Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science

Electronic Notes in Theoretical Computer Science 308 (2014) 129–146

www.elsevier.com/locate/entcs

Total Maps of Turing Categories

J.R.B. Cockett [1,2]

Department of Computer Science
University of Calgary
Calgary, Alberta, Canada, T2N 1N4

P.J.W. Hofstra[3]

Department of Mathematics and Statistics
University of Ottawa
Ottawa, Ontario, Canada, K1N 6N5

P. Hrubeš[4]

Department of Computer Science and Engineering
University of Washington
Seattle, WA, USA

Abstract

We give a complete characterization of those categories which can arise as the subcategory of total maps of a Turing category. A Turing category provides an abstract categorical setting for studying computability: its (partial) maps may be described, equivalently, as the computable maps of a partial combinatory algebra. The characterization, thus, tells one what categories can be the total functions for partial combinatory algebras. It also provides a particularly easy criterion for determining whether functions, belonging to a given complexity class, can be viewed as the class of total computable functions for some abstract notion of computability.

Keywords: Computability theory, Partial Combinatory Algebra, Turing Category, Complexity Theory.

1 Introduction

Turing categories [1,2,6], provide an abstract categorical setting for computability theory which, unlike partial combinatory algebras (PCAs) and related structures, are presentation-independent and purely formulated in terms of categorical properties. The standard example of a Turing category has objects powers of the natural

[1] This work was partial supported by NSERC Canada

[2] Email: robin@ucalgary.ca

[3] Email: phofstra@uottawa.ca

[4] Email: pahrubes@gmail.com

http://dx.doi.org/10.1016/j.entcs.2014.10.008

numbers and maps all the partial recursive functions. However, there are many other less well-known examples deriving from the "computable maps" of PCAs (or more generally relative PCAs) or from syntactical methods. Of special relevance for the purposes of the present work are the Turing categories described in [3] which have as total maps the programs belonging to various complexity classes (PTIME, LOGSPACE, etc.). These examples naturally lead to the question of exactly what categories can be the total maps of a Turing category. Intuitively, as in any Turing category one can simulate all computable functions, it would seem reasonable to suppose that the total maps would have to satisfy some fairly demanding closure properties.

The question is of significance for various reasons. To start with, it is one way to determine the limits of the applicability of Turing categories in studying computability. If it where impossible for the total maps of a Turing category to be exactly, say, the linear time functions, then one cannot hope to use Turing categories as a basis for investigating feasible computation at very low complexity levels. On the other hand, if one knows that the total maps of a Turing category *can* be of such low complexity then Turing categories can be a tool for formally unifying computability and complexity theory and allowing a fluid flow of ideas between the subjects.

A second reason for considering this question in the abstract categorical setting is that it leads to an interesting comparison with the more traditional view on these matters, namely via logic. Given a logical theory, one may ask which functional relations are provably total: the weaker the theory, the smaller the class of provably total maps. One may also wish for the system to be strong enough to allow for the *representation* of partial computable maps. It is well known that even relatively weak fragments of arithmetic ensure this. For example, Robinson's Arithmetic Q is enough to ensure all the partial recursive functions are represented. The study of complexity, using bounded and two sorted logics [5] for example, has further pushed the limits of these methods. It is, thus, in its own right, an interesting question to know exactly what an absolutely minimal logic for generating these settings really is. Although, this paper does not attempt to answer this question directly, the categorical framework we describe can certainly be backward engineered into a logical form: even a brief perusal indicates that there is a significant difference between these approaches, not least because a Turing category is not *a priori* based on arithmetic.

The third point of interest lies not so much in the question itself as in the methods used here to provide the answer. The proof that a cartesian category satisfying certain conditions can be embedded as a category of total maps in a Turing category makes use of two ideas: first, it uses the Yoneda embedding to create a canonical category of partial maps into which the original category embeds. Next, we use the concept of a *stack machine* in the presheaf category to create a partial combinatory algebra which in turn will generate the desired Turing category. A stack machine can be thought of as a categorical implementation of the canonical rewrite system in combinatory logic (but augmented with additional data). As

such, this concept helps clarify the connection between syntactical approaches to generating models of computation and categorical methods.

Below we develop necessary and sufficient conditions for a Cartesian category to be the total maps of a Turing category. The conditions are perhaps a little surprising as apparently very little is actually required: there must be a universal object, U, which has a pair of "disjoint" elements, and which has an "abstract retract" structure which allows coding of maps. A universal object is an object into which every other object can be embedded as a subobject: to have such an object is already a somewhat non-trivial requirement as this, in particular, implies there is an embedding $U \times U \longrightarrow U$ showing that U must be an infinite object. The remaining conditions are somewhat more technical: they are explained in section 3 below. However, it should be noted that in most of the standard applications, as described in section 5, these technical conditions can be by-passed.

We start the exposition by considering the much more general question of how a category can arise as the subcategory of total maps of an arbitrary restriction category. (For background on restriction categories we refer to [4].) This leads to the notion of a totalizing extension of the given category, and we show that the category of totalizing extensions of a particular category has a final object which is naturally a restriction category. This insight allows us to transfer the general question of finding a Turing category which extends a given cartesian category into finding a partial combinatory algebra in this final extension whose total maps include the given maps. This perspective allows us to propose necessary conditions for a Cartesian category to be the total maps of a Turing category, see section 3. To show that these are sufficient we demonstrate that one can build, using a simple abstract machine, a combinatory algebra in the final totalizing extension (actually of a slightly modified category) which has elements representing all the total maps: this suffices in view of Theorem 4.12 of [2]. Finally we provide simpler sufficient conditions to show the wide applicability of the theorem.

2 Totalizing map subcategories

When $\mathbb{X}$ is a restriction category, the inclusion of the subcategory of total maps $\mathrm{Total}(\mathbb{X}) \longrightarrow \mathbb{X}$ satisfies various properties. This leads to the following definition:

Definition 2.1 [Totalizing functor] A functor $T : \mathbb{X}' \longrightarrow \mathbb{X}$ is **totalizing** when it satisfies the following three conditions:

(i) T on objects, $T_{\mathsf{Obj}} : \mathsf{Obj}(\mathbb{X}') \longrightarrow \mathsf{Obj}(\mathbb{X})$, is an isomorphism,

(ii) T is faithful,

(iii) T is left factor[5] closed, meaning that when $Th = gf$ then there is a (necessarily unique) k such that $Tk = f$.

