Notions of computability at higher types II

John Longley

April 6, 2001

In Part I of this series of papers [Lon01a] we gave a historical survey of the study of notions of higher-type computability. In the present paper and its sequel [Lon01b], we undertake a more systematic exposition of notions of higher-type computability, showing how many of the existing ideas and results can be fitted into a coherent framework. In Part II we will restrict our attention to notions of computable *functional*—that is, the objects of type $\sigma \to \tau$ that we consider will always be (total or partial) *functions* from objects of type σ to objects of type τ. For such notions, the overall picture is easy to grasp with the help of some fairly concrete and naive definitions. A more general picture, embracing non-functional notions of computability and requiring slightly more sophisticated conceptual apparatus, will be presented in Part III.

In Section 1 we develop a general theory of type structures which, although simple, furnishes a useful framework for understanding particular notions of computable functional and their interrelationships. In Section 2 we survey the various known notions of *hereditarily total* computable functional within this framework, and in Section 3 the known notions of *hereditarily partial* computable functional. For the most part we consider the total and partial cases separately, and this is perhaps a limitation of our present treatment. In Section 4 we go some way towards rectifying this by considering how total and partial notions may be related or combined.

1 Type structures

1.1 Basic definitions

We first introduce some simple general theory to allow us to talk about notions of higher-type computable functional. The following definitions (with minor variations) appear frequently in the literature.

Definition 1.1 (Weak partial type structures) *A* weak partial type structure*, or* weak PTS *A [over a set X], consists of the following data:*

- *for each type σ, a set A_σ of* elements *of type σ [equipped with a canonical bijection $A_{\overline{0}} \cong X$],*
- *for each σ, τ, a partial* application *function $\cdot_{\sigma\tau} : A_{\sigma\to\tau} \times A_\sigma \rightharpoonup A_\tau$.*

We usually omit type subscripts from application operations, and often write $x \cdot y$ simply as xy. By convention, we take application to be left-associative.

Definition 1.2 (Partial type structures) *(i) A* partial type structure*, or* PTS *[over X] is a weak PTS A [over X] such that for all ρ, σ, τ there are elements $k_{\sigma\tau} \in A_{\sigma\to\tau\to\sigma}$ and $s_{\rho\sigma\tau} \in A_{(\rho\to\sigma\to\tau)\to(\rho\to\sigma)\to(\rho\to\tau)}$ satisfying*

$$kxy \;=\; x, \qquad sxy{\downarrow}, \qquad sxyz \simeq (xz)(yz)$$

for all x, y, z of appropriate types.

(ii) A (weak) PTS A is total*, or is a* (weak) type structure*, if each $\cdot_{\sigma\tau}$ is a total function.*

(iii) A (weak) PTS A is extensional *if for all σ, τ and all $f, g \in A_{\sigma\to\tau}$ we have*

$$(\forall x \in A_\sigma . fx \simeq gx) \implies f = g.$$

Extensional (total) type structures were introduced in [Hen50] and so are often called *Henkin models* (see e.g. [Mit96, Section 4.5]). Total type structures are called *combinatory type structures* in [Bar84, Appendix A], and weak total type structures are called *typed applicative structures* in [Mit96].

A weak PTS A is called a *partial type frame* (cf. [Mit96, Section 4.5]) if each $A_{\sigma\to\tau}$ is a set of partial functions from A_σ to A_τ, and each $\cdot_{\sigma\tau}$ is ordinary function application. It is easy to see that a weak PTS is extensional iff it is isomorphic to a partial type frame.

The significance of the conditions involving k, s in Definition 1.2 is shown by the following result, essentially due to Curry [Cur30]. Given a weak PTS A, let us write $E(A)$ for the set of well-typed *formal expressions* over A freely generated from constants (corresponding to elements of A) and typed variables via formal application. If e is a formal expression with variables among $\boldsymbol{x}$, and $\boldsymbol{a}$ is a vector of elements of A whose types match $\boldsymbol{x}$, we write $e[\boldsymbol{x} := \boldsymbol{a}]$ for the element of A denoted by e under the valuation $x_i \mapsto a_i$ (if defined). The following proposition shows that in a PTS, every function that is representable by a formal expression is also representable by an element of A itself.

Proposition 1.3 (Combinatory completeness) *A weak PTS A is a PTS iff for every formal expression $e \in E(A)$ with variables among $\boldsymbol{x}, y$, there is a formal expression $\lambda^* y.e \in E(A)$ with variables among $\boldsymbol{x}$, such that for all $\boldsymbol{a}, b \in A$ whose types match $\boldsymbol{x}, y$ we have*

$$(\lambda^* y.e)[\boldsymbol{x} := \boldsymbol{a}]\ {\downarrow}, \qquad ((\lambda^* y.e)[\boldsymbol{x} := \boldsymbol{a}])b \;\simeq\; e[\boldsymbol{x}, y := \boldsymbol{a}, b].$$

Throughout this section, our primary interest will be in *total* type structures. Type structures over N will represent notions of total computable functional, and type structures over $\mathrm{N}_\perp = \mathrm{N} \sqcup \{\perp\}$ will represent notions of partial computable functional. Partial type structures will mainly play an auxiliary role, as a means of constructing interesting total type structures. It may come as a surprise that we do not intend to use partial type structures as the primary means of representing notions of partial computability; however, we will argue below that any good notion of computable functional that can be embodied by a non-total type structure can equally well be embodied by a total type structure over $\mathrm{N}_\perp$. Moreover, our decision to focus on total type structures allows us to give a somewhat unified treatment of the total and partial cases.

The following examples show that it is easy to construct a plentiful supply of PTSs:

Examples 1.4 (i) Let $\mathcal{C}$ be any cartesian closed category. Let Y an object of $\mathcal{C}$; then we may define an interpretation $[\![-]\!]$ of the simple types by $[\![\overline{0}]\!] = Y$, $[\![\sigma \to \tau]\!] = [\![\tau]\!]^{[\![\sigma]\!]}$. Take $A_\sigma = \mathrm{Hom}(1, [\![\sigma]\!])$, and take $\cdot_{\sigma\tau}$ to be the function induced by the evaluation morphism $[\![\tau]\!]^{[\![\sigma]\!]} \times [\![\sigma]\!] \to [\![\tau]\!]$ in $\mathcal{C}$. This yields a total type structure A, which is extensional if $\mathcal{C}$ is well-pointed.

(ii) More or less the same construction can be understood syntactically in terms of typed lambda calculi. Let $\mathcal{L}$ be the simply-typed lambda calculus over a signature Σ containing a single ground type $\overline{0}$ and any desired collection of typed constants. Let T be any *theory* for $\mathcal{L}$: that is, a type-respecting equivalence relation on terms of $\mathcal{L}$ which is a congruence with respect to the term-forming operations, and contains α-equivalence and ordinary (call-by-name) β-equivalence: $(\lambda x.M)N \sim M[N/x]$). Take A_σ to be the set of closed $\mathcal{L}$-terms of type σ modulo T, with $\cdot_{\sigma\tau}$ induced by juxtaposition of terms. This yields a total type structure A, called the *closed term model for $\mathcal{L}$ modulo T*. As interesting special cases, one may take $\mathcal{L}$ to be Gödel's System T, or Plotkin's PCF (see 3 below), and T to be *observational equivalence*:

$$T \vdash M = N \text{ iff for all contexts } C[-],\ C[M] \to^* 0 \Longleftrightarrow C[N] \to^* 0.$$

In the former case we obtain a total type structure over N, since in System T every closed term of ground type evaluates to a numeral. In the latter case we obtain a total type structure over $\mathrm{N}_\perp$, since in PCF there are diverging terms of ground type but all such terms are observationally equivalent to one another.

One may obtain further variations on this theme by considering different evaluation strategies for the lambda calculus. We will discuss these in Section 1.2 below.

(iii) Recall that a *partial combinatory algebra (PCA)* consists of a set A equipped with a partial application operation $\cdot : A \times A \rightharpoonup A$ and containing elements k, s satisfying

$$kxy \;=\; x, \qquad sxy\!\downarrow, \qquad sxyz \simeq (xz)(yz)$$

for all $x, y, z \in A$. The leading example of a PCA is *Kleene's first model* K_1: here the underlying set is N, and $\cdot$ is the operation $(n, m) \mapsto \{n\}(m)$.

