

Introduction to Turing categories

J. R. B. Cockett ^{a,1}, P. J. W. Hofstra ^{b,*,2}

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

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

Abstract

We give an introduction to Turing categories, which are a convenient setting for the categorical study of abstract notions of computability. The concept of a Turing category first appeared (albeit not under that name or at the level of generality we present it here) in the work of Longo and Moggi; later, Di Paolo and Heller introduced the closely related recursion categories. One of the purposes of Turing categories is that they may be used to develop categorical formulations of recursion theory, but they also include other notions of computation, such as models of (partial) combinatory logic and of the (partial) lambda calculus. In this paper our aim is to give an introduction to the basic structural theory, while at the same time illustrating how the notion is a meeting point for various other areas of logic and computation. We also give a detailed exposition of the connection between Turing categories and partial combinatory algebras and show the sense in which the study of Turing categories is equivalent to the study of PCAs.

Key words: Computability theory, models of computation, partial combinatory algebra, partial map category, partial algebra

1991 MSC: 03D75, 08A55, 18A15, 18C50, 68Q05

1. Introduction

Recursion theory, in its traditional form, is the study of computable functions and subsets of the natural numbers. However, soon after its foundations had been laid by Kleene, Post, Turing and Gödel, the initial focus on computability soon shifted to other topics. In particular, various researchers started investigating to which extent one could

* Corresponding author.

Email addresses: robin@ucalgary.ca (J. R. B. Cockett), phofstra@uottawa.ca (P. J. W. Hofstra).

¹ Partially supported by NSERC, Canada.

² Supported by NSERC, Canada.

replace the natural numbers by other structures and thus generalize recursion theory to wider settings (see, e.g. [22] for a lively discussion of why one would do such a thing). Such a generalization can take different forms: one typically starts by isolating certain structural aspects of recursive functions and the natural numbers which one wants to abstract. For example, one may replace the natural numbers by ordinals and generalize recursion theory to recursion on ordinals or admissible sets (see Chapter 3 in [17] for an overview, or [47] for an introduction). Or, if one takes closure properties of the recursive functions such as enumeration and parametrization as fundamental, then one is led to computation on various kinds of abstract structures (e.g. [16,15,44])³.

In connection to the present work, we mention in particular Uniformly Reflexive Structures, developed by Wagner and Strong [51,49], and Moschovakis' work on abstract computability [33] and (pre)computation theories [34]. Essentially, both approaches (which differ in presentation but mostly agree in content) single out a class of well-behaved combinatory structures and prove the elementary recursion theoretic results for the objects in this class. In hindsight, this may be seen as the study of the class of decidable partial combinatory algebras in the category of Sets; as such, they are examples of the kind of structures we shall be considering in this paper.

Given the success with which categorical methods were used in other areas of logic and theoretical computer science, it was not a surprise that people started looking for possible categorical formulations of recursion theory. One may distinguish two lines: in the first, the aim is to find a categorical setting in which certain aspects of recursion theory are reflected in a particularly nice way. This led to the discovery of the Recursive Topos by Mulry [35] and of the Effective Topos by Hyland [21], as well as to the study of various categories of domains and effective domains (see [1] for an overview and references). The second line is a more low-level approach: one tries to find minimal categorical settings which embody certain elementary recursion theoretic phenomena. It is this second line which is of interest to us; it started with the pioneering work by Eilenberg and Elgot [14], who first investigated recursiveness from the point of view of elementary category theory.

Longo and Moggi, in [30,29], made significant contributions to this programme. While their main motivation seems to have been the development of categorical settings for the study of computability at higher types (as opposed to elementary recursion theory), they formulated the appropriate categorical concepts corresponding to Gödel numberings and parametrization. Moreover, they studied these concepts in the context of models of the lambda calculus (being mainly interested in type structures, they concentrated on the cartesian closed case). Later, in [31], this was generalized to arbitrary total combinatory algebras. One of their ideas was to show that every total combinatory algebra gave rise to a category with certain properties (which embody the notions of Gödel numbering and parametrization), while every such category in turn must contain⁴ a total combinatory algebra. This should be compared to the well-known result that every model of the untyped lambda calculus may be realized as a reflexive object in a cartesian closed category. Although this was a significant step forward, we stress that the results were

³ A more elaborate and detailed account of various approaches as well as further references can be found in [36].

⁴ In fact, this was not formulated quite in this manner: it was shown that there was an object in the category such that the global sections of that object form a total combinatory algebra in Sets. See also the remarks in Section 4.4.

formulated only for total structures, and that no full treatment of partial structures was offered.

Much different in flavor is the well-known 1987 paper “Dominical categories: recursion theory without elements”, [38], in which Roberto Di Paola and Alex Heller introduced a class of categories called *recursion categories* and showed how in these categories the basic aspects of recursion theory could be developed. Their primary goal apparently was to use this as an approach to algebraic treatments of the incompleteness theorems; however, the development of basic concepts from recursion theory and the categorical proofs of recursion theoretic results took center stage, and a variety of result and concepts, ranging from the recursion theorems and creative sets to Rice’s theorem, were analysed. However, the connection to partial combinatory algebras was not made, and neither was any connection to other category-theoretic approaches to computability.

Later publications by Heller, Di Paola and Montagna [18,39], and also by Lengyel [24], presented new examples of recursion categories, as well as techniques for constructing such categories. Also, better understanding of partial map categories due to the work by Rosolini and Robinson in [42,40] allowed for improved treatment of the material (mainly, [43]). Recently, a few other people have taken up this line of research: for example, in [48], the possibility of formulating notions of relative computability in these settings is investigated, while [25] adds to the study of recursion categories with specific additional structure (which is needed for the formulation of certain recursion-theoretic concepts).

Despite the evident close relationship between the work by Moggi and Longo and that by Di Paola and Heller, there has, apparently, been no effort to unite them and to harvest the rewards of such a unification. The main purpose of this paper is therefore to present such a unified approach, in the form of Turing categories. These directly generalize the kind of categories studied by Longo and Moggi, and thus are closely related to partial combinatory algebras and models of the untyped lambda calculus. On the other hand, the particular axiomatization of recursion categories, while capturing a number of important examples, is not wide enough to make a claim at being a setting for abstract computability. In order to broaden the approach we rejected two technical aspects of the setting they proposed. First Di Paolo and Heller insisted on zero morphisms: this forces the partiality to be nontrivial, thus excluding total models of computation. From a strictly recursion-theoretic viewpoint these total models are perhaps of secondary interest. However, we felt that it was unnecessary to exclude those models from the outset, and we shall introduce zero morphisms only when required. The fact that, in this paper, we do not introduce a zero at all, means that there is quite a bit to say before this condition is required: indeed, various results (such as the recursion theorems, or the completeness of the halting set) can be proved without it.

More seriously, they imposed the requirement that all objects in a recursion category be isomorphic. This not only seems an awkward condition from a type-theoretic point of view, it also excludes a large number of examples, namely all those structures where one does not have surjective pairing (which is responsible for obtaining an isomorphism $A \times A \cong A$ as opposed to a mere retraction). In addition, this requirement prevents the class of categories to be stable under idempotent splittings, a construction which will be shown to be of crucial importance for the structural and representational results in this work.

Thus, in our formulation of Turing categories, these conditions are notably absent. This way, we not only develop all results in as general a setting as possible, we also

keep full control over the interaction between the total and the partial world and do not discard total examples (such as models of the lambda calculus and, indeed, various effective models useful for higher type computation).

The work by Lars Birkedal [7] also bears close connections to the present work. In his thesis, a notion of weakly closed partial cartesian category is developed, with the aim of providing a general framework in which to study realizability constructions. In our language, a Turing category is a weakly closed partial cartesian category with a universal object (a concept also appearing in *loc. cit.*). Birkedal also shows that every PCA gives rise to such a category, and proves an “impredicativity entails untypedness”-result, stating that the presence of a universal object is both necessary and sufficient to construct a tripos, and hence a topos (in turn, this is closely related to the result in [26]).