The following observation indicates that this class of functors is reasonably well-behaved.

[5] Note that "left" factor refers to the diagrammatic order of composition: $A \xrightarrow{Th} B = A \xrightarrow{f} B \xrightarrow{g} C$.

Lemma 2.2 *In the category of categories and functors, the class of totalizing functors, $\mathcal{T}$, form a stable system of monics:*

(i) Every $T \in \mathcal{T}$ is monic,

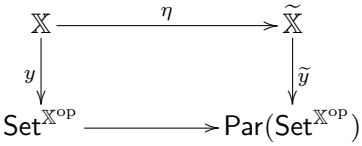
(ii) $\mathcal{T}$ is closed to composition,

(iii) $\mathcal{T}$ contain all isomorphisms,

(iv) $\mathcal{T}$ is closed to pulling back along any functor.

Proof. Routine. □

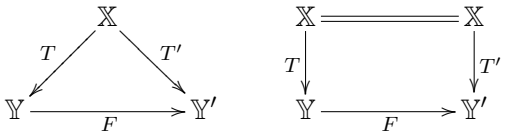
Recall that given any (small) category $\mathbb{X}$ the Yoneda embedding, $y : \mathbb{X} \to \mathsf{Set}^{\mathbb{X}^{\mathrm{op}}}$, gives a full and faithful embedding of $\mathbb{X}$ into presheaves on $\mathbb{X}$. This category of presheaves is finitely complete and therefore one can form the partial map category on all monics $\mathsf{Par}(\mathsf{Set}^{\mathbb{X}^{\mathrm{op}}})$. Notice that the total maps in this category are exactly the morphisms of $\mathsf{Set}^{\mathbb{X}^{\mathrm{op}}}$. Restricting this category to the representables gives a full subcategory which we shall denote $\widetilde{\mathbb{X}}$. While this is in general no longer a partial map category, it certainly is a restriction category. We then have the following pullback:



This exhibits $\mathbb{X}$ as a subcategory of total maps of the restriction category $\widetilde{\mathbb{X}}$, whence η is totalizing.

While there may be many other categories $\mathbb{Y}$ into which $\mathbb{X}$ embeds via a totalizing functor, we shall now make precise the sense in which $\eta : \mathbb{X} \to \widetilde{\mathbb{X}}$ is the universal such functor.

Consider the category of **totalizing extensions** of $\mathbb{X}$, denoted $\mathsf{Ext}_{\mathcal{T}}(\mathbb{X})$, whose objects are totalizing functors $T : \mathbb{X} \to \mathbb{Y}$ and whose morphisms are commuting triangles below $\mathbb{X}$



which **reflect total maps** in the sense that the square on the right is a pullback.

Proposition 2.3 $\mathsf{Ext}_{\mathcal{T}}(\mathbb{X})$ *is a (finitely) complete category in which $\eta : \mathbb{X} \to \widetilde{\mathbb{X}}$ is the final object.*

Proof. The verification that pullback exist (and are constructed as in Cat is routine. We shall show that $\eta : \mathbb{X} \to \widetilde{\mathbb{X}}$ is the final object in this category. To this end, suppose $T : \mathbb{X} \to \mathbb{Y}$ is totalizing. Without loss of generality we may assume that $\mathbb{Y}$ has the same collection of objects as $\mathbb{X}$. It is then obvious how the functor $E : \mathbb{Y} \to \widetilde{\mathbb{X}}$ should act on objects. For a morphism $g : X \to X'$ in $\mathbb{Y}$, we need to define a

partial map $y(X) \to y(X')$. First consider the following sieve on X:

$$S_g = \{h : Z \to X | gh \in \mathbb{X}\}.$$

This sieve corresponds to a subobject of the representable $y(X)$. Next, define a natural transformation $\tau(g) : S_g \to y(X')$ by

$$\tau(g)_Z : S_g(Z) \to \mathbb{X}(Z, X'); \qquad \tau(g)_Z(h) = gh.$$

This data defines a partial map $E(g) : y(X) \to \mathcal{Y}(X')$, which is total if and only if $g \in \mathbb{X}'$. This in particular shows that E reflects total maps. The verification that E is functorial is straightforward and left to the reader.

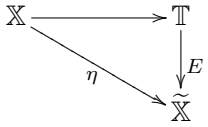
To show that E is unique suppose we are given an extension $E' : \mathbb{Y} \to \widetilde{\mathbb{X}}$. Since E' must respect maps from $\mathbb{X}$, we verify its action on a map $g : X \to X'$ which is not in $\mathbb{X}$. Then $E'(g)$ is a span $y(X) \supseteq S \xrightarrow{\sigma} y(X')$. Now if $h \in S_g$, the composite gh is in $\mathbb{X}$, and therefore $h \in S$, whence $E'(gh) = E'(g)E'(h) = \sigma(h)$ must equal $y(gh)$. This means that $E'(g) \geq E(g)$. To show the converse, assume that $h \in S$ but $h \notin S_g$. Then $gh \notin \mathbb{X}$. But then $E'(gh)$ is a total map, contradicting the reflection of total maps. □

We also note that in case $\mathbb{Y}$ is a restriction category and $T : \mathbb{X} \to \mathbb{Y}$ the inclusion of total maps, then in fact $E : \mathbb{Y} \to \widetilde{\mathbb{X}}$ is a restriction functor. In addition, we have the following result, which states that if $\mathbb{X}$ has products, then E preserves the induced restriction products.