Clearly, one may regard any PCA A as a partial type structure simply by taking $A_\sigma = A$ for all σ, and $\cdot_{\sigma\tau} = \cdot$ for all σ, τ.

1.2 A more categorical view

Before proceeding further, we examine more closely the relationship between type structures and certain categories. This will give us a valuable additional perspective on the structures of interest, with some important conceptual advantages. However, since thereafter we will work primarily with the "applicative" definition of type structures, this paragraph may be skimmed by readers with little interest in category theory.

We first recall the following relaxation of the definition of cartesian closed category:

Definition 1.5 *A category $\mathcal{C}$ is* weakly cartesian closed*, or is a* weak CCC*, if $\mathcal{C}$ has finite products, and for any objects X, Y there are a chosen object Y^X and morphism $\epsilon_{XY} : Y^X \times X \to Y$, such that for any $f : Z \times X \to Y$ there is a morphism $\hat{f} : Z \to Y^X$ (not necessarily unique) such that $f = \epsilon_{XY} \circ (\hat{f} \times \mathrm{id}_X)$.*

For any category $\mathcal{C}$ with terminal object, we have an equivalence relation on each homset given by $f \sim g : X \to Y$ iff $\forall e : 1 \to X.\, fe = ge$. We thence obtain a quotient category $\mathcal{C}/\sim$ which we call $\mathcal{C}^\bullet$, *well-pointed collapse* of $\mathcal{C}$. Thus, $\mathcal{C}^\bullet$ is isomorphic to the concrete category constituted by the image of $\mathcal{C}$ under the functor $\mathrm{Hom}(1, -) : \mathcal{C} \to \mathbf{Set}$. It is easy to see that if $\mathcal{C}$ is a weak CCC then so is $\mathcal{C}^\bullet$. However, $\mathcal{C}^\bullet$ need not be an ordinary CCC when $\mathcal{C}$ is.

Evidently, Example 1.4(i) immediately generalizes to weak CCCs, so that any object Y in a weak CCC $\mathcal{C}$ gives rise to a total type structure $T_{\mathcal{C}}(Y)$. Clearly $T_{\mathcal{C}}(Y) \cong T_{\mathcal{C}^\bullet}(Y)$. (Note that for $\mathcal{C}$ a *weak* CCC, $T_{\mathcal{C}}(Y)$ need not be extensional when $\mathcal{C}$ is well-pointed.)

Somewhat conversely, given any total type structure A we may build a category $\mathcal{C}(A)$ as follows. Objects of $\mathcal{C}(A)$ are finite tuples $\langle \sigma_1, \ldots, \sigma_s \rangle$ of types. Morphisms $\langle \sigma_1, \ldots, \sigma_s \rangle \to \langle \tau_1, \ldots, \tau_t \rangle$ are functions $f : A_{\sigma_1} \times \cdots \times A_{\sigma_s} \to A_{\tau_1} \times \cdots \times A_{\tau_t}$ such that each $f_i = \pi_i f$ is *representable* in A, in the sense that there is an element $r_i \in A_{\sigma_1 \to \cdots \to \sigma_s \to \tau_i}$ such that for all $a_1 \in A_{\sigma_1}, \ldots, a_s \in A_{\sigma_s}$ we have $r_i a_1 \cdots a_s = f_i(a_1, \ldots, a_s)$. It is easy to check that if A is a type structure then $\mathcal{C}(A)$ is canonically a well-pointed weak CCC, and that if A is extensional then $\mathcal{C}(A$ is an ordinary CCC.

By abuse of notation we write $\overline{0}$ for the object $\langle \overline{0} \rangle$ of $\mathcal{C}(A)$. It is easy to see that for any type structure A we have $T_{\mathcal{C}(A)}(\overline{0}) \cong A$. Conversely, for any weak CCC $\mathcal{C}$ and object Y of $\mathcal{C}$, $\mathcal{C}(T_{\mathcal{C}}(Y))$ is isomorphic to a full subcategory of $\mathcal{C}$ whose objects are generated from Y via finite products and exponentials.

We may therefore identify total type structures with well-pointed weak CCCs whose objects are generated from a single "ground type" object. To some extent it is a matter of taste which of these we work with, and both have their advantages. We have opted for the definition of type structure, in which application rather than composition is taken as fundamental, since, in particular, the notion of *morphism* between models which we wish to consider (see Section 1.5) is much more naturally phrased in these terms. However, the categorical view has the significant conceptual advantage that it leads to a much more "presentation-independent" characterization of notions of computability, as we shall now explain.

In any category, an object Y is said to be a *retract* of X if there are morphisms $e : Y \to X$ and $d : X \to Y$ such that $de = \mathrm{id}_Y$. We may think of this as saying that Y may be embedded in X (or, in terms of datatypes, that elements of Y may be represented by elements of X) in such a way that morphisms into and out of Y may be adequately represented by morphisms into and out of X. Note that the morphism $f = ed : X \to X$ is idempotent (that is, $f^2 = f$), and that this idempotent determines Y up to isomorphism. In general we say an idempotent $f : X \to X$ *splits* if there exist an object Y and morphisms d, e such that $de = \mathrm{id}_Y$ and $ed = f$.

Thinking set-theoretically, we can imagine that any idempotent f on X

implicitly determines a "subset" of X, namely $\{x \in X \mid x = f(x)\}$. Even if there is no object Y corresponding to this in our world of datatypes, we can imagine that such a datatype could be added for no extra cost, since f completely determines how the elements of such a datatype would behave. This motivates the following well-known construction for formally splitting all idempotents in a category:

Definition 1.6 *Given any category $\mathcal{C}$, its* category of retracts *or* Karoubi envelope *$K(\mathcal{C})$ is constructed as follows: objects are pairs (X, f), where f is an idempotent on X in $\mathcal{C}$; and morphisms $(X, f) \to (Y, g)$ are morphisms $h : X \to Y$ in $\mathcal{C}$ such that $h = ghf$.*

Note that $\mathcal{C}$ embeds fully and faithfully in $K(\mathcal{C})$, and that in $K(\mathcal{C})$ every idempotent splits. One should therefore think of $K(-)$ as a "completion" process whereby we throw in all datatypes that are trivially representable in terms of already existing datatypes. In particular, if we are given a notion of computable function between all the types in $\mathcal{C}$, this uniquely determines a notion of computable function between all the types in $K(\mathcal{C})$.

Let us now see how this applies to categories built from type structures. In fact, we can start from an even simpler category than the weak CCC $\mathcal{C}(A)$ defined above: given a type structure A, let $\mathcal{P}(A)$ be the category whose objects are simple types σ, and whose morphisms $\sigma \to \tau$ are total functions $A_\sigma \to A_\tau$ representable by an element of $A_{\sigma\to\tau}$. Now consider the category of retracts $K(\mathcal{P}(A))$. Under mild conditions on A to the effect that that certain boring "basic operations" are representable, we may show (with a little effort) how products of simple types arise as retracts of other simple types, whence $K(\mathcal{P}(A)) \equiv K(\mathcal{C}(A))$. (Being even more parsimonious, we could start from the category whose objects are just the *pure* types, since under mild conditions all simple types arise as retracts of pure types.) The following proposition summarizes some of the important properties of $K(\mathcal{P}(A))$ in the cases of interest:

Proposition 1.7 *Let A be a type structure over N or $\mathrm{N}_\perp$, containing an element ifzero $\in A_{\overline{0}\to\overline{0}\to\overline{0}\to\overline{0}}$ with the following properties:*

ifzero $0\ x\ y = x$, *ifzero* $(n{+}1)\ x\ y = y$, *(for A over $\mathrm{N}_\perp$) ifzero* $\perp\ x\ y = \perp$.

Then $K(\mathcal{P}(A))$ is a well-pointed weak CCC, which is an ordinary CCC iff A is extensional. Moreover, if A is over $\mathrm{N}_\perp$, then $K(\mathcal{P}(A))$ is equipped with a lifting monad *$((-)_\perp, \eta, \mu)$, such that for any object X there is just one global element $1 \to X_\perp$ that does not factor through η_X.*

With a little more effort it can also be shown that, under the same hypotheses, $K(\mathcal{P}(A))$ is closed under the formation of (positive) *inductive types* and of *dependent types*. Thus, the meagre skeleton of $\mathcal{P}(A)$ is fleshed out by this construction to yield an abundant world of datatypes rich enough to support many interesting type-theoretic constructs.