Purpose of this work. One of the goals of this paper is to give a clear exposition of the basic theory of Turing categories, and the way in which they are connected to various other topics, ranging from recursion theory to models of combinatory logic and from the lambda calculus to realizability. As indicated above, we cannot claim much originality for the concept of a Turing category. However, the basic structural theory of Turing categories hasn’t been developed previously, and we use this paper as an occasion to give a precise and detailed account of that development. Although the development of recursion theory is mostly preserved for a later paper, we will have occasion to indicate how some elementary concepts are to be viewed in this setting, and we hope this both illustrates and justifies the axiomatization.

Furthermore, we give an exposition of partial combinatory algebras (PCAs) with an emphasis on their relation to Turing categories. Longo and Moggi already showed how a total combinatory algebra (in the category of Sets) gives rise to a Turing category, and how one can extract a combinatory algebra from a Turing category by applying the global sections functor. We first extend this to the partial case, working over an arbitrary cartesian base category. We remark that Birkedal had already shown how to construct such a category from a PCA, but that his method is a generalization of the usual idempotent splitting of a monoid of endofunctions, while we obtain a smaller category which does not have its idempotents split.

As a first significant improvement over these existing results, we turn the construction into an actual correspondence by investigating to which extent the Turing category is determined by the PCA. We want to draw attention to the fact that, in order to recover the Turing category (and thus obtain a representational result), one actually needs a relativised notion of computability based on pairs of PCAs; this seems to be more than a coincidence, since a similar construction has appeared in realizability, and also in [28], in the context of computability at higher types.

The correspondence between Turing categories and PCAs provides insight in both directions. First of all, examples of PCAs give rise to examples of Turing categories, and one may translate results about PCAs into results about Turing categories. This can be beneficial as it is usually the case that the language of PCAs is more concise than the categorical language. Next, Turing categories will be seen to be a presentation-free representation of PCAs; indeed, a Turing category may be viewed as representing an equivalence class of PCAs, where the equivalence is given by so-called strict simulation. This in turn sets up to possibility of obtaining results about PCAs up to strict equivalence by purely categorical methods (for if two Turing categories have different

categorical properties, then they distinguish the classes of PCAs). In particular, the correspondence allows one to study how certain structural or recursion theoretic properties of Turing categories relate to properties of PCAs. For example, the work by Di Paola and Heller already indicates that additional requirements on Turing categories (for example, coproducts, regularity, discreteness, certain choice principles) become important in the formulation of recursion-theoretic concepts. Thus one may consider “PCAs with coproducts”, “PCAs with choice”, etcetera, and all of these notions are stable under equivalence.

We also bring simulations into the picture: these are (a strict version of) morphisms of PCAs which have proved useful in realizability, but so far haven’t been applied in this context. We will argue that these morphisms arise naturally within our setting and provide us with the right way of expressing that two notions of computability are equivalent, a question which in a paper on this subject should not be avoided.

Finally, while this paper should form an accessible introduction to anyone who is interested in how category theory and recursion theory meet (and who has an elementary grasp of both), it is also supposed to set the stage for a number of more specialized results on which we hope to report later.

Outline of the paper. We begin by providing the necessary background on partial map categories in Section 2; in particular we introduce restriction categories and functors, as well as partial products, and idempotent splittings. The section also serves to introduce some relevant notation and terminology.

Section 3 contains the key definition of a Turing category, and explores the direct consequences of the definition. A number of examples are described in some detail in order to illustrate how the various components of the definition relate to classical concepts in computability. We also present a few elementary results concerning the non-canonicity of the structure of a Turing category. Moreover, a number of basic constructions to construct new Turing categories from old ones is discussed.

Then, in Section 4, we turn attention to partial combinatory algebras (PCAs). We explain how combinatory completeness allows us to build a Turing category from a PCA, generalizing Longo and Moggi’s approach in a straightforward manner. Moreover, we present one of the basic representational results for Turing categories, by showing how every Turing category essentially arises in this way, provided we generalize PCAs to relative PCAs, and Turing categories are regarded up to Morita-equivalence.

In Section 5 we present some results which tighten the connection between PCAs and Turing categories. First, we recapitulate the notion of a simulation between PCAs. This allows us to speak of equivalent PCAs, and then to prove that equivalent PCAs induce equivalent Turing categories, and vice versa. We also give a criterion for a PCA in a Turing category to be a Turing object.

We end the paper with a brief description of some of the topics and results which can be expected in future work.

Acknowledgements. We have greatly benefited from discussions with, and helpful suggestions from Phil Scott, Phil Mulry and Joachim de Lataillade. We also wish to express our gratitude to the anonymous referee who not only provided us with many suggestions for improvement, but also alerted us to an omission in our discussion of relevant related work.

| | |
|------------|--|
| R.1 | $f\bar{f} = f$ |
| R.2 | $\bar{f}\bar{g} = \bar{g}\bar{f}$ whenever $\text{dom}(f) = \text{dom}(g)$ |
| R.3 | $\overline{gf} = \overline{g}\bar{f}$ whenever $\text{dom}(f) = \text{dom}(g)$ |
| R.4 | $\bar{g}f = f\overline{g\bar{f}}$ whenever $\text{cod}(f) = \text{dom}(g)$ |

Fig. 1. Axioms for a restriction combinator

2. Background and Preliminaries

In this Section we introduce the general categorical setting in which we will be working, and fix some terminology and notation. All of the material discussed here can be found in more complete and detailed form in [9,10].

2.1. Categories of partial maps

There are various approaches to handling partiality in category theory. For our purposes, restriction categories provide a convenient framework: their axiomatization is algebraic and fairly simple, while they are at the same time a completely general setting. Essentially, a restriction category is a category with an additional operator, which associates to each map $f : X \rightarrow Y$ an idempotent $\bar{f} : X \rightarrow X$ which is to be thought of as the domain of definition of f . This idempotent measures the degree of partiality of the map f . For example, in Par , the category of sets and partial functions, we associate to a partial function f the partial endofunction

$$\bar{f}(x) = \begin{cases} x & \text{if } f(x)\downarrow \\ \uparrow & \text{otherwise.} \end{cases}$$

Here, we temporarily use the notation $f(x)\downarrow$ to indicate that x is in the domain of definition of f , and $f(x)\uparrow$ for the assertion that $f(x)$ is undefined.

Definition 2.1 (Restriction category). *A restriction category is a category \mathcal{C} endowed with a combinator $\bar{(-)}$, sending $f : A \rightarrow B$ to $\bar{f} : A \rightarrow A$, such that the axioms in Figure 1 are satisfied.*

These axioms are intended to capture the behaviour of domains; indeed, in the example Par of sets and partial functions the axioms are readily verified. Also, it can be shown that the axioms are independent, in that none of them follows from the others.

Special classes of maps. A morphism f in a restriction category for which $\bar{f} = 1$ is called a *total map*. The identity map is total and total maps compose (to show this, it is useful to prove first that $\overline{gf} = \bar{g}\bar{f}$ whenever the composite gf is defined). Thus the total maps in a restriction category \mathcal{C} form a subcategory denoted $\text{Tot}(\mathcal{C})$. This total map category contains all the isomorphisms as well as all the monics.

Recall that a map f is *idempotent* if $ff = f$. It is easily verified that maps of the form \bar{f} are idempotent. If, in addition, f satisfies $f = \bar{f}$, then we say that f is a *restriction idempotent*, or a *domain of definition* where we shall often shorten the latter to simply a *domain*, when the sense is clear. Given an object X , we write $\mathcal{O}(X)$ for the collection of

domains on X . Note that, in general, not every idempotent is a restriction idempotent: for example, in \mathbf{Par} , if X is a set, then every constant total function $f : X \rightarrow X$ is idempotent. But if X has more than one element, such a function is not a domain. Indeed, in this category the domains on X are the partial identity functions or, equivalently, the subsets of X .

Given two objects A and B , we say that A is a *retract* of B if there exist maps $m : A \rightarrow B$ and $r : B \rightarrow A$ such that $rm = 1_A$. Note that the embedding part m , being a section, has to be monic and hence total, but that the retraction r need not be total. The composite mr is an idempotent on B , and the object A is said to be a splitting of the idempotent mr . If A is a retract of B we will denote this by $A \triangleleft B$, or by $(m, r) : A \triangleleft B$ if we wish to indicate a specific embedding-retraction pair. It need not be the case that mr is a restriction idempotent; in the special situation where $mr = \bar{r}$, we will call r the *restriction retraction*, as it is now determined uniquely, and say that m is a *restriction monic*. Thus, restriction monics are exactly those monics which are part of the splitting of a restriction idempotent.