Lemma 2.4 *Suppose that $\mathbb{X}$ has finite products. Then $\widetilde{\mathbb{X}}$ also has finite restriction products, and η preserves them.*

Proof. Finite products in a split restriction category are completely determined by their counterparts in the total map subcategory. Thus, as the inclusion $\mathsf{Set}^{\mathbb{X}^{\mathrm{op}}} \to \mathsf{Par}(\mathsf{Set}^{\mathbb{X}^{\mathrm{op}}})$ preserves products and $y : \mathbb{X} \to \mathsf{Set}^{\mathbb{X}^{\mathrm{op}}}$ preserves products cutting down to $\widetilde{\mathbb{X}}$ preserves products. □

With a view towards our aim of characterizing those categories which arise as the total map subcategory of a Turing category, we can now observe the following. When $\mathbb{X} \to \mathbb{T}$ is a totalizing extension with $\mathbb{T}$ a Turing category, then we have the following situation:



Then by the above lemma, E is a Cartesian restriction functor; since any such functor preserves partial combinatory algebras, we find that there is a PCA $E(U)$, the image of the Turing object $U \in \mathbb{T}$. Thus we obtain:

Proposition 2.5 *$\mathbb{X}$ is the total map category of a Turing category if and only if there is a combinatory algebra in $\widetilde{\mathbb{X}}$ whose total computable maps include the maps of $\mathbb{X}$.*

Thus we may see our original problem as one of finding a suitable PCA in $\widetilde{\mathbb{X}}$ for which all the maps in $\mathbb{X}$ are computable. Indeed, if such a PCA exists, then we may let $\mathbb{T}$ be the subcategory of $\widetilde{\mathbb{X}}$ on the computable maps. Notice one rather nice aspect of this reformulation of the problem: since the maps of $\mathbb{X}$ already account for *all* the total maps in $\widetilde{\mathbb{X}}$, the total maps represented by a PCA in $\widetilde{\mathbb{X}}$ necessarily lie in $\mathbb{X}$ already; thus we don't have to worry about having too many total maps represented. Therefore, the only thing to verify when constructing a candidate PCA is that it represents all the maps in $\mathbb{X}$.

3 Properties of the total maps of a Turing category

We are now in a position to collect the necessary conditions for being the total maps of a Turing category. Clearly the total map category has to be Cartesian (i.e. has finite products), as Turing categories are Cartesian by definition. The remaining conditions are somewhat more technical. However, it should be stressed that in many cases of interest these conditions greatly simplify.

The purpose of the section is to prove:

Proposition 3.1 *Every total map category of a Turing category has a universal object which has a pair of disjoint elements and is equipped with an abstract retract structure for which there exist codings.*

Below we introduce the required notions of having a *universal object*, *disjoint elements*, *abstract retract structure*, and *codes*, showing each is present in the total map category of a Turing category.

3.1 A universal object

One of the properties of the Turing object U in a Turing category is that every object is a retract of it. More explicitly, given any object A, there are morphisms $\iota_A : A \longrightarrow U$ and $\rho_A : U \longrightarrow A$ for which $\rho_A \iota_A = 1_A$. This forces ι_A to be a total map (in fact, a monomorphism), but ρ_A can still be partial. We will typically denote the situation by $A \prec U$, or by $(\iota_A, \rho_A) : A \prec U$ if necessary.

We call an object U in a category **universal** if for every object A there exists a monomorphism $\iota_A : A \longrightarrow U$. (Thus we don't ask that every object is a retract of U; this is stronger, and will be analyzed in the section on abstract retract structures below.)

Lemma 3.2 *In any Cartesian category $\mathbb{X}$ with a universal object U:*

(i) There is always an element $\iota_1 : 1 \longrightarrow U$ of the universal object;

(ii) There is always an embedding $\iota_{U\times U} : U \times U \longrightarrow U$;

(iii) The homset $\mathbb{X}(U, U)$ either has one element, in which case $\mathbb{X}$ is trivial, or is infinite.

3.2 A pair of disjoint elements

A Turing object U in a Turing category admits the interpretation of all combinatory logic terms. In particular, U has two global elements $\mathtt{t} = \lambda^* xy.x$ and $\mathtt{f} = \lambda^* xy.y$. We shall prove that these two elements have the property of being **disjoint** in the sense that whenever $\mathtt{t}h = \mathtt{f}h$ for a map h, the domain $\overline{h}$ of h must be a *strict* initial object in the idempotent splitting. For the subcategory of total maps, this simply means that for any h with $\mathtt{t}h = \mathtt{f}h$ the domain of h is a strict initial object.

Proposition 3.3 *In every Turing category the total maps have a universal object U with a disjoint pair of elements.*

Notice first that when the Turing category is trivial (in the sense that it is equivalent to the terminal category) all objects are strict initial objects, and hence all elements are disjoint. In general, in a Turing category, an element will be disjoint from itself only when the category is trivial as this forces the initial and final object to be the same.

We start by observing:

Lemma 3.4 *Let $\mathbb{X}$ be a Cartesian category. A map $h : H \longrightarrow 1$ makes*

$$H \times A \xrightarrow{h \times 1} 1 \times A \xrightarrow{\pi_1} A \underset{g}{\overset{f}{\rightrightarrows}} B$$

commute for all $f, g : A \longrightarrow B$ if and only if H is a strict preinitial object.

Recall that an object H is called **preinitial** if there is at most one map from that object to any other object. It is **strict preinitial** if any object with a map to H is itself (strict) preinitial. Preinitial objects are quite common: for example in the category of (commutative) rings $\mathbb{Z}$ is the initial object, while $\mathbb{Z}_n$ is preinitial for each n. However, neither $\mathbb{Z}$ nor $\mathbb{Z}_n$ are strict preinitial.

Proof. Suppose $x, y : H \longrightarrow A$. Then:

$$x = x\pi_1\langle h, 1\rangle = x\pi_1(h \times 1)\Delta = y\pi_1(h \times 1)\Delta = y\pi_1\langle h, 1\rangle = y.$$

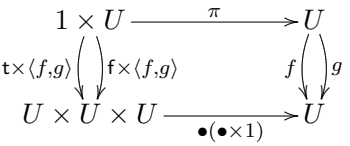
So H is preinitial. To show that it is strict, suppose that $q : Q \longrightarrow H$; then the above reasoning applies to qh, making Q preinitial. □

Notice that if H is strict preinitial then, as $H \times Y \xrightarrow{\pi_0} H$, it follows that $H \times Y$ is always strict preinitial. This means in a Cartesian category once one has one strict preinitial there must, for each object, be a preinitial with a map to that object. This does not imply there will be an initial object, but it does force that an initial object, if it exists, is automatically strict.

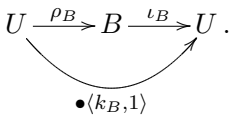
We are now ready to prove that the elements $\mathtt{t}$ and $\mathtt{f}$ in a Turing category are disjoint.