It is clear that in various ways the category $K(\mathcal{P}(A))$ (considered as given up to equivalence of categories) gives us a more presentation-independent way

of embodying a notion of computable operation. Firstly, it is not sensitive to inessential details in our definition of type structure, such as whether we restrict attention to pure types, whether we include product types, whether we give ourselves a separate ground type for the booleans, or even whether we work with a much richer language of types including inductive and dependent types. Superficial differences such as these are washed out when we pass to the category of retracts; thus, giving a type structure under one definition of this kind is precisely equivalent to giving a type structure under any other such definition.

Another way in which $K(\mathcal{P}(A))$ is independent of presentation details has to do specifically with *partial* functions or operations. We have declared that, for us, notions of partial computable functional will be represented by total type structures over $\mathrm{N}_\perp$. But this seems arbitrary: why not partial type structures over N, or even partial type structures over $\mathrm{N}_\perp$? Again, the categorical perspective shows us that these three possibilities are in fact equivalent: thus, any total type structure over $\mathrm{N}_\perp$ is equivalent to some partial type structure over N (in the sense that they yield the same category of retracts), and similarly for the other combinations.

To make this precise, let us adopt the following definitions which demarcate the classes of structures we are talking about:

- A *standard TTS over* $\mathrm{N}_\perp$, or *call-by-name type structure*, is an extensional TTS over $\mathrm{N}_\perp$ containing an element *ifzero* satisfying the conditions in Proposition 1.7.
- A *standard PTS over* N, or *call-by-value type structure*, is an extensional PTS over N containing an element *ifzero* satisfying the first two conditions of Proposition 1.7, and an element *eval* of type $\overline{0} \to \overline{0}$ such that $\mathit{eval}\, n = n$ and $\mathit{eval} \perp \uparrow$.
- A *standard PTS over* $\mathrm{N}_\perp$, or *lazy type structure*, is an extensional PTS over $\mathrm{N}_\perp$ containing an element *ifzero* satisfying the conditions in Proposition 1.7, and an element *eval* with the above properties.

Furthermore, let us call a category $\mathcal{C}$ a *standard category for partial functionals* if it is well-pointed, cartesian closed, contains splittings for all idempotents, contains a distinguished object $N_\perp$ such that all objects are retracts of

- $\mathcal{C}$ is a well-pointed CCC in which all idempotents split;
- $\mathcal{C}$ contains a distinguished object $N_\perp$, equipped with a bijection $\mathrm{Hom}(1, N_\perp) \cong \mathrm{N}_\perp$, such that all objects in $\mathcal{C}$ are retracts of powers of $N_\perp$.
- $\mathcal{C}$ contains a morphism $\mathit{Ifzero} : N_\perp \times N_\perp \times N_\perp \to N_\perp$ whose action on global elements is given (modulo the above bijection) by

$$\mathit{Ifzero}\langle 0, x, y\rangle \;=\; x, \qquad \mathit{Ifzero}\langle n+1, x, y\rangle \;=\; y, \qquad \mathit{Ifzero}\langle \perp, x, y\rangle \;=\; \perp.$$

One can show that any standard category for partial functionals may be endowed with a lifting monad $((-)_\perp, \eta, \mu)$ with the property mentioned in Proposition 1.7 (one takes $X_\perp$ to be a certain retract of $N_\perp \times X$).

Theorem 1.8 *There are canonical bijective correspondences between the following classes of structures:*

- *Standard TTSs over* $\mathrm{N}_\perp$*, up to isomorphism.*
- *Standard PTSs over* N*, up to isomorphism.*
- *Standard PTSs over* $\mathrm{N}_\perp$*, up to isomorphism.*
- *Standard categories for partial functionals, up to equivalence of categories.*

Remark 1.9 The alternative terminology of call-by-name, call-by-value and lazy type structures arises from a connection with the evaluation strategies for lambda calculi and other programming languages known to computer scientists under these names. For instance, the language PCF has been considered in three flavours: the original call-by-name version [Plo77], a call-by-value version [SF90], and a lazy version [BR89]. Term models for these languages naturally give rise to call-by-name, call-by-value and lazy type structures respectively.

The general form of call-by-name lambda calculi was more or less indicated in Example 1.4(ii) above. We will not explicitly define the call-by-value and lazy variants here, since the only purpose in doing so would be to point out that they are equivalent for our present purposes. In the case of PCF, the relationship between the three variants has been studied in [Sie90, Rie93], and more fully in [Lon95, Chapter 6], where a syntactic equivalence result comparable with Theorem 1.8 is proved in detail.

The view we are advocating here is that, for any notion of computable functional, the corresponding "standard category" is the real underlying object of interest, and different definitions of type structure merely represent different windows onto this object.[1] From this point of view, one might even say that such a category *is* the mathematical incarnation of the corresponding notion of computable functional, in something like the way in which the classifying model for a logical theory is the incarnation of the theory itself.

As usual, this point of view cuts both ways. On the one hand, we are really more interested in the underlying object than in any particular presentation; on the other hand, since the Karoubi envelope construction is obviously trivial, all the interesting information about the standard category is clearly present in the corresponding type structure, so we may just as well study the type structure if it is technically more convenient to do so. We shall henceforth work with type structures on the understanding that they are really just economical concrete presentations of underlying notions of computability of which the categorical models are conceptually a more faithful embodiment.

[1] Our notion of standard category for partial functionals is not quite as canonical as we have made out, since there are other ways of building a categorical model from a class of partial functions. Intuitively, the notion we have used is analogous to the category of pointed sets and total functions, but one might also consider the analogues of ordinary sets and partial functions, or pointed sets and partial functions. However, the relationships between these categories are well understood and each can be recovered from the others. Moreover, the "conceptual distance" between them is intuitively much smaller than that between the different notions of type structure, so that any of these categories would serve equally well in the role of underlying mathematical object.

1.3 Effective type structures

Clearly, type structures as defined above may consist of computable or non-computable objects. To formalize the idea of a type structure consisting entirely of computable objects, we propose the following definition.

Definition 1.10 (Effective type structures) *(i) An* effective structure $\Vdash$ *on a (weak) PTS A is a family of relations $\Vdash_\sigma \subseteq \mathrm{N} \times A_\sigma$ satisfying $\forall x \in A_\sigma . \exists n \in \mathrm{N}. n \Vdash_\sigma A$, such that for all σ, τ the application $\cdot_{\sigma\tau}$ is "tracked" by a binary partial recursive function $\phi_{\sigma\tau}$, i.e.*

$$\forall f \in A_{\sigma\to\tau}, x \in A_\sigma, n, m \in \mathrm{N}.\ n \Vdash f \wedge m \Vdash x \wedge fx\downarrow \implies \phi_{\sigma\tau}(n,m) \Vdash fx.$$

An effective (weak) PTS *is a (weak) PTS equipped with an effective structure.*

(ii) Two effective structures $\Vdash$, $\Vdash'$ on A are effectively equivalent *if for each σ there are unary partial recursive functions ζ_σ, ξ_σ such that*

$$\forall x \in A_\sigma, n \in \mathrm{N}.\ (n \Vdash_\sigma x \implies \zeta_\sigma(n) \Vdash'_\sigma x) \ \wedge\ (n \Vdash'_\sigma x \implies \xi_\sigma(n) \Vdash_\sigma x).$$

The idea implicit here is that any reasonable notion of "computable" object can ultimately be encoded by natural numbers, with computable operations corresponding to partial recursive functions. One can of course generalize the above definition of effectivity in many ways—for instance, one can relativize to an arbitrary PCA in place of N—but there is an intuition that this would correspond to a genuine notion of effectivity only if the PCA in turn admitted some encoding in N. (These ideas will be explored on a technical level in Part III.) This seems to raise an interesting philosophical question: *why* do we believe (if we do) that all reasonable forms of "computation" are ultimately representable in terms of natural numbers? Put another way, why do we think natural numbers are the ultimate atoms of information? We do not intend to discuss this question here, but merely to draw attention to a presupposition underlying our definitions.