Enrichment. Restriction categories are locally ordered: given parallel maps $f, g : A \rightarrow B$, say that

$$f \leq g \Leftrightarrow f = g\bar{f}.$$

Using the axioms for a restriction it is readily seen that this defines a partial order on the hom-sets of \mathbf{C} which is respected by composition. In \mathbf{Par} , $f \leq g$ means that the graph of f is contained in the graph of g , i.e. that f and g agree whenever $f(x)$ is defined.

In particular, the set $\mathcal{O}(X)$ of domains on an object X is partially ordered via $e \leq e' \Leftrightarrow ee' = e$. This partial order has binary meets given by $e \wedge e' = ee' = e'e$, and the identity on X is the top element; thus, $\mathcal{O}(X)$ is a meet-semilattice.

The assignment $X \mapsto \mathcal{O}(X)$ is the object part of a stable meet-semilattice fibration which is sometimes called the “fundamental functor”. Its action on morphisms is defined by “pullback”: given $f : Y \rightarrow X$, the induced $f^* : \mathcal{O}(X) \rightarrow \mathcal{O}(Y)$ is given by $f^*(e) = \bar{e}f$. This preserves the meets, but not necessarily the top element, since f need not be total.

Splitting Idempotents. Given a category \mathbf{C} and a class E of idempotents in \mathbf{C} , one can formally split the idempotents in E ; the resulting category is typically denoted by $\mathcal{K}_E(\mathbf{C})$. Since we will be using this at more than one occasion, we briefly sketch the construction of $\mathcal{K}_E(\mathbf{C})$. Its objects are pairs (X, e) where X is an object of \mathbf{C} and $e \in E$ is an idempotent on X . A morphism $(X, e) \rightarrow (Y, f)$ is a map $k : X \rightarrow Y$ in \mathbf{C} for which $ke = k = fk$. Provided E contains the identity arrows, there is a functor $\mathbf{C} \rightarrow \mathcal{K}_E(\mathbf{C})$ sending X to $(X, 1_X)$.

Note that an object (X, e) in $\mathcal{K}_E(\mathbf{C})$ becomes a retract of $(X, 1_X)$ via the morphisms $(e, e) : (X, e) \triangleleft (X, 1_X)$.

One can verify that in the case that \mathbf{C} is actually a restriction category, then so is the splitting $\mathcal{K}_E(\mathbf{C})$. Also, the inclusion functor $\mathbf{C} \rightarrow \mathcal{K}_E(\mathbf{C})$ then preserves the restriction structure. Of special interest is the case where we take E to consist of all idempotents (not only the restriction idempotents); categories in which all idempotents split are called *split*, and we write $\mathcal{K}(\mathbf{C})$ for the result of splitting all idempotents in \mathbf{C} . Finally, any two categories \mathbf{C}, \mathbf{D} for which $\mathcal{K}(\mathbf{C}) \simeq \mathcal{K}(\mathbf{D})$ will be said to be *Morita-equivalent*.

Examples.

- (i) Any category is in particular a restriction category under the definition $\bar{f} = 1$. This means that we can regard categories as a special kind of restriction categories.
- (ii) Any category \mathbf{C} equipped with a stable system of monics \mathcal{M} (see [9] for more information) gives rise to a category of partial maps $\text{Par}(\mathbf{C}, \mathcal{M})$. Such a category is a split restriction category and conversely, every split restriction category is essentially of this form. The example $\text{Par} = \text{Par}(\text{Set}, \mathcal{M})$ with \mathcal{M} =all monics is of this form, as is the category $\text{Par}(\text{Top}, \text{Open})$ of topological spaces and partial continuous maps with open domain.
- (iii) For a “small” example, one can take any set X and consider the monoid of partial endofunctions on X ; this is a one-object restriction category (with restriction inherited from Par). This category is not split; its splitting is the category where the objects are all subsets of X and the morphisms are all partial functions between those.
- (iv) Our last example looks ahead to Section 3.2; it has one single object, \mathbb{N} , and has all partial recursive (=partial computable) functions as morphisms. The domains of this category correspond to the recursively enumerable sets.

Restriction functors. Given two restriction categories \mathbf{C}, \mathbf{D} , a *restriction functor* $F : \mathbf{C} \rightarrow \mathbf{D}$ is an ordinary functor from \mathbf{C} to \mathbf{D} which must satisfy the condition that $F(\bar{f}) = \overline{F(f)}$, i.e. that it preserves domains. It follows that F preserves the enrichment as well, in the sense that $f \leq g$ implies $F(f) \leq F(g)$.

2.2. Cartesian structure

We next discuss possible additional structure on restriction categories, in particular finite products.

The thing to note about products (and limits in general) in a restriction category is that they are not genuine categorical products, but that they are products in a suitable bicategorical sense.

Concretely, an object 1 in a restriction category \mathbf{C} is said to be a *restriction terminal object* if for each object A there is a unique total map $!_A : A \rightarrow 1$, such that $!_1 = 1$; moreover, this family of maps must satisfy the condition that for each $f : A \rightarrow B$ we have $!_B f = !_A \bar{f}$, as in the diagram below.

$$\begin{array}{ccc}
 A & & \\
 \downarrow f & \searrow !_A & \\
 B & \xrightarrow{!_B} & 1
 \end{array}$$

A *partial product* of two objects A, B is an object $A \times B$ equipped with total projections $\pi_A : A \times B \rightarrow A$ and $\pi_B : A \times B \rightarrow B$, such that for each C and each pair of maps $f : C \rightarrow A, g : C \rightarrow B$, there is a unique map $f, g : C \rightarrow A \times B$ with the properties that $\pi_A f, g \leq f, \pi_B f, g \leq g$ and $\overline{f, g} = \overline{f} \bar{g}$.

$$\begin{array}{ccccc}
& & C & & \\
& f \swarrow & \downarrow \langle f, g \rangle & \searrow g & \\
A & \xleftarrow{\pi_A} & A \times B & \xrightarrow{\pi_B} & B
\end{array}$$

Thus, the domain of the pairing $\langle f, g \rangle$ is the intersection of the domains of f and g .

Even though partial products are not real products and a restriction terminal object is not a terminal object, we will simply speak of products and terminal objects.

Definition 2.2 (Cartesian restriction categories and functors).

- (i) A restriction category is called cartesian if it has all binary partial products as well as a restriction terminal object.
- (ii) A restriction functor $F : \mathcal{C} \rightarrow \mathcal{D}$ between cartesian restriction categories is a cartesian restriction functor (or simply a cartesian functor) when the canonical comparison morphisms $\langle F\pi_A, F\pi_B \rangle : F(A \times B) \rightarrow FA \times FB$ and $!_{F1} : F1 \rightarrow 1$ are isomorphisms.

An important example of a cartesian functor is the *global sections functor*: given any cartesian restriction category \mathcal{C} , the global sections functor $\Gamma : \mathcal{C} \rightarrow \mathbf{Par}$ is defined by

$$\Gamma(C) = \mathbf{Tot}(\mathcal{C})[1, C],$$

the collection of total points of C . On morphisms, Γ acts by composition. It is easily seen that this is a cartesian functor.

We note that if \mathcal{C} is a cartesian category then $\mathcal{K}_E(\mathcal{C})$ is again cartesian provided E is closed under products.

3. Turing Categories

We will introduce Turing categories as cartesian restriction categories with a special kind of universal object, called a Turing object. After the basic definition we show that one can normalize the structure, which is of great help in verifying that a given category is a Turing category. After that, we consider a few examples (Subsection 3.2), mainly from recursion theory (after all, Turing categories are supposed to embody an abstract notion of computability, so they had better encompass the traditional, concrete notions). Then we turn to the non-canonicity of Turing objects and their structure (Subsection 3.3), and see how this works in some of the examples. We also describe how different Turing structures on a category are related, and give criteria which allow one to test whether given structure on an object is indeed Turing structure.