Lemma 3.5 *In any Turing category, if $h : H \longrightarrow 1$ is a total map which equalizes* t *and* f *(as chosen above) then H is a strict initial object in the total map category. Thus the elements* t *and* f *are disjoint.*

Proof. Suppose $f, g : A \longrightarrow B$ are total maps. Without loss of generality we may assume $A = B = U$. Note that the diagram



commutes serially. But then precomposing with $h \times 1 : H \times U \longrightarrow 1 \times U$ shows that h satisfies the conditions of the lemma above and so is a strict preinitial object. It remains only to show that there is a total map $H \longrightarrow B$ for each object B. Each B is a retract of the Turing object and so the composite $\iota_B \rho_B : U \longrightarrow U$ is a split idempotent. As with any map in a Turing category, there is a code $k_B : 1 \longrightarrow U$ such that $\bullet\langle k_B, 1\rangle = \iota_B \rho_B$, as in



While ρ_B may be partial, we argue that $\rho_B h$ is total:

$$\bullet\langle k_B, 1\rangle h = \bullet\langle k_B h, h\rangle = \bullet\langle \mathsf{k} h, h\rangle = \bullet(\mathsf{k} \times 1)\Delta h = \pi_0 \Delta h = h$$

so that $\iota_B \rho_B h$ is total, whence $\rho_B h$ is total. □

It may be useful at this stage to provide an example of a total map category which has a universal object and yet *cannot* be the total maps of a Turing category. The simplest example, which also shows that such a category cannot consist entirely of preinitial objects, is when the category is a meet-semilattice. Then the only element is the identity on the top and this must be disjoint from itself, forcing the top to also be the bottom thereby collapsing the lattice.

3.3 An abstract retract structure

Recall that each object A in a Turing category comes equipped with $(\iota_A, \rho_A) : A \prec U$ exhibiting it as a retract of the (chosen) Turing object. There may be many choices for this family of retractions although we shall assume that $(\iota_U, \rho_U) = (1_U, 1_U)$. Below we describe how this structure introduces an analogous structure on the total map category.

First, consider a span between representable objects $y(X)$ and $y(X')$ in $\mathsf{Set}^{\mathbb{X}^{\mathrm{op}}}$:

$$y(X) \xleftarrow{s} A \xrightarrow{t} y(X').$$

The apex A need not be representable, but we may consider the family of spans in $\mathbb{X}$ arising by covering A by representables as follows:

$$\mathcal{S}(s,t) = \{X \xleftarrow{sh} B \xrightarrow{th} y(X') | h : \mathcal{Y}(B) \to A\}.$$

This family is then clearly closed under precomposition with maps in $\mathbb{X}$. In fact, we may regard $\mathcal{S}(s,t)$ as a category whose objects are the spans in $\mathbb{X}$ factoring through (s,t), and whose morphisms are morphisms of spans.

Conversely, consider a family $\mathcal{R}$ of spans in $\mathbb{X}$ from X to X' which is closed under precomposition. For the present purposes, we shall call such a family an **abstract span**. Regarding $\mathcal{R}$ as a category, we may consider the functor

$$\mathcal{R} \to \mathrm{Span}(y(X), y(X')) \xrightarrow{\mathrm{Apex}} \mathsf{Set}^{\mathbb{X}^{\mathrm{op}}}$$

which sends a span $X \xleftarrow{p} B \xrightarrow{q} X'$ to the $y(B)$. The colimit of this diagram gives a span $y(X) \leftarrow \hat{\mathcal{R}} \to y(X')$ in $\mathsf{Set}^{\mathbb{X}^{\mathrm{op}}}$. The proof of the following is now routine:

Lemma 3.6 *The assignments $(s,t) \mapsto \mathcal{S}(s,t)$ and $\mathcal{R} \mapsto \hat{\mathcal{R}}$ are mutually inverse (up to isomorphism of spans).*

Next, we wish to characterize when the left leg of a span between representables is monic (so that it is a partial map). Call an abstract span $\mathcal{R}$ **deterministic** when $(p,q),(p,q') \in \mathcal{R}$ implies $q = q'$.

Lemma 3.7 *Given an abstract span $\mathcal{R}$ corresponding to a span $y(X) \xleftarrow{s} \hat{\mathcal{R}} \xrightarrow{t} y(X')$ between representables, s is monic if and only if the spans in $\mathcal{R}$ are deterministic.*

Of course, in this situation $\hat{\mathcal{R}}$ may be regarded as a sieve on X: it is precisely the sieve

$$\hat{\mathcal{R}}(Z) = \{h : Z \to X | (h,k) \in \mathcal{R} \text{ for some } k : Z \to X'\}.$$

So far, we have described partial maps between representables in terms of abstract spans. Next, we wish to characterize when such a partial map is in fact a retraction of a morphism $\iota_A : A \to U$. So suppose that in $\mathsf{Set}^{\mathbb{X}^{\mathrm{op}}}$, we have a span $y(X) \supseteq S \xrightarrow{t} y(X')$ which is a retraction of a map $\iota : X' \to X$. This of course means that $\iota \in S$ and $t(\iota) = 1_{X'}$. In terms of the corresponding abstract span $\mathcal{S}$, this means that $(\iota, 1) \in \mathcal{S}$ and that for each $(h,f) \in \mathcal{S}$ and each diagram

$$\begin{array}{ccccc} X' & \xleftarrow{k} & P & & \\ {\scriptstyle\iota}\downarrow & & \downarrow{\scriptstyle z} & & \\ X & \xleftarrow[h]{} & M & \xrightarrow[f]{} & X' \end{array}$$

with the square commuting, we have $fz = k$. We shall refer to this condition by saying that the abstract span $\mathcal{S}$ is **retracting** on ι. To summarize:

Lemma 3.8 *Given a morphism $\iota : X' \to X$ in $\mathbb{X}$ and a span $y(X) \supseteq S \xrightarrow{t} y(X')$, the span is a retraction of ι in $\mathsf{Par}(\mathsf{Set}^{\mathbb{X}^{\mathrm{op}}})$ if and only if the corresponding abstract span contains $(\iota, 1)$ and is retracting on ι.*

We are now ready for the main definition of this section:

Definition 3.9 An **abstract retract structure** on a Cartesian category with a universal object U consists of a choice of embeddings $\iota_A : A \to U$ (with $\iota_U = 1_U$) and, in addition, for each object A an **abstract retraction** of ι_A, meaning a family of spans $\mathcal{R}_A$ from U to A satisfying:

[RS.1] Each $\mathcal{R}_A$ is closed under precomposition;

[RS.2] Each $\mathcal{R}_A$ is deterministic;

[RS.3] $U \xleftarrow{\iota_A} A \xrightarrow{1_A} A \in \mathcal{R}_A$;

[RS.4] Each $\mathcal{R}_A$ is retracting on ι_A.