In the case of type structures over N or $\mathrm{N}_\perp$, we typically want to restrict attention to effective structures that behave reasonably at ground type:

Definition 1.11 *(i) An* effective (weak) PTS over N *is a (weak) PTS A over* N *equipped with an effective structure $\Vdash$ such that $n \Vdash_{\overline{0}} m$ iff $n = m$.*

(ii) An effective (weak) PTS over $\mathrm{N}_\perp$ *is a (weak) PTS A over $\mathrm{N}_\perp$ for which there exists a unary partial recursive function θ such that $n \Vdash_{\overline{0}} m$ iff $\theta(n) = m$, and $n \Vdash_{\overline{0}} \perp$ iff $\theta(n)$ is undefined.*

Note in passing that, in terms of the correspondences given by Theorem 1.8, a standard TTS over $\mathrm{N}_\perp$ is effective iff the corresponding standard PTS over N [resp. standard PTS over $\mathrm{N}_\perp$] is effective.

The above definitions allow us to talk about type structures consisting entirely of computable objects acting on other computable objects. However, we will also be interested in classes of computable functions whose domain of definition includes non-computable functions. We may take these into account by means of an obvious notion of substructure:

Definition 1.12 (Substructure) *Suppose A, B are weak PTSs. We say B is a* substructure *of A if*

- *$B_\sigma \subseteq A_\sigma$ for each σ, and*
- *whenever $f \in B_{\sigma\to\tau}$ and $x \in B_\sigma$, we have $f \cdot^A_{\sigma\tau} x \simeq f \cdot^B_{\sigma\tau} x$.*

Our thesis is that any reasonable notion of higher-type computable functional will be embodied by an effective PTS B occurring as a substructure of a (possibly non-effective) extensional PTS A. Here A gives the domains of the computable functionals while B gives the computable functionals themselves (it may happen that $B = A$). This thesis rests on various assumptions, e.g. we expect appropriate k and s combinators to be computable, and we expect that if f and x are computable objects then so is $f \cdot x$ (if defined). The condition that A is extensional reflects the idea that our higher-type objects are functions. Notice that it does not follow that B is extensional and there seems to be no obvious reason to insist this, though it will indeed be true for most of the important examples.

Note that, under the correspondence given by Theorem 1.8, effective substructures of call-by-name structures correspond precisely to effective substructures of call-by-value or lazy structures (in fact, all of these correspond to "effectively presented" subcategories of standard categories for partial functionals in a certain sense). Thus, our view of what counts as a notion of higher-type computable functional in the partial case is not prejudiced by our decision to focus on call-by-name structures.

1.4 Constructions on type structures

Next we mention some general constructions for obtaining new PTSs from old ones. These constructions will recur frequently when we come to consider concrete examples.

The first construction gives a canonical way of picking out total substructures from an arbitrary weak PTS.

Definition 1.13 (Total hull) *Given a weak PTS A and a subset $Z \subseteq A_{\overline{0}}$, one may define a substructure B of A as follows:*

$$B_{\overline{0}} \;=\; Z, \qquad B_{\sigma\to\tau} = \{f \in A_{\sigma\to\tau} \mid \forall x \in B_\sigma.\, fx{\downarrow} \wedge fx \in B_\tau\}.$$

We call B (with the application operations inherited from A) the total hull *of A with respect to Z. If Z is the whole of $A_{\overline{0}}$, we sometimes call B simply the total hull of A.*

Essentially, the total hull of A w.r.t. Z just corresponds to the unary *logical relation* on A determined by the subset Z (see [Sta85]). Clearly the total hull of A w.r.t. Z is always a weak total type structure; moreover, if A is a PTS then the total hull is a total type structure, since it necessarily contains the k and s combinators from A.

It is easy to see that the total hull of A [w.r.t. Z] is *maximal* among total substructures B of A [such that $B_{\overline{0}} = Z$]. However, we emphasize that in

general it need not be the *unique* maximal such substructure: there may be total substructures of A [over Z] that are not contained in the total hull. For example... [Refer to naturally arising examples later in paper...] Intuitively, one may think of Definition 1.13 as giving a "short-sighted" or "greedy" strategy for picking out a total substructure of A, in the sense that at each type level we take as many elements as possible to be in the substructure. Other total substructures may arise since, as it were, by sacrificing some total elements at one type level one might allow oneself more, or other, total elements at some higher type level.

The second construction gives a canonical way of turning a weak PTS into an extensional one. Recall that a *partial equivalence relation* (or PER) on a set S is simply a symmetric, transitive relation E on S—that is, an equivalence relation on a subset of S. We write S/E for the set of equivalence classes of E, and $[x]_E$ for the equivalence class (if any) containing the element x.

Definition 1.14 (Extensional collapse) *(i) Given a weak PTS A and a partial equivalence relation $\equiv$ on $A_{\overline{0}}$, we may define a PER E_σ on each A_σ by*

$$xE_{\overline{0}}y \iff x \equiv y, \qquad fE_{\sigma\to\tau}g \iff (\forall x, y.\, xE_\sigma y \Longrightarrow fx\, E_\tau\, gy).$$

Now let C be the substructure of A determined by $C_\sigma = \{x \in A_\sigma \mid xE_\sigma x\}$. We call C the pre-extensional hull *of A w.r.t. $\equiv$; we say A is* pre-extensional *w.r.t. $\equiv$ if $C = A$. If $\equiv$ is just the equality relation on $A_{\overline{0}}$, we may call C simply the pre-extensional hull of A, and say A is pre-extensional if $C = A$.*
(ii) In the above situation, we may obtain a total extensional weak type structure D by defining $D_\sigma = C_\sigma/E_\sigma$, with application induced by C. We call D the extensional collapse*, or* Gandy hull*, of C (or of A w.r.t. $\equiv$).*

The pre-extensional hull of A w.r.t. $\equiv$ is thus given by the binary logical relation on A determined by $\equiv$. In the above definition, if A is a PTS then so are C and D. Once again, the pre-extensional hull of A w.r.t. $\equiv$ is a maximal pre-extensional substructure of A w.r.t. $\equiv$, though not necessarily the unique such. [REFER TO a natural counterexample...] Thus, our remarks above on the total hull construction apply equally well to the extensional collapse construction: it defines a greedy strategy for obtaining one particular extensional type structure from a given type structure.

Note that an effective structure on A induces an effective structure on any total hull, pre-extensional hull or extensional collapse of A in an obvious way.

The above two constructions may be combined as follows: given a weak PTS A, a subset $Z \subseteq A_{\overline{0}}$ and a PER $\equiv$ on Z, we may take B the total hull of A w.r.t. Z, and D the extensional collapse of B w.r.t. $\equiv$. We will refer to D as the *modified extensional collapse* of A w.r.t. Z and $\equiv$. If no set Z is explicitly mentioned, it is assumed to be $\{x \mid x \equiv x\}$. If neither Z nor $\equiv$ is explicitly mentioned, it is assumed that $Z = A_{\overline{0}}$ and $\equiv$ is equality on $A_{\overline{0}}$.

Remark 1.15 Clearly, if A is total and $Z = A_{\overline{0}}$, then the total hull of A w.r.t. Z is clearly A itself. Hence, in this case, for any PER $\equiv$ on $A_{\overline{0}}$, the extensional and modified extensional collapses of A w.r.t. Z and $\equiv$ coincide. However, there

is no reason why more generally the two collapses should coincide (it is easy to fabricate artificial counterexamples), though interesting this does seem to hold for almost all the natural examples, sometimes for very non-trivial reasons (see for example [REFER TO BEZEM EXAMPLE]).

The following examples show how these collapse constructions arise naturally in the setting of some well known categories. A more general formulation of the same idea will be presented in Part III.

Examples 1.16 (PERs on partial combinatory algebras) (i) For any partial combinatory algebra A, the category $\mathbf{PER}(A)$ is defined as follows: objects are PERs on A, and morphisms $E \to E'$ are functions $f : A/E \to A/E'$ that are tracked by some element $r \in A$ (that is, for all x with xEx we have $f([x]_E) = [rx]_{E'}$). The category $\mathbf{PER}(A)$ enjoys many good properties (see e.g. [Lon95, Chapter 1]). In particular it is cartesian closed: the exponential E'^E is defined by $r(E'^E)r'$ iff r, r' both track the same morphism $f : E \to E'$.

