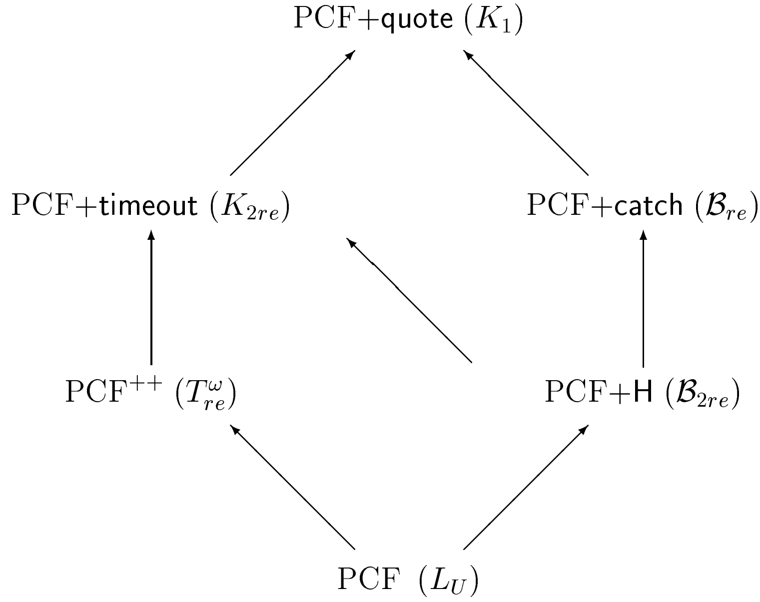


Fig. 1. Typed languages and their associated PCAs

5 Some logical examples

We have shown how both typed and untyped models of computation correspond to logical theories. These theories in some way capture the amount of computational power embodied by the models of computation. We now illustrate this with some particular examples of logical formulae, both to highlight the similarities and differences between our various notions of computability, and to demonstrate how logical formulae give a convenient way to summarize information about what is or is not computable in a certain setting. The two aspects of computability that seem to show up best are issues of *extensionality* (the difference between $\forall x.\exists y$ and $\exists f.\forall x$) and of *constructivity* (the difference between $\neg\neg\exists x$ and $\exists x$).

We begin with an assortment of simple examples, and then give some examples relating to exact real-number computability. We outline how, using our results, one can forge a link between real-number computability in various programming languages and real analysis inside various realizability toposes.

5.1 Simple examples

We have already mentioned a few examples of logical formulae: for instance, (certain instances of) the *axiom of choice* are realizable in all the purely functional languages but in none of the non-functional ones; and *Church's thesis* is realizable in K_1 (hence in PCF+quote) but in none of the other settings. We now mention some further examples:

5.1.1 Local moduli of continuity

Let us write approx for the PCF term

$$\lambda g^{\iota\to\iota}.\lambda n^{\iota}.(\lambda m^{\iota}.\, g(\mathsf{if}\,(m \leq n)\, m\, \Omega)$$

where $\leq$ is implemented as expected and Ω is some diverging term. Since all computable type 2 functions are continuous, it is realizable in all our settings that

$$\models \forall F^2.\forall g^1.\neg\neg\exists n^0.\, F(\mathsf{approx}\, g\, n) = Fg$$

(where 0 stands for the type ι, and $i+1$ stands for $i \to \iota$). Moreover, in PCF+H and all the languages above it in Figure 1, one can actually compute a suitable modulus of continuity n from F and g, so in these settings the formula

$$\models \forall F^2.\forall g^1.\exists n^0.\, F(\mathsf{approx}\, g\, n) = Fg$$

is realizable. However, it is easy to see by monotonicity that this latter formula is not realizable in PCF or PCF^{++}. Thus, this formula is internally true in four out of six of the corresponding realizability toposes.

In PCF+H and PCF+catch, we even have that

$$\models \exists \Phi^{2\to 1\to 0}.\forall F^2.\forall g^1.\, F(\mathsf{approx}\, g\, (\Phi F g)) = Fg.$$

However, this is not realizable in PCF+quote or PCF+timeout, since in these languages there is no *extensional* way to compute a modulus of continuity. (This is related to the fact that the interpretation of type 2 in these languages includes parallel functions.)