Some basic constructions for obtaining new Turing categories from old are sketched in Subsection 3.5. We conclude (Subsection 3.6) by commenting on some of the differences between Turing categories and related concepts from the literature.

3.1. Definition and first properties

We start by giving the central definitions of the paper:

Definition 3.1 (Turing category). *Let \mathcal{C} be a cartesian restriction category.*

- (i) *Given a morphism $\tau_{X,Y} : A \times X \rightarrow Y$, a morphism $f : Z \times X \rightarrow Y$ is said to admit a $\tau_{X,Y}$ -index when there exists a total map $h : Z \rightarrow A$ for which the following diagram is commutative:*

$$\begin{array}{ccc}
A \times X & \xrightarrow{\tau_{X,Y}} & Y \\
\uparrow h \times X & & \nearrow f \\
Z \times X & &
\end{array}$$

In this case, h is called a $\tau_{X,Y}$ -index for f .

- (ii) A morphism $\tau_{X,Y}$ is called a universal application when every $f : Z \times X \rightarrow Y$ admits a $\tau_{X,Y}$ -index. We will also simply say that $\tau_{X,Y}$ is universal.
- (iii) A Turing object in \mathcal{C} is an object A such that for each $X, Y \in \mathcal{C}$, there is a universal application $\tau_{X,Y} : A \times X \rightarrow Y$.
- (iv) The category \mathcal{C} is called a Turing category if it possesses a Turing object.

One way to think of a Turing object A is as an object which plays, for each pair of objects X, Y , the role of a weak exponential of X and Y ; in that light, the universal application morphism $\tau_{X,Y} : A \times X \rightarrow Y$ then acts as an evaluation map. We do not insist that the “exponential transpose” h of f is unique; in the (rather special) case where transposes are unique we say that $\tau_{X,Y}$ is *extensional*.

In the case where $Z = 1$, the terminal object, we find, for each morphism $f : X \rightarrow Y$, a total point $a : 1 \rightarrow A$, such that $\tau_{1,Y}(a \times 1) = f$. We shall then refer to a as a *code* for f .

It is important to realize that, given a particular Turing object, the universal morphisms $\tau_{X,Y}$ are not part of the data; rather, their existence is taken to be a property. Generally, for a Turing object there may be many families $\tau_{X,Y}$ which realize it as a Turing object. Similarly, in a Turing category, we do not take a choice of Turing object to be part of the structure and, indeed, a Turing category may possess many non-isomorphic Turing objects.

Definition 3.2 (Turing structure). *Let \mathcal{C} be a Turing category and A an object of \mathcal{C} .*

- (i) A family of maps $\tau = \{\tau_{X,Y} : A \times X \rightarrow Y \mid X, Y\}$ where X, Y range over all objects of \mathcal{C} will be called an applicative family for A .
- (ii) An applicative family τ is called universal for A when each $\tau_{X,Y}$ is universal, in which case we also say that τ is a Turing structure on A .
- (iii) A pair (A, τ) , where τ is universal for A is called a Turing structure on \mathcal{C} .

The question of how different Turing objects in a given category are related and of how different Turing structures on a given Turing object are related will be investigated in Subsection 3.3.

We should give some examples at this point, but as one can imagine, verifying whether a given category is a Turing category using the above definition may not be straightforward. We therefore postpone this until we have gathered a few results which will allow for an equivalent formulation which is easier to check in practice.

As a first observation, we note that a Turing object is a *universal object*, in the strong sense that every object is a retract of it.

Lemma 3.3. *In a Turing category \mathcal{C} with Turing object A , every object X is a retract of A .*

Proof. Consider the diagram

$$\begin{array}{ccc}
A \times 1 & \xrightarrow{\tau_{1,X}} & X \\
\widetilde{\pi}_X \times 1 \uparrow & & \nearrow \pi_X \\
X \times 1 & &
\end{array}$$

where $\widetilde{\pi}_X$ is a $\tau_{1,X}$ -index for π_X . Precomposing both ways around with the isomorphism $X \rightarrow X \times 1$ gives the desired result. \square

In particular we have that all finite powers $1, A, A^2, A^3, \dots$ are retracts of A . This example not only indicates something about the size of the object A (which, if we were working over the category of sets, would have to be either a singleton or infinite) but also tells us that there is an internal pairing operation on A .

We next turn to the problem of normalizing the Turing structure in the sense of showing how such structure can be generated from some key components. One of these key ingredients is a *Turing morphism*: this is a universal self-application map, which we shall write as $A \times A \xrightarrow{\bullet} A$. Clearly, given a universal applicative family $\tau = \{\tau_{X,Y}\}$ on A , the map $\tau_{A,A} : A \times A \rightarrow A$ is such a Turing morphism. However, simply having such a Turing morphism is not sufficient to generate a Turing structure on A ; one needs the additional requirement that A is a universal object.

Let X, Y be arbitrary objects, and suppose we seek to construct a universal application map $\tau_{X,Y} : A \times X \rightarrow Y$. Let us assume that we have embedding-retraction pairs $(m_X, r_X) : X \triangleleft A$ and $(m_Y, r_Y) : Y \triangleleft A$. We define $\tau_{X,Y}$ to be the composite

$$A \times X \xrightarrow{A \times m_X} A \times A \xrightarrow{\bullet} A \xrightarrow{r_Y} Y.$$

To verify that this morphism is a universal application suppose we are given $f : Z \times X \rightarrow Y$. Consider the diagram

$$\begin{array}{ccccc}
A \times X & \xrightarrow{A \times m_X} & A \times A & \xrightarrow{\bullet} & A & \xrightarrow{r_Y} & Y \\
\uparrow h \times X & & \uparrow h \times A & & \uparrow m_Y f & & \nearrow f \\
Z \times X & \xrightarrow{Z \times m_X} & Z \times A & \xrightarrow{Z \times r_X} & Z \times X & &
\end{array}$$

The triangle on the right commutes because of the equality $r_Y m_Y = 1$. The square in the middle uses the fact that \bullet is a Turing morphism and hence that there is an index h for the composite $m_Y f (Z \times r_X)$. Obviously the left hand square commutes, and the bottom composite is simply the identity. Thus h is also an index for f relative to the top composite.

Since we have constructed all Turing morphisms from \bullet under the assumption that every object is a retract of A , we have thus proved:

Theorem 3.4 (Recognition criterion for Turing categories). *For a cartesian category \mathcal{C} , the following are equivalent:*

- (i) \mathcal{C} is Turing category;
- (ii) There is an object A of which every object is a retract, and for which there exists a Turing morphism $A \times A \xrightarrow{\bullet} A$.

We have to warn the reader that the above way of constructing the morphisms $\tau_{X,Y}$ from the single instance $\bullet = \tau_{A,A}$ is not canonical, and that there may in general be many ways of extending $\tau_{A,A}$ to a Turing structure on A .

3.2. Examples

We now present a few basic examples of Turing categories, ranging from the Turing category for classical recursion theory and models of the lambda calculus to less well-known examples.

3.2.1. The classical recursion category

We first recall some concepts and notation from elementary recursion theory. First of all, one may consider a suitable enumeration ϕ_0, ϕ_1, \dots of the partial recursive functions $f : \mathbb{N} \rightarrow \mathbb{N}$ (of one variable), based on the fact that such a function can be described by finitely many instructions, and that these instructions can be coded into one single number. Alternatively, one may consider a coding for Turing machines, since the Turing machine-computable functions are exactly the partial recursive ones. Similarly, there exist enumerations of partial recursive functions of several variables; $\phi_0^{(n)}, \phi_1^{(n)}, \dots$ will be an enumeration of the n -ary partial recursive functions. When $f = \phi_e$, one also says that the number e is a *code* for f .

The key properties one can prove about such a family $\phi_m^{(n)}$ (and this is where one needs the enumeration to be “suitable”) are:

- (i) **The Universality Theorem.** This result says, that the functions (for each $n > 0$)

$$\Phi^{(n)}(e, x_1, \dots, x_n) = \phi_e(x_1, \dots, x_n),$$

called the *universal functions*, are partial recursive.

- (ii) **The Parameter Theorem.** This says that there are primitive recursive functions S_m^n , for each $n, m > 0$, such that

$$\Phi^{(n+m)}(e, x_1, \dots, x_m, u_1, \dots, u_n) = \Phi^{(m)}(S_m^n(e, x_1, \dots, x_m), u_1, \dots, u_n).$$

Moreover, the S_m^n functions may be taken to be injective.