Clearly, given any PER $\equiv$ on A, the type structure over $\equiv$ in $\mathbf{PER}(A)$ is precisely the extensional collapse of A (viewed as a PTS) w.r.t. $\equiv$. It follows that if $\equiv, \equiv'$ are isomorphic objects in $\mathbf{PER}(A)$ then the extensional collapses of A w.r.t. $\equiv$ and $\equiv'$ are isomorphic.

We will most often be interested in type structures over PERs that play the role of N of $N_\perp$. In particular, in any PCA A there is a standard coding (due to Curry) of the natural numbers as elements of A, defined by

$$\bar{0} \;=\; i, \qquad n \dot{+} 1 \;=\; s(si(k(ki)))(k\bar{n}).$$

Let N be the PER on A given by all instances of $\bar{n}N\bar{n}$. It can be shown that N is a *natural number object* in $\mathbf{PER}(A)$. (Thus, the details of the above coding are not essential, since if N' is any other natural number object in $\mathbf{PER}(A)$ then $N' \cong N$ and so the type structures over N and N' coincide.)

No similar uniform method is known for constructing a suitable PER $N_\perp$ over an arbitrary PCA. However, in many individual cases, a suitable object may be obtained from N by applying a *lifting construction* specific to the PCA in question. We will see many examples of this below.

(ii) Likewise, the category $\mathbf{MPER}(A)$ of *modified PERs* on a PCA A is defined as follows: objects a pairs (P, E) where $P \subseteq A$ and E is a PER on P; and morphisms $(P, E) \to (P', E')$ are functions $f : A/E \to A/E'$ that are tracked by some $r \in A$ in the following strong sense: whenever xEx we have $f([x]_E) = [rx]_{E'}$, and for all $x \in P$ we have $rx \in P'$. Once again, the category $\mathbf{MPER}(A)$ is cartesian closed: the exponential $(P', E')^{(P,E)}$ is the pair (Q, F) defined by $Q = \{y \mid \forall x \in P.yx \in P'\}$, rFr' iff r, r' track the same morphism $f : (P, E) \to (P', E')$ in the strong sense.

Clearly, given any object $(Z, \equiv)$ of $\mathbf{MPER}(A)$, the type structure over $(Z, \equiv)$ in $\mathbf{MPER}(A)$ is precisely the modified extensional collapse of A w.r.t. Z and $\equiv$.[2]

[2]As we will see in Part III, the categories defined here are important in connection with realizability interpretations of logic: $\mathbf{PER}(A)$ provides a model-theoretic setting for Kleene-

The constructions we have described seem to exhaust the possible *general* ways of picking out interesting substructures or subquotients from given structures. However, we wish to stress the idea that in *particular* cases these are often not the only interesting substructures or subquotients to be considered. This possibility sometimes seems to be neglected: for instance, people sometimes refer loosely to "the extensional part" of a certain type structure, when they really mean just the particular substructure picked out by the extensional collapse construction.

An important question to ask in this connection is: Is there anything special or distinguished about the extensional collapse among subquotients of a given type structure, beyond the fact that it *is* the extensional collapse (or, equivalently, that it corresponds to the type structure arising from the cartesian closed category of PERs)?

[LEVELWISE MAXIMALITY as a unifying concept? Stability/robustness of total hull and extensional collapse? E.g. general theorem saying cbn, cbv extensional collapses always agree. For this we might want to generalize Theorem 1.8 to non-extensional structures...]

[Sufficient conditions for when two TSs have the same ext collapse: see Part III.]

1.5 Morphisms between type structures

It is natural to ask how different PTSs are related. In particular, we would like a notion of *morphism* $A \to B$ between PTSs that somehow corresponds to a way of simulating A within B; this would allow us to say that the notion of computability embodied by B was in some sense at least as powerful as that embodied by A. For our purposes, we would prefer our notion of morphism to be as general as possible, both so as to be able to embrace as many different kinds of relationships between PTSs as possible within a unified framework, and so as to be able to state certain results in as strong a form as possible (see REF. ANTI CHURCH'S THESES).

There are several possible candidates for a definition of morphism $A \to B$. For instance, we might consider

- *algebraic homomorphisms*: that is, families of functions $A_\sigma \to B_\sigma$ commuting with the application operations; or
- *logical relations*: that is, families of total relations $R_\sigma : A_\sigma \dashrightarrow B_\sigma$ such that
$$R_{\sigma\to\tau}(f,g) \;\Leftrightarrow\; \forall x, y \in A_\sigma . R_\sigma(x,y) \;\Longrightarrow\; R_\tau(fx, gy).$$

However, both of these notions turn out to be unduly restrictive. The definition that we shall adopt in fact subsumes both of the above possibilities:

style *standard* realizability, while $\mathbf{MPER}(A)$ is the corresponding model for Kreisel-style *modified* realizability. This is the reason for our terminology "modified extensional collapse". Experts will note that our definition of $\mathbf{MPER}(A)$ is not quite the "right" one—we have omitted mention of the right-absorptive set, which is not needed for our present purposes.

Definition 1.17 (Morphisms between PTSs) *Let A, B be weak PTSs. A* morphism $R : A \to B$ *is a family of total relations $R_\sigma : A_\sigma \rightarrowtail\!\!\!\!\!\rightarrow B_\sigma$ such that*

$$R_{\sigma\to\tau}(f,g) \wedge R_\sigma(x,y) \wedge (fx\!\downarrow \vee gy\!\downarrow) \implies R_\tau(fx,gy).$$

If A, B are both weak PTSs over X, a morphism $A \to B$ over X *is a morphism $R : A \to B$ such that $R_{\overline{0}} = \mathrm{id}_X$.*

[Motivate convention re definedness. Hang on, I've forgotten exactly whether we need it somewhere... It's similar to the situation for the s condition, somehow.]

[Examples of morphisms, showing the ways they can arise. More later!] From total hulls and extensional collapses; Effective substructure inclusions; Interpretations of syntactic calculi in semantic structures (mention theories); Translations or implementations of one syntactic calculus in another.

Morphisms clearly compose, and so we have a category **WPTS** of weak PTSs and morphisms between them, with full subcategories **PTS** and **TTS** of partial and total type structures respectively. Likewise, for any set X we have the category $\mathbf{WPTS}_X$ of weak PTSs and morphisms over X, with full subcategories $\mathbf{PTS}_X$ and $\mathbf{TTS}_X$.

There are many interesting special classes of morphisms, such as substructure inclusions and quotient maps. A morphism R is a quotient map iff each R_σ is single-valued and surjective; the quotient maps may be characterized as precisely the morphisms with a right inverse. Hence, if C is a substructure of B and A is a quotient of C, we obtain a morphism $A \to B$. This occurs, for instance, when A is an extensional collapse of B, in which case the morphism $A \to B$ is a logical relation. Conversely, if we have a *discrete* morphism $R : A \to B$ (that is, $R(x,y) \wedge R(x',y)$ implies $x = x'$), then A is a subquotient of B: we may regard R itself as a weak TS, with an evident inclusion $R \to B$ and quotient map $R \to A$.

Note, however, that the constructions given in Section 1.4 are highly *non-functorial* with respect to the above notion of morphism: morphisms $A \to B$ do not in general give rise to morphisms between their total hulls or their extensional collapses. Moreover, extensional collapses (i.e. logical relations) are not even closed under composition, and this is one reason for favouring our definition of morphism over the concept of logical relation. In fact, our morphisms correspond to *pre-logical relations* as studied in [HS99], where the reader may find further details arguments for the advantages of pre-logical over logical relations.

We also have the following simple facts:

Proposition 1.18 *Suppose A is an extensional PTS over X. Then*
(i) Any morphism $A \to B$ over X is discrete.
(ii) The only morphism $A \to A$ over X is the identity.

If A, B are extensional PTSs over X, let us write $A \preceq B$ iff there exists a morphism $A \to B$ over X. In view of part (ii) of the above proposition, the relation $\preceq$ is a partial order. Let us write $\mathcal{P}(X)$ [resp. $\mathcal{T}(X)$] for the poset of

extensional partial [resp. total] type structures over X, and $\mathcal{P}_{\mathit{eff}}(X)$ [$\mathcal{T}_{\mathit{eff}}(X)$] for the sub-poset of partial [total] type structures that admit an effective structure.