5.1.2 Uniform moduli of continuity

Classically, every continuous function from Cantor space 2^{N} to N is uniformly continuous: this is essentially König's Lemma. The corresponding result fails in all our effective settings, because the notorious *Kleene tree* yields functions that are continuous on the effective analogue of Cantor space but not uniformly continuous there (see e.g. [2]). However, given a function which classically is uniformly continuous, we can effectively obtain a modulus of uniform continuity. That is, if we write UnifMod (F^2, n^0) for the formula

$$\forall g^1 h^1.(\forall m^0.\,(m \leq n) \Rightarrow (gm < 2 \wedge gm = hm)) \Rightarrow (Fg\downarrow \wedge Fg = Fh)$$

then in all six of our settings we have

$$\models \forall F^2.\,(\neg\neg\exists n^0.\mathrm{UnifMod}(F,n)) \Rightarrow \exists n^0.\mathrm{UnifMod}(F,n).$$

In PCF+quote (and for that matter in PCF^{++} or PCF+timeout), a realizer can be easily constructed by means of a parallel search. In all our languages except PCF+quote, a realizer can be given using the remarkable Berger-Gandy definition of the fan functional in PCF (described e.g. in [27]), and so in fact we have the stronger formula:

$$\models \exists \Phi^3.\forall F^2.\,(\neg\neg\exists n^0.\mathrm{UnifMod}(F,n)) \Rightarrow \mathrm{UnifMod}(F, \Phi F).$$

However, this stronger version is *not* realizable in PCF+quote (at least with the above definition of UnifMod). Essentially this is because although we can

obtain a uniform modulus of continuity by a parallel search, we can never be sure that we have found the smallest possible modulus.

5.1.3 *Sequentiality indices*

In the languages PCF, PCF+H and PCF+catch, (but none of the others), every non-constant type 2 function has a sequentiality index, and so we have

$$\models \forall F^2. (\neg F(\lambda n^0.\Omega) \downarrow) \Rightarrow \neg\neg\exists n^0.\forall g^1. (Fg \downarrow) \Rightarrow (gn \downarrow).$$

Moreover, in PCF+catch (only), we can effectively compute a sequentiality index:

$$\models \forall F^2. (\neg F(\lambda n^0.\Omega) \downarrow) \Rightarrow \exists n^0.\forall g^1. (Fg \downarrow) \Rightarrow (gn \downarrow).$$

However, even in PCF+catch, there is no way to compute the sequentiality index extensionally in F, so the corresponding formula $\exists\Phi^3.\forall F^2.\ldots$ fails.

5.2 *Real-number computability*

Exact real-number computation provides an attractive application area for computation at higher types, so it is not surprising that the real numbers show up interesting differences between our various computational settings. This is an area of current joint research with Martín Escardó; we give here an informal sketch of some of our preliminary results.

Any standard realizability topos contains a *real number object* R (fortunately in such toposes the Cauchy and Dedekind reals always coincide). This means we can interpret formulae of real analysis (say in a language $\mathcal{R}$ involving the types R and $R \to R$) in the internal logic of any realizability topos. In general, different toposes will give rise to different flavours of real analysis, according to what formulae are true in them.

We can also represent real numbers using the finite types we have considered in this paper. The recursive reals (say in the interval $[-1, 1]$) can be represented exactly by recursive infinite sequences of extended binary digits $-1, 0, 1$; thus, arbitrary recursive reals can be represented by certain functions of type 1 (in any of our six languages $\mathcal{L}$). Computable functions on these reals can then be represented by functions of type $1 \to 1$ that behave extensionally on representations of reals.

It is easy to define predicates $\mathrm{Real}(x^1)$, $\mathrm{RealEq}(x^1, y^1)$, $\mathrm{RealFun}(f^{1\to1})$ and $\mathrm{RealFunEq}(f^{1\to1}, g^{1\to1})$, meaning (respectively) that x represents a real number, that x, y represent the same real number, that f represents an (extensional) total function on the recursive reals, and that f, g represent the same real function. Using these predicates, it is easy to see how one can define a translation from the logic $\mathcal{R}$ of the real number object to the logic $\mathbf{J}$ (which we may take to be $\mathbf{J}(\mathrm{PCF})$) in such a way that, in any of our models, a closed formula ϕ of $\mathcal{R}$ is true iff its translation $\hat{\phi}$ is. By logical full abstraction, it follows that ϕ holds internally in one of our toposes iff $\hat{\phi}$ is realizable in the corresponding typed programming language.

A simple example is given by the formula of $\mathcal{R}$ asserting that *all functions on the reals are continuous*. This beautiful result holds in many constructive settings, and is sometimes known as the Kreisel-Lacombe-Shoenfield (KLS) theorem (see e.g. [2]):

$$\models \forall f : R \to R. \forall x : R. \forall \epsilon > 0. \exists \delta > 0. \forall y \in R. \ |y - x| < \delta \Rightarrow |fy - fx| < \epsilon.$$

(We will feel free to sugar the syntax of $\mathcal{R}$ as long as the meaning is evident.) The constructive force of this is that given f, x and ϵ we can actually compute a δ which works. Not surprisingly in view of the above results on local moduli of continuity, the translation of the this formula is realizable in PCF+H and above, but not in PCF or PCF^{++}. This corresponds to the fact that the KLS theorem holds in the realizability toposes over $K_1, K_{2re}, \mathcal{B}_{re}$ and $\mathcal{B}_{2re}$, but not those over T^{ω}_{re} or L_U.