We shall indicate an abstract retraction pair by $(\iota_A, \mathcal{R}_A) : A \prec U$.

Here are some examples abstract retractions for a fixed morphism $\iota_A : A \to U$.

Examples 3.1 *(1) If $\mathcal{R}$ is an abstract retraction, then $\mathcal{R}_U$ is always just the family $U \xleftarrow{x} X \xrightarrow{x} U$. Indeed suppose $U \xleftarrow{h} Y \xrightarrow{f} U \in \mathcal{R}_U$ then by* **[RS.4]** *$h1_Y = \iota_U h$, so $f = f1_X = h$. Similarly, if ι_A is an isomorphism then $\mathcal{R}_A$ is always just the family $U \xleftarrow{\iota_A x} X \xrightarrow{x} A$.*

(2) The smallest retraction structure for each A is the one generated by the span $(\iota_A, 1_A)$. Each span in the abstract retraction is of the form $U \xleftarrow{\iota_A x} X \xrightarrow{x} A$; this means the set is deterministic and retracting on ι_A.

(3) Suppose that ι_A has a (total) retraction ρ_A. Then we may generate a retraction structure $\{(x, x\rho_A) | x : X \to U\}$. Note that when $x = \iota_A$ we obtain the span $U \xleftarrow{\iota_A} A \xrightarrow{1_A} A$. As all the spans have their right leg determined by the left leg post-composed with ρ_A the set is deterministic.

(4) Finally, consider a totalizing extension $T : \mathbb{X} \to \mathbb{Y}$, in which U is a universal object. If, in $\mathbb{Y}$, each object A is equipped with a retraction $(\iota_A, \rho_A) : A \prec U$, then we may consider the abstract span in $\mathbb{X}$ given by $\mathcal{R}_A = \{U \xleftarrow{h} A \xrightarrow{f} A | \rho_A h = f\}$. This is readily seen to be an abstract retraction on ι_A.

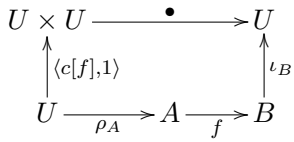
The last example shows in particular that whenever $\mathbb{T}$ is a Turing category with total map subcategory $\mathbb{X}$, there is an induced abstract retract structure on $\mathbb{X}$. For the record:

Lemma 3.10 *Every Turing category induces, for any Turing object U and chosen retractions $(\iota_A, \rho_A) : A \prec U$, an abstract retract structure on its total map category.*

3.4 Codes

A Turing object U in $\mathbb{T}$ is not only universal, but it also acts as a weak exponential for every pair of objects in the category. In particular, given $f : A \to B$, there

exists a code $c[f] : 1 \to U$ making



commute. Here, ρ_A and ι_B are part of the chosen retractions $A \prec U$ and $B \prec U$, and the morphism $U \times U \xrightarrow{\bullet} U$ is the universal application map. The code $c[f]$ is required to be total, but not unique. Choosing a code $c[f]$ for each f amounts to giving a family of mappings

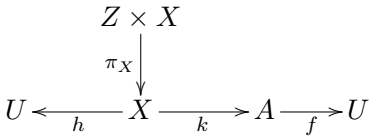
$$c = c_{A,B} : \mathbb{T}(A, B) \to \mathbb{T}(1, U). \tag{1}$$

Since the inclusion ι_B is total, it in fact suffices to specify codes only for the case $B = U$, that is, for maps $A \to U$.

We will now translate this existence of codes to structure on the total maps. Since codes are formulated in terms of the retractions ρ_A, this involves dealing with the abstract retract structure on the total maps. We may cut down the mapping (1) to the total maps to obtain a mapping $c = c_A : \mathbb{X}(A, U) \to \mathbb{X}(1, U)$, where $\mathbb{X} = \mathsf{Total}(\mathbb{T})$.

To motivate the coming definition, first we need a bit of notation. Given a span $U \xleftarrow{h} X \xrightarrow{k} A$ we denote by $Z^*(h, k)$ the span $U \xleftarrow{h\pi_X} Z \times X \xrightarrow{k\pi_X} A$, obtained by precomposing each leg with the projection $Z \times X \to X$. When $\mathcal{R}$ is a family of spans from U to A we will write $Z^*\mathcal{R} = \{(Z^*(h, k)|(h, k) \in \mathcal{R}\}$. Similarly, given $f : A \to U$, we write $f_*\mathcal{R} = \{(h, fk)|(h, k) \in \mathcal{R}\}$.

Now consider two maps $f : A \to U$ and $g : B \to U$ with codes $c[f], c[g]$, and suppose we have an object Z for which the unique total map $Z \xrightarrow{!} 1$ equalizes $c[f]$ and $c[g]$. Then as per example 3.1 (4), the a typical span in $\mathcal{R}_{\mathcal{A}}$ is of the form $U \xleftarrow{h} X \xrightarrow{k} A$ satisfying $k = \rho_A h$. Given such (h, k), we may then consider the span



In the above notation, this is $Z^*(fh, k) \in Z^* f_* \mathcal{R}$. Under the given condition on Z, the composite $fk\pi_X$ factors as $g \circ (\rho_B h \pi_X)$, as shown by the following calculation:

$$\begin{aligned} fk\pi_X &= f\rho_A h\pi_X = f\rho_A\pi_U(z \times h) = \bullet(c_A[f] \times U)(z \times h) \\ &= \bullet(c_A[f]z \times h) = \bullet(c_B[g]z \times h) = \bullet(c_B[g] \times U)(z \times h) \\ &= g\rho_B\pi_U(z \times h) = g\rho_B h\pi_X \end{aligned}$$

Note that the statement that $fk\pi_X$ factors as indicated implies that $Z^*(h, kf) \in Z^* f_* \mathcal{R}_A$ can be regarded as a span of the form $Z^*(h, gq)$ for $(h, q) \in \mathcal{R}_B$. By symmetry, this implies that $Z^* f_* \mathcal{R}_A = Z^* g_* \mathcal{R}_B$. Thus, informally speaking, when

restricting the abstract retractions to Z, the operations of composing with f and with g become equal.