(In some texts, the Universality Theorem is taken to be part of the Enumeration Theorem. Moreover, some texts take the validity of the above two theorems to be the *definition* of a suitable enumeration.) We now fix such an enumeration and use it to define the so-called *Kleene-application* on the natural numbers; this is the function $\mathbb{N} \times \mathbb{N} \xrightarrow{\bullet} \mathbb{N}$, defined by

$$n \bullet x = \phi_n(x).$$

That is, $n \bullet m$ is the result of applying the n -th partial recursive function to input m (and is undefined whenever that computation diverges).

Having this at our disposal, consider the category whose objects are the finite powers of \mathbb{N} , and where a morphism $\mathbb{N}^k \rightarrow \mathbb{N}^m$ is an m -tuple of partial recursive functions of k variables. In case $k = 0$, we simply get m -tuples of elements of \mathbb{N} . Denote this category by $\text{Comp}(\mathbb{N})$.

The object \mathbb{N} is a Turing object in this category. First, any (primitive) recursive pairing function $p : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$ together with unpairing functions $p_0, p_1 : \mathbb{N} \rightarrow \mathbb{N} \times \mathbb{N}$ will exhibit $\mathbb{N} \times \mathbb{N}$ as a retract of \mathbb{N} (in fact, these objects are isomorphic), and similarly for the other objects.

The application on \mathbb{N} is Kleene-application; the Universality Theorem guarantees that this is a partial recursive map, and the weak universal property of \bullet is now an application of the Parameter Theorem.

Note that all of the universal functions are in fact Turing morphisms in this category. In the light of this example, we thus arrive at a justification for the axioms of a Turing category; they are supposed to give a minimal setting in which the Universality Theorem and the Parameter Theorem hold. (The fact that classically the reparametrization functions are primitive recursive and not just merely total recursive does not have a direct analogue in a general Turing category; the fact that they may be taken to be injective does - see Section 3.3.)

3.2.2. Variation

A minor variation on the above category is obtained by considering as objects all recursively enumerable (r.e.) subsets of \mathbb{N}^k , $k \geq 0$, and (tuples of) partial recursive functions between those. Again, \mathbb{N} , equipped with the Kleene-application, is a Turing object in this category. (This category is in fact obtained from the first example by splitting the class of restriction idempotents, see Subsection 3.5.)

One could also consider a “smaller” version of the classical recursion category $\mathbf{Comp}(\mathbb{K}_1)$, namely by taking only the objects 1 and \mathbb{N} (and all computable maps between them). Since $\mathbb{N} \cong \mathbb{N} \times \mathbb{N}$ via a computable isomorphism, this is actually a skeleton of $\mathbf{Comp}(\mathbb{K}_1)$. One recovers the old category by splitting the idempotents which correspond to the products.

3.2.3. Reflexive objects

A *reflexive object* in a cartesian closed category is an object A together with embedding-retraction pairs $1 \triangleleft A$, $A \times A \triangleleft A$, $A^A \triangleleft A$. It is well-known that such an object is a model of untyped lambda calculus (see, e.g. [23]). A category where the search for non-trivial reflexive objects often leads to success is that of (directed-)complete partial orders (CPOs) and Scott-continuous maps. Given a reflexive object A in that category, one may consider the smallest cartesian closed subcategory generated by A . This is a Turing category with Turing object A ; the Turing morphism is defined by taking an embedding-retraction pair $(m, r) : A^A \triangleleft A$, and by defining

$$\bullet_A : A \times A \xrightarrow{r \times A} A^A \times A \xrightarrow{ev} A.$$

This category only has total maps; although one might initially expect a treatment of recursion theory to have partiality (as Di Paola and Heller insisted), this limiting case is not excluded and, indeed, we wish to make a point here that the study of categorical models of the untyped lambda calculus may be viewed as the study of cartesian closed (total) Turing categories.

Also, we note that, more generally, one could work in a *partial* cartesian closed Turing category (for more on this notion, see, e.g. [13], or [46]), and look for reflexive objects in there; those correspond to models of the untyped *partial* lambda calculus (see [32]). Hence in general cartesian closed Turing categories will correspond to partial lambda algebras.

3.2.4. A non-example

One could wonder whether our first example, $\mathbf{Comp}(\mathbb{N})$, has any interesting Turing subcategories (subcategories which are also Turing categories). One might consider, for example, the subcategory of $\mathbf{Comp}(\mathbb{N})$ on the total maps, i.e. the total recursive functions,

or maybe even on the primitive recursive functions. However, that cannot work since, by diagonal arguments, there cannot exist suitable enumerations for these classes of functions (see, e.g., [36]).

3.2.5. *Classifying categories*

So far, our examples were set-based, in the sense that the Turing object was always a set equipped with some extra data (or in the more precise sense that the Turing categories were faithful over \mathbf{Par}). Thus, they embody set-based notions of computation. We will now sketch an example which is quite different.

The theory *Partial Combinatory Logic* (PCL, for short) is a untyped partial algebraic theory which has two constants \mathbf{k} and \mathbf{s} , and a binary application symbol \bullet . From these (and variables of course), partial terms are formed; the term formation rules include the usual rules, but there is the added rule of *term restriction*: given terms t, r , one may form a new term $t|_r$, to be thought of as t restricted to (the domain of) r . The PCL axioms are $\mathbf{k}xy = x$, $\mathbf{s}xyz = xz(yz)$ and $x|_{sxy} = x$, where the application is written as an invisible, left associating, operation. The first two axioms are the usual axioms for the combinators \mathbf{k} and \mathbf{s} and the last axiom is to be read as “ sxy is total”. The models of this theory are the partial combinatory algebras which will be discussed in Section 4.

Now for ordinary equational theories one can always consider a *term model*, which is built from the syntax of the theory and which is an initial model. In the case of a partial theory such as PCL, this is replaced by a *classifying category*: this is a cartesian restriction category containing a generic model. It follows from the specific properties of PCL (combinatory completeness) that the classifying category for PCL is a Turing category. Because of its initiality, it may be regarded as a generic model of computation.

3.2.6. *Turing categories vs. Kleene categories*

There is a variation on the definition of Turing category, in which one demands that every morphism f now has an index h *up to inequality*, in the sense that $f \leq \bullet(h \times 1)$ instead of the usual $f = \bullet(h \times 1)$. Such an h is then called a realizer for f . One is tempted to call such structure a Kleene category, since they are more suited for purposes of realizability than for computability (this will be discussed in [11]).

Many things which are true for Turing categories are already true for Kleene categories. For example, in a Kleene category each object is a retract of the Kleene object, and hence one can normalize the structure. However, there are some important differences: some recursion theoretic concepts, such as m -completeness (see Section 3.4), make sense in Kleene categories, but basic results about them break down.

3.3. *Comparing Turing structures*

We remarked earlier that a given Turing category may have many different Turing structures. We now investigate how different Turing structures on a Turing category are related, and give some criteria for verifying whether certain data induces Turing structure. Categorically, this may be viewed as a coherence issue; on the other hand it immediately leads to the subject of simulations, and is closely related to the classical concept of recursive invariance.

Let us fix a Turing category \mathcal{C} . All we know is that there exists a Turing object, say A . But already from A , we can obtain more Turing objects, namely the powers A^2, A^3, \dots . This follows directly from the following Lemma, which gives a criterion for an object to be a Turing object.

Lemma 3.5. *Let A be a Turing object in \mathcal{C} . Then an object B is a Turing object if and only if A is a retract of B .*

Proof. Clearly, by Lemma 3.3, the condition is necessary. Conversely, let us assume that A is a retract of B , and let us construct universal application morphisms for B . Define $\sigma_{X,Y} : B \times X \rightarrow Y$ as the composite

$$\sigma_{X,Y} : B \times X \xrightarrow{r \times X} A \times X \xrightarrow{\tau_{X,Y}} Y$$

where the map r is a retraction of B onto A , and where $\tau_{X,Y}$ is a universal application for A . Given $f : Z \times X \rightarrow Y$, we now can consider an index $h : Z \rightarrow A$ for f relative to A , and compose this index with a splitting of r . \square

Thus, all Turing objects in a Turing category are retracts of each other. From this, it does not, in general, follow that they are isomorphic - this requires (a computable version of) the Cantor-Schröder-Bernstein Theorem.