Unfortunately, many of the formulae of $\mathbf{J}$ that express interesting facts about real-number computability are not in the image of the translation from $\mathcal{R}$—that is, the language $\mathcal{R}$ seems to be not as expressive as we would like. In particular, in $\mathbf{J}$ we have the following useful formula UnifCts($f^{1\to 1}$), saying that a function f (representing, say, a function on $I = [0,1]$) is "uniformly continuous" in a sense analogous to that defined in Section 5.1.2:

$$\forall p^0. \exists n^0. \forall x^1 y^1. \, (\forall m^0. \, (m \leq n) \Rightarrow (xm < 3 \, \wedge \, xm = ym)) \Rightarrow$$
$$((fx)p \downarrow \wedge (fx)p = (fy)p)$$

This condition is stronger than the usual $\epsilon\delta$ definition of uniform continuity in real analysis, and is useful for excluding pathological functions with Kleene-tree-like behaviour. (Roughly speaking, it says that f would be total on the *classical* reals if we could apply it to them.) However, it seems that this property cannot be expressed in $\mathcal{R}$, since it is essentially a property of a *representation* $f^{1\to 1}$ of a real function rather than the real function itself. It would be pleasing if the above condition could be replaced by some reasonably clean mathematical condition involving the object R, but at present we do not know whether this is possible.

Meanwhile, let us add the predicate UnifCts($f^{1\to 1}$) to our language. (From now on, we will use a hybrid of $\mathcal{R}$ and $\mathbf{J}$ for our syntax, but officially we have in mind a corresponding formula of $\mathbf{J}$). By analogy with the results of Section 5.1.2, the following formula holds in all six of our settings:

$$\models \forall f : I \to R. \, (\neg\neg \mathrm{UnifCts}(f)) \Rightarrow \mathrm{UnifCts}(f)$$

There is an interesting class of formulae expressing the idea that (under various conditions) we can locate a zero of a function. One of the simplest examples is the following, which again holds in all our settings:

$$\models \forall f : I \to R. \, \mathrm{UnifCts}(f) \Rightarrow (\neg\neg \exists ! x : I. fx = 0) \Rightarrow (\exists ! x : I. fx = 0).$$

The hypothesis that the zero is unique is essential here. However, one can also consider similar formulae with other hypotheses, and here it seems that interesting distinctions emerge between the different notions of computability.

Finally, we mention some formulae expressing the idea that we can compute (Riemann) integrals for some class of functions. Again, the simplest such formula holds in all our settings:

$$\models \forall f : I \to R.\ \mathrm{UnifCts}(f) \Rightarrow \mathrm{Integrable}(f)$$

However, differences emerge when we try to integrate (partial) functions with discontinuities. For instance, the following formula says that we can integrate all functions that are undefined on at most one point, strictly between 0 and 1:

$$\models \forall f : I \rightharpoonup R.\ (\neg\neg\exists x \in (0,1).\, \forall y \in I.\, y \neg x => \mathrm{Real}(fx)) \Rightarrow \mathrm{Integrable}(f).$$

This formula is not realizable in PCF, but it is realizable in PCF+H. (The algorithm required is a simple adaptation of the integration algorithm described in [16].) In fact, for any k there is a formula asserting that all functions which are undefined on at most k points are integrable, and this is realizable in PCF+H. In PCF+catch one can do even better: we can integrate all functions that are undefined on only finitely many points without knowing a bound k in advance.

It would be interesting to undertake a more systematic investigation of these different flavours of real analysis, and perhaps for complex and functional analysis. It seems that there is a potentially very large research field here waiting to be explored!

Acknowledgement

I am grateful to Martín Escardó for many stimulating discussions, and especially for his input to the material in Sections 4.3 and 5.2. I also thank all the participants in the Domains IV workshop for their interest and encouragement. Andrej Bauer drew my attention to some over-enthusiastic claims involving uniform continuity on my Last Slide. The motivating philosophy behind this work owes much to the influence of Martin Hyland, and to my collaboration with Alex Simpson. This research was funded by the EPSRC Research Grant GR/L89532 "Notions of computability for general datatypes".