This leads to the following:

Definition 3.11 A Cartesian category $\mathbb{X}$ with universal object U and retract structure $(\iota_A, \mathcal{R}_A) : A \prec U$ has **codes** when there are maps $c_A : \mathbb{X}(A, U) \to \mathbb{X}(1, U)$ such that whenever $Z \xrightarrow{!} 1$ satisfies $c[f]z = c[g]z$ for $f : A \to U, g : B \to U$ then $Z^* f_* \mathcal{R}_A = Z^* g_* \mathcal{R}_B$.

The preceding discussion shows:

Lemma 3.12 *The total map subcategory of a Turing category always has codes.*

4 Building a Turing category from total maps

We shall now construct a PCA in $\widetilde{\mathbb{X}}$ which has all the total maps computable. We use the notion of a "stack object", which is an object satisfying two domain equations which allow for the implementation of a rewriting system. Using the fact that $\widetilde{\mathbb{X}}$ admits a trace, we may then define a universal application morphism which will equip the stack object with a PCA structure. From now on, assume $\mathbb{X}$ is a Cartesian category satisfying the conditions explained in the previous section.

Before we start, however, it is convenient to modify $\widetilde{\mathbb{X}}$ a little. Recall that the Yoneda embedding does not preserve initial objects (if these happen to be present), but that there is always a subcanonical topology J on the category $\mathbb{X}$ which corrects this. (Simply take the smallest topology containing the empty cover on each initial object of $\mathbb{X}$.) This gives an embedding $\mathbb{X} \xrightarrow{y} Sh(\mathbb{X}, J)$ preserving any existing initial object. For the rest of the paper, we will let $\widetilde{\mathbb{X}}$ denote the full subcategory of the partial map category on $Sh(\mathbb{X}, J)$ on the representables. It will also be useful to assume that we have access to the splittings of restriction idempotents and to finite coproducts, thus it will be more convenient to work in $\mathsf{Par}(Sh(\mathbb{X}, J))$, and to restricting to $\widetilde{\mathbb{X}}$ by observing that the structures we build are always on representable objects.

Another important property of a partial map category of a (pre)sheaf category (and hence also of $\widetilde{\mathbb{X}}$) is that it is traced on the coproduct: given $f : X \to X + Y$ there is a map $f^\dagger : X \to Y$ which is the joint of all the finite partial iterates: $f^\dagger = \bigvee_{i \in \mathbb{N}} f^{(i)}$, where

$$\begin{aligned} f^{(1)} &= \sigma_1^{(-1)} f : X \to Y \\ f^{(2)} &= \sigma_1^{(-1)} f \sigma_0^{(-1)} f : X \to Y \\ &\cdots \\ f^{(n+1)} &= \sigma_1^{(-1)} f (\sigma_0^{(-1)} f)^n : X \to Y \end{aligned}$$

where $\sigma_i^{(-1)}$ is the partial inverse of the i^{th} coproduct injection. Intuitively, $f^\dagger$ takes an input $x \in X$, and computes the iterates $f^i(x)$ for as long as the output lies in X. Once the output lies in Y, this is the value of $f^\dagger(x)$.

4.1 Stack objects

An object A in a distributive restriction category is said to be a **stack object** in case there are maps

$$\mathsf{put} : 1 + A \times A \to A \qquad \mathsf{get} : A \to 1 + A \times A \qquad \text{with } \mathsf{put}\ \mathsf{get} = 1_{1+A\times A}.$$

Thus, a stack object can be indicated by $(\mathsf{put}, \mathsf{get}) : 1 + A \times A \prec A$.

Lemma 4.1 *In any distributive restriction category the following are equivalent conditions for an object:*

(i) $1 + 1 \prec A$ *and* $A \times A \prec A$*;*

(ii) $1 + A \times A \prec A$ *(it is a stack object)*

(iii) *For any polynomial functor* P*, we have* $P(A) = n_0.1 + n_1.A + ... + n_r.A^r \prec A$

The point is that in the universal extension $\widetilde{\mathbb{X}}$ of a category $\mathbb{X}$ satisfying the requirements discussed above, we know the first condition for a stack object is satisfied by the universal object U: indeed, $U \times U \prec U$ because $U \times U$ is an object of $\mathbb{X}$, and $1 + 1 \prec U$ because U has two disjoint elements $\mathsf{t}, \mathsf{f} : 1 \to U$, giving $[\mathsf{t}, \mathsf{f}] : 1 + 1 \to U$. A partial retraction may be defined by letting S be the sieve on U generated by $\{\mathsf{t}, \mathsf{f}\}$, and by defining a natural transformation $S \xrightarrow{\sigma} 1 + 1$ by $\sigma_X(m) = in_L(*)$ whenever m factors through t, and $in_R(*)$ otherwise. Note that this is well-defined precisely because t, f are disjoint, so that m factors through both t and f only when its domain is 0; but then $(1 + 1)(0)$ is a singleton since $1 + 1$ is assumed to be a sheaf.

4.2 A stack machine for partial combinatory algebras

We shall now use this trace to define a partial application $A \times A \xrightarrow{\bullet} A$ making A into a PCA. To this end, we will define a partial map $\mathsf{step} : A \times A \times A \to A \times A \times A + A$ whose trace then is of the desired type. More precisely, we will define $x \bullet y := \mathsf{step}^\dagger(x, y, [])$. The intuition that should be kept in mind is that the map step executes one step of a program/rewrite system, and that its trace runs the entire computation/rewriting sequence. The components of $A \times A \times A$ will be regarded as a code stack, a value, and a dump stack, respectively. Only when the code is end and the dump stack is empty does the computation halt.