The fact that notions of computability can be partially ordered in this way is special to *functional* notions, and is one reason why functional notions of computability seem to deserve special treatment. The idea is that thinking about these posets should help us to understand the important notions of computability, how they are related to one another, and what makes them special.

2 Total functionals

We now use the framework of total type structures over N to survey some natural classes of hereditarily total functionals of finite type. Our intention here is to collect together in a systematic fashion what is known about the interesting notions of total computability that have been discovered to date. There remains, of course, the possibility that other equally compelling and natural notions await discovery; however, the fact that (as we shall see) so many attempts to define a class of total computable functionals converge on one of these notions constitutes some kind of evidence that the known type structures cover all the really important notions of computability.

In each case, we first survey a range of characterizations or presentations of the type structure in question, then consider some of the intrinsic properties of the type structure.

2.1 The total continuous functionals

We begin with a very important class of functionals which, though not embodying an effective notion of computability, occupies a central position among interesting total type structures: the Kleene/Kreisel total continuous functionals. The relevance of this type structure to effective computability will become clearer in the following sections, but for the time being, one might be motivated by the evident connections between effectivity and continuity (see [Lon01a, Section 1]) to seek a total type structure based on a notion of continuity as a halfway house on the way towards type structures based on effectivity.

For a more detailed study of this type structure and its characterizations, with proofs of most of the main results, we refer the reader to [Hyl79]. Another more recent survey is [Nor99].

2.1.1 Characterizations of the continuous functionals

There are very many ways to characterize the type structure C of continuous functionals, providing ample evidence of its canonical status. We here review some of the important characterizations.

2.1.1.1 L-spaces There are many cartesian closed categories of spaces equipped with some kind of "convergence" structure (e.g. L-spaces, filter spaces, certain kinds of topological space) in which C arises simply as the type structure over

the space N of natural numbers. Our account of these characterizations is drawn largely from [Hyl79].

Perhaps the simplest such category is the category of *L-spaces* as studied e.g. in [Kur52].[3] An L-space is a set X equipped with a relation $x_i \downarrow x$ (read as "x_i tends to x") between countably infinite sequences $x_0, x_1, \ldots$ and elements x, satisfying the following axioms (whose details are not too important here):

- If all but finitely many of the x_i are x, then $x_i \downarrow x$;
- If $x_i \downarrow x$ then any subsequence of x_i tends to x;
- If x_i does not tend to x, there is a subsequence $\langle y_i \rangle$ of $\langle x_i \rangle$ such that no subsequence of $\langle y_i \rangle$ tends to x.

As a particular example we have the L-space N of natural numbers, with $n_i \downarrow n$ iff all but finitely many of the n_i are n.

A function $f : X \to Y$ between L-spaces is *continuous* if whenever $x_i \downarrow x$ in X, $f(x_i) \downarrow f(x)$ in Y. The category **LSP** of L-spaces and continuous functions is cartesian closed: the exponential Y^X is the space of continuous functions $f : X \to Y$, with $f_i \downarrow f$ iff whenever $x_i \downarrow x$ in X, $f_i(x_i) \downarrow f(x)$ in Y. Moreover, morphisms $1 \to X$ in **LSP** clearly correspond to elements of the space X. Thus, if we define

$$C_{\overline{0}} \;=\; N, \qquad C_{\sigma\to\tau} \;=\; C_\tau^{C_\sigma}$$

then the underlying sets of the spaces C_σ constitute the type structure C.

2.1.1.2 Topological characterizations There is an interesting category of topological spaces which arises naturally as a reflective subcategory of **LSP**. Any L-space has a natural topology on it, whose open sets are the sets O such that if $x_i \downarrow x$ and $x \in O$ then $\langle x_i \rangle$ is eventually in O. The topological spaces that arise from L-spaces in this way are called *sequential spaces*. Given a sequential space X we may recover the convergence relation by $x_i \downarrow x$ iff $\langle x_n \rangle$ is eventually inside any open set containing x. Thus, the sequential spaces may be characterized topologically as those spaces in which the open and "sequentially open" sets coincide.

It is now easy to check that the category **SEQ** of sequential spaces and continuous maps (in the sense of ordinary topology) is a full reflective subcategory of **LSP** and is cartesian closed. Moreover, it turns out that the type structure over N (with the discrete topology) in **SEQ** coincides with C, since the inclusion **SEQ** $\to$ **LSP** preserves the relevant exponentials (see [Hyl79, Section 10]). This gives us a purely topological construction of C.

In fact, there are many other cartesian closed categories of topological spaces that give rise to C in this way, such as the well known category of compactly generated Hausdorff spaces. (This fact is mentioned in [Hyl79] and proved in [Nor80, Chapter 3].) However, all these characterizations suffer from the limitation that even the topology on $C_{\overline{2}}$ does not have a countable basis. (It is shown in [Hyl79] that this limitation applies to any topological characterization of C.) This means that these characterizations are not readily amenable to effectivization or a constructive treatment of computability.

[3] There they are called L^*-spaces.

2.1.1.3 Filter spaces In the other direction, one can extend **LSP** to the larger cartesian closed category **FIL** of *filter spaces*. A *filter* on a set X is a non-empty family Φ of non-empty subsets of X which is upward-closed and closed under binary intersections. A *filter space* is a set X together with for each $x \in X$ a collection $F(x)$ of filters on X such that

- If $\Phi \supseteq \Psi$ and $\Psi \in F(x)$ then $\Phi \in F(x)$;
- The principal ultrafilter $\{A \mid x \in A\}$ is in $F(x)$.

Informally, each filter in $F(x)$ gives a way of "converging to" x by means of a family of subsets of X representing approximations to x. A *continuous map* $f : (X, F) \to (Y, G)$ between filter spaces is a function $f : X \to Y$ that preserves these convergent families: if $\Phi \in F(x)$ then the filter generated by $f(\Phi)$ is in $G(f(x))$.

It is straightforward to check that the resulting category **FIL** is cartesian closed. We also have the filter space $N = (\mathrm{N}, F)$, where $F(n)$ contains only the principal ultrafilter over n. It turns out that the type structure over N in **FIL** is again isomorphic to C; in fact, there is an inclusion functor **LSP** $\to$ **FIL** that preserves the relevant exponentials. Details are given in [Hyl79].

One significant advantage of the filter space approach over the topological one is that the filter spaces C_σ in question all have countable bases: that is, there is a countable family of subsets of C_σ from which we can completely determine the filter structure. We will exploit this in giving an effectivization of this construction in Section 2.2 below.

2.1.1.4 Kreisel's definition via neighbourhoods We now give Kreisel's construction of C via a notion of continuity based on *neighbourhoods*. In Kreisel's original definition [Kre59], "neighbourhoods" were first constructed rather syntactically as formal expressions; and the points themselves were obtained essentially as limits of shrinking sequences of neighbourhoods. We give here a slight recasting of Kreisel's definition due to Hyland [Hyl75, pages 21–22], and trivially equivalent to that in [Kre59].

This definition and those that follow have an intensional flavour, in that we first define some class of intensional objects and then obtain C as a type structure of functionals represented or *realized* by these objects. Here, we first define a non-extensional type structure K, where each set K_σ is equipped with a countable base N_σ for a topology on K_σ. Consider $\mathrm{N}_\perp$ as a poset with the usual ordering: $x \sqsubseteq y$ iff $x = y$ or $x = \perp$. Let $K_{\overline{0}}$ be N and $N_{\overline{0}}$ be the set of singletons. For types $\sigma = \sigma_1 \to \cdots \to \sigma_r \to \overline{0}$, Given K_{σ_i} and N_{σ_i} for $1 \leq i \leq r$, let K_σ be the set of functions $\alpha : N_{\sigma_1} \times \cdots \times N_{\sigma_r} \to N_\perp$ such that

- α is anti-monotone: if $V_i \subseteq U_i$ for each i then $\alpha(U_1, \ldots, U_r) \sqsubseteq \alpha(V_1, \ldots, V_r)$;
- If $x_i \in K_{\sigma_i}$ for each i, there are sets $U_i \in N_{\sigma_i}$ such that $\alpha(U_1, \ldots, U_r) = \perp$.