Even though the above Lemma is, as a coherence result, quite crude, in some cases it still gives us an easy way to determine which objects in a given Turing category are Turing objects. For example, consider the category where the objects are r.e. subsets of powers of \mathbb{N} , and where the morphisms are the partial recursive functions between those (example 3.2.2). Of course, \mathbb{N} is a Turing object, as are all powers of \mathbb{N} . Moreover, it is known (see [36]) that every infinite r.e. set U can be viewed as the range of an injective total recursive function, and this function has a recursive section, exhibiting \mathbb{N} as a retract of U . Thus, every such infinite U is a Turing object as well. Obviously a finite set cannot be a Turing object, which leads us to the conclusion that the Turing objects are exactly the infinite r.e. sets.

Now let us suppose that we have two Turing objects A and B , together with chosen structure $\tau = \{\tau_{X,Y} | X, Y\}$ on A and $\sigma = \{\sigma_{X,Y} | X, Y\}$ on B . How are τ and σ related? Well, we first pick an embedding-retraction pair $(m, r) : A \triangleleft B$. Then consider the diagram

$$\begin{array}{ccc} B \times X & \xrightarrow{\sigma_{X,Y}} & Y \\ \uparrow h \times X & & \uparrow 1 \\ A \times X & \xrightarrow{m \times X} B \times X \xrightarrow{r \times X} A \times X \xrightarrow{\tau_{X,Y}} & Y \end{array}$$

The arrow h is a σ -index for the composite $\tau_{X,Y}(r \times X)$. Since $rm = 1$, this gives the equation

$$\tau_{X,Y} = \sigma_{X,Y} \circ (hm \times X).$$

Thus, the map hm relates the two application morphisms; because m was just a choice of embedding and h was just any index, no coherence (varying X and Y) can be expected.

Of course, we may reverse the roles of A and B , and then we will get a comparison morphism in the other direction: given an embedding-retraction pair $(n, t) : B \triangleleft A$, we find j such that

$$\sigma_{X,Y} = \tau_{X,Y} \circ (jn \times X).$$

We may turn the above observation into a criterion for when a family $\sigma_{X,Y} : B \times X \rightarrow Y$ defines Turing structure on B .

Lemma 3.6. *Let A be a Turing object with universal maps $\tau_{X,Y}$. A family $\sigma_{X,Y} : B \times X \rightarrow Y$ constitutes a Turing structure on B if and only if there exist total maps $p_{X,Y} : A \rightarrow B$ such that $\tau_{X,Y} = \sigma_{X,Y}(p_{X,Y} \times X)$.*

Proof. One direction was proved in the preceding discussion. For the converse, suppose such a family $p_{X,Y}$ is given. Then, for a morphism $f : Z \times X \rightarrow Y$, one finds a σ -index by first finding a τ -index $\tilde{f} : X \rightarrow A$ and then composing with p . \square

We can also use this to give a criterion for when a morphism $B \times B \xrightarrow{\bullet} B$ is a Turing morphism.

Lemma 3.7. *Let A be a Turing object and let $\tau_{B,B}$ be a universal application map. Then a morphism $B \times B \xrightarrow{\bullet} B$ is a Turing morphism precisely when there is a total map $p : B \rightarrow A$ making the following diagram commutative.*

$$\begin{array}{ccc} A \times B & \xrightarrow{\tau_{B,B}} & B \\ p \times B \uparrow & \nearrow & \bullet \\ B \times B & & \end{array}$$

Proof. Immediate. \square

3.4. Comparing Turing morphisms

We now know that different Turing objects must be retracts of each other, and that between different universal applicative families there must always be comparison morphisms. To complete the description of the coherence among different Turing structures, we would like to give characterizations in terms of the normalized structures. This is certainly possible, but in order to give a complete account one needs both the language and the results about simulations (Section 5). We therefore present a few simple observations here which can be stated in elementary terms, and illustrate these by connecting them to recursion-theoretic phenomena. As a corollary, we show that the Halting set is m-complete in any Turing category.

Let us first illustrate how it can happen that a given Turing object A has different Turing morphisms. Suppose we have picked a Turing morphism $A \times A \xrightarrow{\bullet} A$ on A ; then we can change this into another application \bullet' by considering the following composite:

$$A \times A \xrightarrow{r \times A} (A \times A) \times A \xrightarrow{\pi_0 \times A} A \times A \xrightarrow{\bullet} A.$$

Here, we have chosen an embedding-projection pair $(m, r) : A \times A \triangleleft A$. Now \bullet' is again a Turing morphism: given $f : X \times A \rightarrow A$, consider the diagram

$$\begin{array}{ccccc} A \times A & \xrightarrow{r \times A} & (A \times A) \times A & \xrightarrow{\pi_0 \times A} & A \times A \xrightarrow{\bullet} A \\ h \times A \uparrow & & \uparrow \langle g, 1 \rangle \times A & \nearrow g \times A & \nearrow f \\ X \times A & \xrightarrow{m_X \times A} & A \times A & \xrightarrow{r_X \times A} & X \times A \end{array}$$

where g is an index of $f(r_X \times A)$ relative to \bullet , and where we have chosen $(m_X, r_X) : X \triangleleft A$. Then h may be defined as the composite

$$h = m\langle g, 1 \rangle m_X,$$

since then

$$\begin{aligned} \bullet(\pi_0 \times A)(r \times A)(h \times A) &= \bullet(\pi_0 \times A)(r \times A)((m\langle g, 1 \rangle m_X) \times A) \\ &= \bullet(\pi_0 \times A)(rm\langle g, 1 \rangle \times A)(m_X \times A) \\ &= \bullet(\pi_0 \times A)(\langle g, 1 \rangle \times A)(m_X \times A) \\ &= \bullet(g \times A)(m_X \times A) \\ &= f(r_X \times A)(m_X \times A) \\ &= f \end{aligned}$$

Observe that this new index h is in fact a section, and thus in particular a monomorphism. Thus we have in fact shown that every Turing category possesses a Turing structure for which every map has a monic index.

Remark. In classical recursion theory one often exploits the scholium to the Parameter Theorem that the reparametrization functions may be taken to be injective. The construction is also related to what is sometimes called ‘‘Padding’’: the fact that, given a particular code for a function, one can effectively generate infinitely many more codes (see [36]). Now in a general Turing category this result will not hold for every Turing structure (for example if the structure is extensional). However, the above observation shows that there always exists a Turing structure *on the same Turing object* for which padding does work (provided the category is non-trivial of course).

This example tells us more than just about the possibility of creating different Turing morphisms: it tells us that certain properties (in this case, having monic indices) are not properties of the Turing category, but are properties of specific choices of structure.

Before we look at the relation between different Turing morphisms, we introduce a bit of notation which will also prove useful later on. Let us be given a Turing morphism \bullet on A . From this, we may define a family of iterated application morphisms $\bullet^{(n)} : A \times A^n \rightarrow A$. These are defined inductively: $\bullet^{(1)} = \bullet$, and if we have defined $\bullet^{(n)}$, then let $\bullet^{(n+1)}$ be the composite

$$A \times A \times A^n \xrightarrow{\bullet \times 1} A \times A^n \xrightarrow{\bullet^{(n)}} A.$$

Thus, $\bullet^{(n)}$ is n -fold application, associated to the left. In case no confusion is possible we will be sloppy and omit the parameter n from the notation and simply write \bullet for all of the members of this family.