To define step, we use the domain equations for A to fix three different repre-

sentations of our stack object:

$$\mathsf{end}, \mathsf{k}, \mathsf{s} : 1 \longrightarrow A \quad \mathsf{k}_0, \mathsf{s}_0 : A \longrightarrow A \quad \mathsf{s}_1 : A \times A \longrightarrow A \quad \mathsf{nam} : A \longrightarrow A$$
$$(\langle \mathsf{end}|\mathsf{k}|\mathsf{s}|\mathsf{k}_0|\mathsf{s}_0|\mathsf{s}_1|\mathsf{nam}\rangle, g_0) : 1 + 1 + 1 + A + A + A \times A + A \prec A$$
$$\mathsf{c}_0 : A \longrightarrow A \quad \mathsf{c}_1 : A \times A \longrightarrow A$$
$$(\langle \mathsf{c}_0|\mathsf{c}_1\rangle, g_1) : A + A \times A \prec A$$
$$\mathsf{nil} : 1 \longrightarrow A \quad \mathsf{cons} : A \times A \longrightarrow A$$
$$(\langle \mathsf{nil}|\mathsf{cons}\rangle, g_2) : 1 + A \times A \prec A$$

Now we define step by the following case distinction:

Code	Value	Stack	Code	Value	Stack
end	x	nil	exit with x		
k	x	S	end	$\mathsf{k}_0(x)$	S
$\mathsf{k}_0(x)$	y	S	end	x	S
s	x	S	end	$\mathsf{s}_0(x)$	S
$\mathsf{s}_0(x)$	y	S	end	$\mathsf{s}_1(x, y)$	S
$\mathsf{s}_1(x, y)$	z	S	x	z	$\mathsf{cons}(\mathsf{c}_0(y, z), S)$
$\mathsf{nam}(c_A[f])$	z	S	end	$f(\rho_A(z))$	S
end	v	$\mathsf{cons}(\mathsf{c}_0(y, z), S)$	y	z	$\mathsf{cons}(c_1(v), S)$
end	v'	$\mathsf{cons}(\mathsf{c}_1(v), S)$	v	v'	S

The one aspect which needs explanation concerns how we use the names of maps. The aim is to implement them as the composite of a retraction to the idempotent and the map itself. (Recall that $c_A[f]$ is a name for f relative to the embedding-retraction pair $A \prec U$ for which $\rho_A : U \longrightarrow A$ is a retraction, as detailed in Section 3.4.) Without these names, the stack machine can be thought of as an implementation of the usual rewriting system on combinatory logic; adding the names of maps from $\mathbb{X}$ together with the given rule essentially amounts to adding rewrites $c_A[f] \bullet a \longrightarrow f(a)$ to the system.

The step partial function is the join of all its individual components. For step to be well-defined all these components must be compatible, in the sense that they agree on overlaps of domains. Those which are in separate components of the stack object by design are disjoint and so compatible. However, the names of maps are all in the same component and, thus, we must establish compatibility of the implementation of names.

Lemma 4.2 step *is a well-defined partial map in* $\widetilde{\mathbb{X}}$ *on the stack object* U*.*

Proof. A typical span corresponding to a partial map $U \times U \times U \to U \times U \times U$ in the component of step which is defined on a tuple of the form $(\mathsf{nam}(c_A[f]), h, S)$ (for $f : A \to U$, $h : X \to U$) looks like

$$U \times U \times U \xleftarrow{\langle \mathsf{nam}(c_A[f])!_X, h, S\rangle} X \xrightarrow{\langle \mathsf{end}, fk, S\rangle} U \times U \times U$$

Here, $(h, k) \in \mathcal{R}(A)$. Given another such span

$$U \times U \times U \xleftarrow{\langle \mathsf{nam}(c_B[g])!_{X'}, h', S'\rangle} X' \xrightarrow{\langle \mathsf{end}, gk', S'\rangle} U \times U \times U$$

with $g : B \to U$ and $(h', k') \in \mathcal{R}_B$, we must show that these two spans are compatible. This is done by considering generalized elements of $z : Z \to X$ and $z' : Z \to X'$; we must verify that if $\langle \mathsf{nam}(c_A[f]!_X, h, S\rangle z = \langle \mathsf{nam}(c_B[g])!_{X'}, h', S'\rangle z'$, then also $\langle \mathsf{end}!_X, fk, S\rangle z = \langle \mathsf{end}!_{X'}, gk', S'\rangle z'$ It is clear that this holds for the last component. Thus we must show that if $c_A[f]!_X z = c_B[g]!_{X'} z'$ and $hz = h'z'$ then $fkz = gk'z'$.

The first condition means $c_A[f]!_Z = c_B[g]!_Z$ which, by the requirement on codes, implies (hz, fkz) is in $\mathcal{R}_B g$. This implies $(hz, fkz) = (hz, gv)$. But $hz = h'z'$, as $\mathcal{R}_B$ is deterministic, now gives $v = k'z'$ and so $fkz = gv = gk'z'$ as required. □

We are now ready for the main result:

Theorem 4.3 *Given a Cartesian category $\mathbb{X}$ with a universal object, a pair of disjoint elements and a retract structure with codes then the above definition of $\bullet$ in $\widetilde{\mathbb{X}}$ gives a partial combinatory algebra; the subcategory of $\widetilde{\mathbb{X}}$ on the computable maps is then a Turing category whose total maps are exactly the maps of $\mathbb{X}$.*

The proof consists of a verification that the combinators k and s indeed perform as required.

Proof. We first verify that $\mathsf{k} \bullet x \bullet y = x$ (where $\bullet$, as usual, associates to the left). Here $\mathsf{step}(\mathsf{k}, x, []) = \sigma_0(\mathsf{end}, \mathsf{k}_(x), [])$ has $\mathsf{step}(\mathsf{end}, \mathsf{k}_0(x), []) = \sigma_1(\mathsf{k}_0(x))$ so that $(\mathsf{k} \bullet x) = \mathsf{k}_0(x)$. But now

$$\begin{aligned}
\mathsf{k}_0(x) \bullet y &= \mathsf{step}^\dagger(\mathsf{k}_0(x), y, []) \\
&= \left\{ \begin{array}{ll} \sigma_0(c, v, d) & \mapsto \mathsf{step}^\dagger(c, v, d) \\ \sigma_1(x) & \mapsto x \end{array} \right\} \mathsf{step}(\mathsf{k}_0(x), y, []) \\
&= \left\{ \begin{array}{ll} \sigma_0(c, v, d) & \mapsto \mathsf{step}^\dagger(c, v, d) \\ \sigma_1(x) & \mapsto x \end{array} \right\} \sigma_1(x) \\
&= x
\end{aligned}$$

which verifies that $(\mathsf{k} \bullet x) \bullet y = x$.

Next, we need $\mathsf{s} \bullet x \bullet y$ to be as defined as x and y. But clearly $\mathsf{s} \bullet x = \mathsf{s}_0(x)$ and $\mathsf{s}_0(x) \bullet y = \mathsf{s}_1(x, y)$ so, as s_1 is total this requirement of the combinator is met.