References

[1] Abramsky, S., R. Jagadeesan and P. Malacaria. Full abstraction for PCF. Accepted for publication, 1996.

[2] Beeson, M., "Foundations of Constructive Mathematics," Springer, 1985.

[3] Bucciarelli, A. and T. Ehrhard. *Sequentiality and strong stability,* in: *Proceedings* 6th Annual Symposium on Logic in Computer Science, IEEE, 1991, pages 138–145.

[4] Cartwright, R., P.-L. Curien and M. Felleisen, *Fully abstract semantics for observably sequential languages,* Information and Computation **111** (1994), pp. 297–401.

[5] Cartwright, R. and M. Felleisen, *Observable sequentiality and full abstraction,* In *Proc. 19th POPL*, ACM Press, 1992, pp. 328–342.

[6] Escardó, M. H., *A metric model of PCF,* Draft paper, March, 1999.

[7] Hyland, J. M. E., *The effective topos,* In *The L.E.J. Brouwer Centenary Symposium*, North-Holland, 1982.

[8] Hyland, J. M. E. and C.-H. L. Ong, *On full abstraction for PCF: I, II and III,* Accepted for publication, 1996.

[9] Kanneganti, R., R. Cartwright and M. Felleisen, *SPCF: its model, calculus, and computational power,* In *Proceedings* REX Workshop on Semantics and Concurrency, Lecture Notes in Computer Science **666** (1993), pp. 318–347.

[10] Kleene, S. C., *On the interpretation of intuitionistic number theory,* Journal of Symbolic Logic **10** (1945).

[11] Kleene, S. C. and R. E. Vesley, "The Foundations of Intuitionistic Mathematics," North-Holland, 1965.

[12] Laird, J., "A Semantic Analysis of Control," PhD thesis, University of Edinburgh, 1998, Examined March, 1999.

[13] Longley, J. R., "Realizability Toposes and Language Semantics," PhD thesis, University of Edinburgh, 1995, Available as ECS-LFCS-95-332.

[14] Longley, J. R., *Realizability models for sequential computation,* In preparation; an incomplete draft is available from the author's home page, 1998.

[15] Longley, J. R., *The sequentially realizable functionals,* Technical Report ECS-LFCS-98-402, Department of Computer Science, University of Edinburgh, 1998, submitted to Annals of Pure and Applied Logic.

[16] Longley, J. R., *When is a functional program not a functional program?,* Submitted to ICFP'99; available from the author's home page, 1999.

[17] Longley, J. R. and G. D. Plotkin, *Logical full abstraction and PCF,* In: J. Ginzburg et al., editors, Tbilisi Symposium on Language, Logic and Computation (1997), SiLLI/CSLI, pp. 333–352.

[18] Longley, J. R. and A. K. Simpson, *A uniform approach to domain theory in realizability models,* Mathematical Structures in Computer Science **7** (1997), pp. 469–505.

[19] Marz, M., A. Rohr and T. Streicher, *Full abstraction via realisability,* Accepted for LICS'99, 1999.

[20] Nickau, H., *Hereditarily sequential functionals,* in: *Proceedings* 3rd Symposium on Logical Foundations of Computer Science, Lecture Notes in Computer Science **813** (1994), pp. 253–264.

[21] Ong, C.-H. L. and C. A. Stewart, *A Curry-Howard foundation for functional computation with control,* in: *Proceedings* Symposium on Principles of Programming Languages, (1997), ACM Press, pp. 215–227.

[22] van Oosten, J., *Extensional realizability,* Technical Report ML-93-18, University of Amsterdam, ILLC, 1993.

[23] van Oosten, J., *A combinatory algebra for sequential functionals of finite type,* Technical Report 996, University of Utrecht, 1997. To appear in Proc. Logic Colloquium, Leeds.

[24] van Oosten, J., *The modified realizability topos,* Journal of Pure and Applied Algebra **116** (1997), pp. 273–289.

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

[26] Plotkin, G. D., *T^ω as a universal domain,* Journal of Computer and System Sciences **17** (1978), pp. 209–236.

[27] Plotkin, G. D., *Full abstraction, totality and PCF,* Accepted for publication, 1997.

[28] Scott, D. S., *Continuous lattices,* n: F.W. Lawvere, editor, *Toposes, Algebraic Geometry and Logic.* Lecture Notes in Mathematics **92** (1972).

[29] Scott, D. S., *Data types as lattices,* SIAM Journal of Computing **5** (1976), pp. 522–587.

[30] Scott, D. S., *A type-theoretical alternative to ISWIM, CUCH, OWHY,* Theoretical Computer Science **121** (1993), pp. 411–440. First written in 1969 and widely circulated in unpublished form since then.