Let N_σ consist of finite intersections of sets of the form $\{\alpha \mid \alpha(U_1, \ldots, U_r) = n\}$ where $U_i \in N_{\sigma_i}$ and $n \in \mathrm{N}$. Application in K is defined in an obvious way: if $\alpha \in K_{\sigma \to \tau}$ (where $\tau \equiv \tau_1 \to \cdots \to \tau_t$) and $\beta \in K_\sigma$, then $\alpha \cdot \beta(U_1, \ldots, U_t)$ is the unique p such that there exists $U \in N_\sigma$ with $\beta \in U$ and $\alpha(U, U_1, \ldots, U_t) = p$.

We may now obtain C as the extensional collapse of K with respect to equality on N. In fact, it is easy to show that K is pre-extensional, so that C is a quotient of K. That this definition of C is equivalent to the others given here is proved in [Hyl75].

2.1.1.5 Realizability over $\mathcal{P}\omega$ We now give three characterizations of C as the type structure realized by certain "continuous" partial combinatory algebras as indicated in Example 1.16: Scott's $\mathcal{P}\omega$, Plotkin's $\mathbb{T}^\omega$, and Kleene's second model K_2. All of these PCAs will reappear later in the paper.

2.1.1.6 Realizability over $\mathbb{T}^\omega$ Ershov; Berger?

2.1.1.7 Kleene's definition via associates Finally we give Kleene's original definition of C from [Kle59]. Here the intensional objects involved are known as *associates*; modulo trivialities, these are essentially realizers drawn from a PCA known as *Kleene's second model* or K_2 (introduced in [KV65]). Since the latter observation is not really explicit in the literature, we here approach the definition of C via K_2.

Elements of K_2 are total functions $\mathrm{N} \to \mathrm{N}$. As a first step, we can regard a element $f \in \mathrm{N}^\mathrm{N}$ as encoding a partial continuous operator $\bar{f} : \mathrm{N}^\mathrm{N} \rightharpoonup \mathrm{N}$ as follows. Fix on any coding $\langle \cdots \rangle$ of finite sequences of natural numbers as natural numbers, and define $\tilde{g}(k) = \langle g(0), \ldots, g(k-1) \rangle$. Given f and g, we may consider successively the values of $f(\tilde{g}(k))$ for $k = 0, 1, 2, \ldots$, until we find the smallest number l such that $f(\tilde{g}(l)) > 0$; we then set $\bar{f}(g) = f(\tilde{g}(l)) - 1$. (If no such l exists then $\bar{f}(g)$ will be undefined.) Thus, $f\langle g(0), \ldots, g(k-1) \rangle = 0$ signals a request for further values of g, while $f\langle g(0), \ldots, g(k-1) \rangle = n+1$ signals an affirmation that $\bar{f}(g) = n$. Formally, we may define

$$\bar{f}(g) \;=\; f\langle \tilde{g}(\mu k. f\langle \tilde{g}(k) \rangle > 0) \rangle - 1.$$

It is clear that any continuous function $F : \mathrm{N}^\mathrm{N} \to \mathrm{N}$ arises as $\bar{f}$ for infinitely many f.

We may now exploit this idea to define a partial application operation $\bullet$ on N^N. Given $g \in \mathrm{N}^\mathrm{N}$ and $n \in \mathrm{N}$, write g_n for the function defined by $g_n(0) = n$, $g_n(m+1) = g(m)$. Define $f \bullet g = \lambda n. \bar{f}(g_n)$ whenever the right hand side is total; otherwise take $f \bullet g$ to be undefined.[4] It is routine but tedious to check that the resulting structure $K_2 = (\mathrm{N}^\mathrm{N}, \bullet)$ is a partial combinatory algebra.

Next, choose some reasonable representation of the natural numbers in K_2. For example, take $\bar{n} = \lambda x.n$, and let $\equiv$ be the equality relation on the subset $\{\bar{n} \mid n \in \mathrm{N}\}$ of K_2. (This defines the natural number object in $\mathbf{PER}(K_2)$; any PER isomorphic to this would of course suffice.) Finally, let C be the extensional collapse of K_2 w.r.t. $\equiv$.

Note that Kleene's original definition of C did not explicitly involve K_2; instead, partial equivalence relations $\equiv_k$ ($k \geq 1$) (corresponding to the types $\overline{k}$)

[4] It is a curious feature of this definition that, in logical terms, the definedness of application is a Π_2 property than a Σ_1 property. But what is perhaps more curious is that K_2 turns out to be such an interesting and useful object despite this.

were defined as follows:

$$f \equiv_1 f' \Leftrightarrow f = f', \qquad f \equiv_{k+1} f' \Leftrightarrow (\forall g, g' . g \equiv_k g' \Longrightarrow \bar{f}(g) \simeq \bar{f}'(g').$$

The sets C_σ for *pure* types σ were then defined by

$$C_{\overline{0}} = \mathrm{N}, \qquad C_{\overline{k}} = K_2/\equiv_k \ (k \geq 1)$$

with application operations defined in the obvious way. From the existence of some evident isomorphisms in $\mathbf{PER}(K_2)$ it is easy to see that this agrees with the above definition.

Not surprisingly, the modified extensional collapse of K_2 w.r.t. $\equiv$ also yields a characterization of C, though we are not aware of a proof of this fact in the literature. [IS THIS IN BEZEM?]

2.1.1.8 Other characterizations Another characterization of C via a hyperfinite type structure (in the sense of non-standard analysis) is given in [Nor83]. Some further characterizations of C as extensional collapses of important type structures over $\mathrm{N}_\perp$ will be mentioned in Section 4.

In summary, it seems that virtually any natural attempt to build a total type structure over N based on a notion of continuity leads to the type structure C. We are therefore inclined to assert that C is *the* full continuous total type structure over N.

2.1.2 Properties of the continuous functionals

2.2 A recursive submodel

As we have seen, the type structure C is based purely on a notion of continuity, and no effectivity is involved. However, one might be inspired by the various characterizations of C to

In this section we will discuss the type structure naturally arising as an effective submodel of C; in the following section we will discuss the natural effective analogue of C.

2.2.1 Characterizations

2.2.2 Properties

2.3 A recursive analogue

2.3.1 Characterizations

2.3.2 Properties

2.3.3 Failure of Church's thesis

Incompatibility.

2.4 Kleene computability

TO BE WRITTEN. Definition; funny contravariance fact.

In C: non-computability of Fan functional (Tait). Normann: no finite basis. Conclusion: a bit odd.

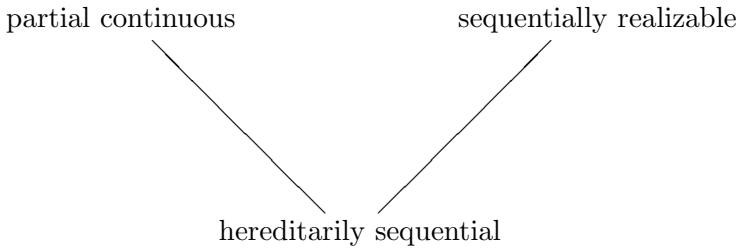
3 Partial functionals

Not written yet. Meanwhile, here are some relevant paragraphs borrowed from [Lon99]:

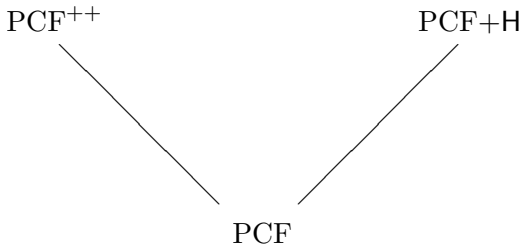
At present, it seems that there are essentially three different finite type structures that occur widely in nature, each of which comes in both a "full continuous" and an "effective" flavour. All six of these type structures have a number of different characterizations, and all have some claim to being 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 [Sco93].
- The *hereditarily sequential* functionals of Nickau [Nic94]: this coincides with the finite type structure arising from the fully abstract game models for PCF due to Abramsky, Hyland *et al* [AJM00, HO00].
- The *strongly stable* functionals of Bucciarelli and Ehrhard [BE91]: these coincide with the *sequentially realizable* functionals of Longley [Lon98b].

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



Here PCF^{++} is the extension of PCF with parallel-or and exists operators as studied in [Plo77]. For the functional H, see [Lon98b]. One can characterize the effective type structures as the closed term models for these programming languages.