Next, we calculate

$$\begin{aligned}((\mathsf{s} \bullet x) \bullet y) \bullet z &= \mathsf{s}_1(x,y) \bullet z \\ &= \mathsf{step}^\dagger(\mathsf{s}_1(x,y), z, []) \\ &= \left\{ \begin{array}{ll} \sigma_0(c,v,d) & \mapsto \mathsf{step}^\dagger(c,v,d) \\ \sigma_1(x) & \mapsto x \end{array} \right\} \mathsf{step}(\mathsf{s}_1(x,y), z, []) \\ &= \mathsf{step}^\dagger(x, z, \mathsf{cons}(\mathsf{c}_1(y,z), [])) \\ &= \mathsf{step}^\dagger(\mathsf{end}, x \bullet z, \mathsf{cons}(\mathsf{c}_1(y,z), [])) \\ &= \mathsf{step}^\dagger(y, z, \mathsf{cons}(\mathsf{c}_0(x \bullet z), [])) \\ &= \mathsf{step}^\dagger(\mathsf{end}, y \bullet z, \mathsf{cons}(\mathsf{c}_0(x \bullet z), [])) \\ &= \mathsf{step}^\dagger(x \bullet z, y \bullet z, []) \\ &= (x \bullet z) \bullet (y \bullet z) \end{aligned}$$

Here we use repeatedly the identity:

$$\mathsf{step}^\dagger(x,y,S) = \mathsf{step}^\dagger(\mathsf{end}, \mathsf{step}^\dagger(x,y,[]), S) = \mathsf{step}^\dagger(\mathsf{end}, x \bullet y, S)$$

which is true by virtue of the fact that the trace is defined inductively. More precisely, we have

$$\mathsf{step}^{(n)}(x,y,s) = \bigsqcup_{i+j=n+1} \mathsf{step}^{(i)}(\mathsf{end}, \mathsf{step}^{(j)}(x,y,[]), s).$$

The left hand expression is empty unless the iteration terminates in that number of steps. If there is a (first) stage j at which the left hand iteration returns the stack to its original state and the first coordinate is end then $j+1$ can be used to terminate the inner loop on the right hand side to bring the two iterations to the same state. Subsequently the result will be the same. If there is no such j both sides will be the empty map. □

5 Applications

For many of the obvious applications we have proven much more than is actually needed. Here are two corollaries of our main theorem:

Proposition 5.1 *Given a Cartesian category $\mathbb{X}$ in which*

- *Every object has at least one element;*
- *There is a universal object U;*
- *There is a monic (set) map $c : \coprod_{A \in \mathbb{X}} \mathbb{X}(A,U) \longrightarrow \mathbb{X}(1,U)$;*
- *There is a faithful product preserving functor into $U : \mathbb{X} \longrightarrow \mathsf{Set}$.*

Then $\mathbb{X}$ occurs as the total maps of a Turing category.

These conditions include the PTIME maps between binary natural numbers:

Corollary 5.2 *The PTIME maps between binary numbers occur as the total maps of a Turing category.*

In fact, one may use a linear time pairing operation on binary numbers by setting, for example, the pairing of $b_1 \cdots b_k$ and $c_1 \cdots c_n$ to be $b_1 b_1 \cdots b_k b_k 01 c_1 c_1 \cdots c_n c_n$. This means we also have:

Corollary 5.3 *The linear time maps between binary numbers occur as the total maps of a Turing category.*

Another class of examples is contained in the following:

Proposition 5.4 *Any countable Cartesian category with a universal object and a pair of disjoint elements is the total maps of a Turing category.*

Here we observe that once one has two distinct elements the fact that one has a stack object allows one to easily obtain countably many distinct and disjoint elements. This means that there is a monic assignment of maps to elements and one can use the smallest abstract retractions.

6 Conclusion

To unify the abstract notion of computability embodied in Turing categories with the study of feasible computation (e.g. LINEAR, LOGSPACE, PTIME, ...) minimally one must know whether these functional complexity classes can form the total maps of a Turing category. In [3] it was shown that both LOGSPACE and PTIME functions had a natural description as the total maps of Turing categories. However, in order to obtain those results, it was necessary to use the well-known facts from complexity theory that transducers and Turing machines can universally simulate themselves with an overhead which can be accommodated within (respectively) LOGSPACE and PTIME. This meant the argument that these complexity classes could be modelled by Turing categories relied heavily on the details of the machine models and the way in which resources were measured. In particular, as there is no widely accepted machine model which can simulate itself with a *linear* time overhead, the linear time maps could not be so readily included in this approach.

The power of the results outlined in this paper is, precisely, that they are abstract. That is, they do not depend on the peculiarities of machine models or on the way resources are counted.

By describing necessary and sufficient conditions for a Cartesian category to be the total maps of a Turing category we have demonstrated that, perhaps somewhat counterintuitively, Turing categories *are* applicable beyond the traditional confines of combinatorial completeness into feasible computation and the domain of complexity theory. In this regard Turing categories, therefore, provide – by more than mere analogy – a medium for the transfer of ideas between computability and complexity theory and thus for a potential economy of presentation which may be beneficial to the further development of the subject. Finally, we believe the methods employed in establishing the results in this paper are yet another strong indication that a complete understanding of the categorical aspects of computability and complexity

necessarily involves the study of partial combinatory algebras in categories other than Set.

References

[1] J.R.B. Cockett, *Categories and computability.* Lecture notes available at http://pages.cpsc.ucalgary.ca/ robin.

[2] J.R.B. Cockett and P.J.W. Hofstra, *Introduction to Turing categories* Annals of Pure and Applied Logic, Volume 156, Issues 2-3, December 2008, Pages 183-209.

[3] J.R.B. Cockett, J. Diaz-Boïls, J. Gallagher and Pavel Hrubeš *Timed Sets, Functional Complexity, and Computability*. Electronic Notes in Theoretical Computer Science, Volume 286, September 2012, Pages 117137.

[4] J. R. B. Cockett and S. Lack, *Restriction categories I: categories of partial maps.* Theoretical Computer Science 270(1-2): 223-259, 2002.

[5] S. Cook and P. Nguyen, *Logical Foundations of Proof Complexity.* In: Perspectives in Logic, Cambridge University Press, 2010.

[6] R. Di Paola and A. Heller, *Dominical categories: recursion theory without elements.* Journal of Symbolic Logic, Volume 56,1987.