Lemma 3.8. *Each of the derived morphisms $\bullet^{(n)}$ is a universal application.*

Proof. This is proved by induction on n , where the case $n = 1$ is true by assumption that \bullet is a Turing morphism. If we have proved the statement for $n = k$, then consider $f : Z \times A^{k+1} \rightarrow A$, and construct an index for f according to

$$\begin{array}{ccccc}
A \times A \times A^k & \xrightarrow{\bullet \times 1} & A \times A^k & \xrightarrow{\bullet^k} & A \\
h \times A^k \uparrow & & g \times A^k \uparrow & \nearrow f & \\
X \times A \times A^k & \xrightarrow{=} & X \times A \times A^k & &
\end{array}$$

where $g : X \times A \rightarrow A$ is a $\bullet^{(k)}$ -index for f (which exists by induction hypothesis), and $h : X \rightarrow A$ is a \bullet -index for g . □

We have started the inductive definition of $\bullet^{(n)}$ at $n = 1$, and one may ask what $\bullet^{(0)}$ should be! Given that the morphisms $\bullet^{(n)}$ allow us to view elements of A as n -ary morphisms, a natural choice would be to let $\bullet^{(0)}$ be the identity, for then we regard an element of A as a 0-ary function, that is, a constant. However, there are compelling reasons for making another choice: we define $\bullet^{(0)}$ to be the composite

$$A \xrightarrow{\Delta} A \times A \xrightarrow{\bullet} A.$$

A first argument in favour of this choice is that many of the interesting properties of the family $\bullet^{(n)}$ are shared by the above map, but not by the identity. (More reasons will be provided in Section 4.) For example, it is also a universal application: given $f : X \rightarrow A$, let $d : X \rightarrow A$ be a \bullet -index for $f \pi_X : X \times A \rightarrow A$. Then d is also a δ -index for f . See the following diagram.

$$\begin{array}{ccccc}
A & \xrightarrow{\Delta} & A \times A & \xrightarrow{\bullet} & A \\
d \uparrow & & d \times A \uparrow & & \uparrow f \\
X & \xrightarrow{(1,d)} & X \times A & \xrightarrow{\pi_X} & X
\end{array}$$

We digress for a moment to note that we have actually proved a result from recursion theory here, namely the m-completeness of the Halting set. Let us briefly explain this: if two domains $e \in \mathcal{O}(X), e' \in \mathcal{O}(Y)$ are given, we say that $e \leq_m e'$ (in words: e many-one reduces to e') when there is a total map $f : X \rightarrow Y$ such that $e = f^*(e')$, i.e. e is the pullback of e' along f . A domain is said to be m-complete if every other domain m-reduces to it⁵.

Now for any Turing structure $\tau = \{\tau_{X,Y}\}$ on A , the domain of $\tau_{1,A}$ is indeed m-complete: given a domain $e \in \mathcal{O}(X)$, consider

$$\begin{array}{ccc}
A & \xrightarrow{\tau_{1,A}} & A \\
h \uparrow & & \uparrow m \\
X & \xrightarrow{e} & X
\end{array}$$

where $m : X \rightarrow A$ is a choice of embedding and h is an index for me . Because $me = \tau_{1,A}h$, the domains of $\tau_{1,A}h$ and of me are equal. Thus, letting K (so $K = \cdot$, we get

$$e = \overline{me} = \overline{\tau_{1,A}h} = h^*(K),$$

by totality of m , which shows that $e \leq_m K$.

Now we may apply this to the domain of the morphism δ ; this gives:

⁵ A more familiar categorical formulation of an m-complete predicate would be: e is m-complete if and only if it is a weak generic object in the appropriate subobject fibration (over the category of total maps).

Corollary 3.9 (Completeness of the halting set). *The domain K is m -complete.*

Similar reasoning shows the m -completeness of other domains. Indeed, if we take any total point $a : 1 \rightarrow A$ of A , then we may consider the composite $A \xrightarrow{\langle 1, a \rangle} A \times A \xrightarrow{\bullet} A$. The domain of this map will then be m -complete, for if we are given a domain e on X , consider

$$\begin{array}{ccccc} A & \xrightarrow{\langle 1, a \rangle} & A \times A & \xrightarrow{\bullet} & A \\ \uparrow h & & \uparrow h \times A & & \uparrow m \\ X & \xrightarrow{\langle 1, a \rangle} & X \times A & \xrightarrow{e\pi_X} & X \end{array}$$

where h is an index for $m\pi_X$. Since h is total and the bottom composite is e , this shows that e is a pullback of the domain of the top composite.

One can strengthen this to 1-completeness by first modifying the Turing morphism as to ensure that indices are monic.

We return to the problem of relating different Turing morphisms. Suppose we have two Turing objects A, B , with specified Turing morphisms $\bullet_A = \tau_{A,A}$ and $\bullet_B = \sigma_{B,B}$. Choose a retraction $(m, r) : A \triangleleft B$. First, consider the diagram

$$\begin{array}{ccccc} B \times B & \xrightarrow{\bullet_B} & B & & \\ \uparrow h \times B & & \uparrow m & & \\ A \times A & \xrightarrow{m \times m} & B \times B & \xrightarrow{r \times r} & A \times A \xrightarrow{\bullet_A} A \end{array}$$

This gives an equation

$$\bullet_B(hm \times m) = m\bullet_A$$

relating the two Turing morphisms. Written in the more palatable internal language of the category (using infix notation for the applications), this reads

$$hm(x) \bullet_B y = m(x \bullet_A y).$$

Thus, m preserves the application modulo the “twist” h .

Remark. In recursion theory, one considers two acceptable systems of indices (enumerations) ϕ_0, ϕ_1, \dots and ψ_0, ψ_1, \dots , or the partial recursive functions, giving rise to Kleene applications \bullet_ϕ and \bullet_ψ , respectively. Then one asks how these are related. One answer is that there is a total recursive function h for which

$$h(x) \bullet_\psi y = x \bullet_\phi y.$$

(The embedding m which figures in our equation is the identity here.) In addition, one can show that h may be taken to be an isomorphism. It is the latter aspect which is specific to classical recursion theory and does not generalize to our setting. Also, it has been shown that there also must exist a total recursive g such that $g(x) \bullet g(y) = g(x \bullet y)$ (see [36]); again, this exploits specific properties of the natural numbers and does not generalize.

We can reformulate the relation between \bullet_A and \bullet_B using the iterated application $\bullet_B^{(2)}$. Indeed, consider

$$\begin{array}{ccccc}
& & B \times B \times B & \xrightarrow{\bullet_B^{(2)}} & B \\
& & \uparrow u \times B \times B & & \uparrow m \\
A \times A & \xrightarrow{m \times m} & B \times B & \xrightarrow{r \times r} & A \times A \xrightarrow{\bullet_A} A
\end{array}$$

where $u : 1 \rightarrow B$ is now a code. Written in the internal language again, this means that

$$(u \bullet_B m(x)) \bullet_B m(y) = m(x \bullet_A y).$$

We may summarize this in the following Lemma:

Lemma 3.10. *If (A, \bullet_A) and (B, \bullet_B) are both Turing objects, then there are codes $u : 1 \rightarrow A$ and $v : 1 \rightarrow B$, and morphisms $m : A \rightarrow B$ and $n : B \rightarrow A$ such that the following diagrams commute:*

$$\begin{array}{ccc}
A \times A \xrightarrow{\bullet_A} A & & B \times B \xrightarrow{\bullet_B} B \\
\langle u, m, m \rangle \downarrow & & \langle v, n, n \rangle \downarrow \\
B \times B \times B \xrightarrow{\bullet_B} B & & A \times A \times A \xrightarrow{\bullet_A} A
\end{array}$$

In Section 5 we will see that the maps m and n are (*strict*) *simulations* between (A, \bullet_A) to (B, \bullet_B) and, moreover, explain the sense in which this pair of simulations constitute an equivalence between A and B . Thus, we may recast the above by saying that any two of Turing morphisms in a Turing category are necessarily equivalent. In Section 5 we will make all of this more precise, strengthen this result and use it to formulate a criterion for when a given morphism is in fact a Turing morphism.

3.5. Elementary constructions

In this section we collect a few basic constructions which enable us to form new Turing categories from old. Since most of the constructions are well-known, we only provide the general idea, leaving routine verifications to the reader.

Splitting Idempotents. The process of formally splitting a given class of idempotents is not only a way to make certain domains into actual subobjects, but will also turn out to play an important role in the structural results about Turing categories. Let \mathcal{C} be a Turing category and let E be a class of idempotents in \mathcal{C} . The idempotents in E need not be restriction idempotents, but it is crucial that all identities are present. Then $\mathcal{K}_E(\mathcal{C})$, the category obtained from \mathcal{C} by formally splitting all idempotents in the given class E , is again a Turing category.