For each of these six type structures there are known examples of PCAs giving rise to it:

- The partial continuous functionals arise from many "continuous PCAs" such as the Scott graph model $\mathcal{P}\omega$ [Sco76], the D_∞ models [Sco72], Plotkin's universal domain T^ω [Plo78], and Kleene's second model K_2 [KV65].
- 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 [Kle45].
- The hereditarily sequential functionals arise from various PCAs recently constructed by Abramsky (see [Lon98a]). They also 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 [MRS99]).
- The effective hereditarily sequential functionals (i.e. the PCF-definable functionals) arise from the effective analogues of any of these, and from the term models of certain impure λ-calculi (see [MRS99]). Moreover, the *Longley-Phoa Conjecture* asserts that this type structure also arises from the pure term model Λ^0/T for any semi-sensible λ-theory T (see e.g. [Lon95]).
- The sequentially realizable (SR) functionals arise from van Oosten's combinatory algebra $\mathcal{B}$ [Oos97], and from the combinatory algebra $\mathcal{A}$ constructed by Abramsky (see [Lon98a]). They also arise from the combinatory algebra $\mathcal{B}_2$ described in [Lon98b].
- The effective SR functionals arise from the effective analogues of these.

Theorem 3.1 (Anti-Church's thesis for partial functionals) *The effective partial continuous functionals (corresponding to* PCF^{++}*) represent a maximal element of $\mathcal{P}_{\mathit{eff}}$. Furthermore, there is no morphism over* $\mathrm{N}_\perp$ *from the effective SR functionals to the* PCF^{++} *functionals. Hence $\mathcal{T}_{\mathit{eff}}(\mathrm{N}_\perp)$ does not contain a top element.*

[Explain the implications more fully, along the lines of [Lon98b].]

4 Relating partial and total functionals

4.1 Relating the above type structures

Total notions as subquotients/collapses of partial notions (demonstrates orthogonality of issues).

Vice versa: getting PCF++ from System T (Escardo; Statman?).

4.2 Mixing total and partial function types

References

[AJM00] S. Abramsky, R. Jagadeesan, and P. Malacaria. Full abstraction for PCF. *Information and Computation*, 163:409–470, 2000.

[Bar84] H.P. Barendregt. *The Lambda Calculus: Its Syntax and Semantics*. North-Holland, revised edition, 1984.

[BE91] A. Bucciarelli and T. Ehrhard. Sequentiality and strong stability. In *Proc. 6th Annual Symposium on Logic in Computer Science*, pages 138–145. IEEE, 1991.

[BR89] B. Bloom and J.G. Riecke. LCF should be lifted. In *Proc. Conf. Algebraic Methodology and Software Technology*. Dept. Comp. Sci., University of Iowa, 1989.

[Cur30] H.B. Curry. Grundlagen der kombinatorischen Logik. *Amer. J. Math.*, 52:509–536, 789–834, 1930.

[Hen50] L. Henkin. Completeness in the theory of types. *Journal of Symbolic Logic*, 15(2):81–91, 1950.

[HO00] J.M.E. Hyland and C.-H. L. Ong. On full abstraction for PCF: I, II and III. *Information and Computation*, 163:285–408, 2000.

[HS99] F. Honsell and D. Sannella. Pre-logical relations. In *Proc. Computer Science Logic, CSL'99*, volume 1683 of *Lecture Notes in Computer Science*, pages 546–561. Springer, 1999.

[Hyl75] J.M.E. Hyland. *Recursion theory on the countable functionals*. PhD thesis, University of Oxford, 1975.

[Hyl79] J.M.E. Hyland. Filter spaces and continuous functionals. *Ann. Math. Logic*, 16:101–143, 1979.

[Kle45] S.C. Kleene. On the interpretation of intuitionistic number theory. *J. Symb. Logic*, 10, 1945.

[Kle59] S.C. Kleene. Countable functionals. In A. Heyting, editor, *Constructivity in Mathematics: proceedings of the colloquium held at Amsterdam, 1957*, pages 81–100. North-Holland, 1959.

[Kre59] G. Kreisel. Interpretation of analysis by means of functionals of finite type. In A. Heyting, editor, *Constructivity in Mathematics: proceedings of the colloquium held at Amsterdam, 1957*, pages 101–128. North-Holland, 1959.

[Kur52] C. Kuratowski. *Topologie Vol. I.* Warsaw, 1952.

[KV65] S.C. Kleene and R.E. Vesley. *The Foundations of Intuitionistic Mathematics.* North-Holland, 1965.

[Lon95] J.R. Longley. *Realizability Toposes and Language Semantics.* PhD thesis, University of Edinburgh, 1995. Available as ECS-LFCS-95-332.

[Lon98a] J.R. Longley. Realizability models for sequential computation. In preparation; an incomplete draft is available from the author's home page., 1998.

[Lon98b] J.R. Longley. The sequentially realizable functionals. Technical Report ECS-LFCS-98-402, Department of Computer Science, University of Edinburgh, 1998. To appear in *Annals of Pure and Applied Logic.*

[Lon99] J.R. Longley. Matching typed and untyped realizability. In L. Birkedal, J. van Oosten, G. Rosolini, and D.S. Scott, editors, *Proc. Workshop on Realizability, Trento*, 1999. Published as Electronic Notes in Theoretical Computer Science 23 No. 1, Elsevier. Available via `http://www.elsevier.nl/locate/entcs/volume23.html`.

[Lon01a] J.R. Longley. Notions of computability at higher types II. 60 pages approx. To appear in Proc. Logic Colloquium 2000. Draft available from `http://www.dcs.ed.ac.uk/home/jrl`, 2001.

[Lon01b] J.R. Longley. Notions of computability at higher types III. 30 pages approx, in preparation. Draft available from `http://www.dcs.ed.ac.uk/home/jrl`, 2001.

[Mit96] J.C. Mitchell. *Foundations of Programming Languages.* MIT Press, 1996.

[MRS99] M. Marz, A. Rohr, and T. Streicher. Full abstraction and universality via realisability. In *Proc. 14th Annual IEEE Symposium on Logic in Computer Science.* IEEE, 1999.

[Nic94] H. Nickau. Hereditarily sequential functionals. In *Proc. 3rd Symposium on Logical Foundations of Computer Science, Lecture Notes in Computer Science 813*, pages 253–264. Springer, 1994.

[Nor80] D. Normann. *Recursion on the countable functionals*, volume 811 of *Lecture Notes in Mathematics.* Springer, 1980.

[Nor83] D. Normann. Characterising the continuous functionals. *Journal of Symbolic Logic*, 48:965–969, 1983.

[Nor99] D. Normann. The continuous functionals. In E.R. Griffor, editor, *Handbook of Computability Theory*, pages 251–275. North-Holland, 1999.

[Oos97] J. van Oosten. A combinatory algebra for sequential functionals of finite type. Technical Report 996, University of Utrecht, 1997. To appear in Proc. Logic Colloquium, Leeds.

[Plo77] G.D. Plotkin. LCF considered as a programming language. *Theoretical Computer Science*, 5, 1977.

[Plo78] G.D. Plotkin. T^ω as a universal domain. *Journal of Computer and System Sciences*, 17:209–236, 1978.

[Rie93] J.G. Riecke. Fully abstract translations between functional languages. *Math. Struct. in Comp. Science*, 3, 1993.

[Sco72] D.S. Scott. Continuous lattices. In F.W. Lawvere, editor, *Toposes, Algebraic Geometry and Logic.* Springer, 1972.

[Sco76] D.S. Scott. Data types as lattices. *SIAM Journal of Computing*, 5(3):522–587, 1976.

[Sco93] D.S. Scott. A type-theoretical alternative to ISWIM, CUCH, OWHY. *Theoretical Computer Science*, 121:411–440, 1993. First written in 1969 and widely circulated in unpublished form since then.

[SF90] D. Sitaram and M. Felleisen. Reasoning with continuations II: Full abstraction for models of control. In *Conference on Lisp and Functional Programming*. ACM, 1990.

[Sie90] K. Sieber. Relating full abstraction results for different programming languages. In *Proc. 10th Conference on Foundations of Software Technology and Theoretical Computer Science, Bangalore*. Springer LNCS 472, 1990.

[Sta85] R. Statman. Logical relations and the typed λ-calculus. *Information and Control*, 65:85–97, 1985.