We sketch the proof; see Section 2 for the notation concerning the category $\mathcal{K}_E(\mathcal{C})$. One wants to show that if A was a Turing object with application $\bullet : A \times A \rightarrow A$, then the image \bullet under the inclusion of \mathcal{C} into $\mathcal{K}_E(\mathcal{C})$ is a Turing object in the latter category. Now a morphism $f : (X, e) \times (A, 1) \rightarrow (A, 1)$ is in particular a map $f : X \times A \rightarrow A$ in \mathcal{C} and so has an index $\tilde{f} : X \rightarrow A$. But then $\tilde{f}e : (X, e) \rightarrow (A, 1)$ is an index for $f : (X, e) \times (A, 1) \rightarrow (A, 1)$.

To see that every object of $\mathcal{K}_E(\mathcal{C})$ is a retract of $(A, 1)$, simply note that every object (X, e) is a retract of $(X, 1)$; as object of \mathcal{C} , X is a retract of A , and this remains to be the case in $\mathcal{K}_E(\mathcal{C})$.

Congruences and quotients. A *cartesian congruence* on a cartesian restriction category is a congruence \sim on \mathbf{C} such that the quotient category \mathbf{C}/\sim is again cartesian and the canonical functor $\mathbf{C} \rightarrow \mathbf{C}/\sim$ preserves the cartesian structure. Explicitly, that means that the congruence should not only respect composition, but also the restriction and pairing, that is $f \sim g \Rightarrow \bar{f} \sim \bar{g}$, and $f \sim g, h \sim k \Rightarrow \langle f, h \rangle \sim \langle g, k \rangle$.

Now suppose that \mathbf{C} happens to be a Turing category. Then it is readily verified that \mathbf{C}/\sim is again a Turing category, and the quotient functor preserves Turing structures.

There are two types of congruences which are of special interest to us: the first arises from considering a filter in the lattice of domains in the terminal object. Whenever $\mathcal{F} \subseteq \mathcal{O}(1)$ is such a filter (meaning: it is upwards closed and closed under binary meets) then we may define a congruence on the category by declaring two maps $f, g : C \rightarrow D$ to be congruent whenever there is an element $u : 1 \rightarrow 1$ in the filter \mathcal{F} such that $f \times u = g \times u : C \times 1 \rightarrow D \times 1$. The resulting quotient category will be denoted \mathbf{C}/\mathcal{F} .

Lemma 3.11. *Let \mathbf{C} be a Turing category and let \mathcal{F} be a filter in $\mathcal{O}(1)$. Then \mathbf{C}/\mathcal{F} is a Turing category. If \mathbf{C} has a zero and \mathcal{F} is a maximal filter then \mathbf{C}/\mathcal{F} is a two-valued Turing category.*

This result may be used to collapse a Turing category which has “too many subterminal domains” so as to obtain a Turing category which has at most 2 subterminals.

Not unrelated is the congruence on \mathbf{C} which is defined by declaring parallel arrows $f, g : C \rightarrow D$ to be equivalent provided they have the same composites with all morphisms $1 \rightarrow C$. The resulting quotient category is now generated by the terminal object: distinct morphisms can always be distinguished by evaluating at a (possibly partial) point.

We can be less subtle and, instead of all points $1 \rightarrow C$, consider only those which are total. This will result in a more drastic collapse of the category onto a well-pointed category. Because of the fact that well-pointed categories are faithful over the category \mathbf{Par} (they are concrete), this construction amounts to factorizing the global sections functor $\Gamma : \mathbf{C} \rightarrow \mathbf{Par}$ as $\mathbf{C} \rightarrow \mathbf{C}/wp \rightarrow \mathbf{Par}$, where the latter half is faithful.

The above examples of congruences arose from general categorical considerations, but there are also many examples specific to computability. In Section 3.2 we discussed the classifying category of partial combinatory logic; congruences on this category correspond to extensions of combinatory logic. For example, we may add a certain new axiom $t_1 = t_2$ to CL, and this will induce an equivalence relation on the terms, refining the existing relation of provable equality. Since the classifying category has equivalence classes of terms as its morphisms, this will now form a congruence. The quotient category is then a classifying category for the strengthened theory.

Factorization. We are often interested not just in Turing categories per se, but in Turing categories over a fixed base category. The idea behind that is, that the base category represents our current universe of mathematics, while a Turing category over such a universe represents a notion of computation which has been extracted (this idea will be made more concrete in Section 4). The situation where a Turing category has a faithful functor down into the base category is especially important; hence, the following result is often useful.

Lemma 3.12. *Let a Turing category \mathbf{C} be given, together with a cartesian functor $F : \mathbf{C} \rightarrow \mathbf{D}$. Factor F as a composite $\mathbf{C} \xrightarrow{F_0} \mathbf{C}_0 \xrightarrow{F_1} \mathbf{D}$ where \mathbf{C}_0 is the quotient category obtained by identifying maps which have the same image under F . Then \mathbf{C}_0 is again a*

Turing category.

Proof. This follows from the result on congruences. \square

We will often use this in the following form (“Change of base”): given a cartesian faithful functor F from a Turing category \mathbb{C} into the base \mathbb{D} , and a cartesian functor $H : \mathbb{D} \rightarrow \mathbb{D}'$, one may factor the composite HF in the above manner; then the image of HF is a Turing category over \mathbb{D}' .

3.6. Remarks

As explained in the introduction, Turing categories directly generalize the categories studied by Longo and Moggi in [31]. While they focus on total structures (and only briefly mention the possibility of developing the partial case) we have taken the partial morphisms as the basis of the approach, obtaining their categories as a special case. Also, it will be seen in the following sections that we often wish to regard Turing categories as living over a certain base category, or Turing categories which are not concrete, while Longo and Moggi seemed to consider only concrete categories (corresponding to combinatory algebras in \mathbf{Set}). We wish to stress that this extra generality is actually important for a full understanding of the matters; for example, the generic Turing category (Example 3.2.5) is not concrete over \mathbf{Set} . One may summarize the difference between Longo and Moggi’s categories and Turing categories by saying that the former are Turing categories which (1) are concrete, (2) are total and (3) correspond (in the way explained in the next section) to absolute notions of computation, as opposed to relative notions.

We also mentioned in the introduction that the recursion categories by Di Paola and Heller [38] differ from Turing categories in that they have zero morphisms (forcing non-trivial partiality) and must have all objects isomorphic. We have dropped both requirements as they seem inessential to the first stages of the development and exclude quite a few examples. In conclusion, a recursion category is (up to Morita-equivalence) a Turing category without a terminal object, with zero morphisms and with the special property that it has surjective pairing.

We have chosen not to adopt the name “recursion category” because of the technical differences, and to stress the fact that Turing categories are good for more than only recursion. A case could be made for the terminology in [31], where universal application maps are called Kleene-universal maps. However, we decided to name all the strict concepts after Turing, while naming the relaxed versions (which, after all, are the ones used in realizability) after Kleene.

Birkedal’s “weakly closed partial cartesian categories with universal object” are essentially the same as Turing categories, although the underlying framework is that of \mathbf{p} -categories. As stated earlier, Birkedal proves that such structure is sufficiently rich to build a tripos and hence a topos. Our results in the next sections will imply that this is always a relative realizability topos.

Partial combinatory logic (PCL), which was briefly mentioned as an example (3.2.5) is an instance of a partial equational theory; such theories are based on the term logic for cartesian restriction categories. Logics of this kind have been studied in various guises and levels of generality (see [32,37,12,45], just to name a few). One may consider models

of PCL in any cartesian restriction category, and a model will be a Partial Combinatory Algebra (PCA), to be discussed in the next section.

The classifying category of PCL seems to have escaped attention so far, even though it has various interesting structural properties. These will be reported on in future work.

4. Partial Combinatory Algebras

From the few examples in the previous section, as well as the recognition theorem, we saw that Turing categories are, in a certain sense, generated from a single object which comes equipped with a binary application map. In this section we first introduce partial combinatory algebras (Subsection 4.1) and the way in which they generate Turing categories. Indeed, every Turing object is an internal PCA in the ambient Turing category. It turns out that in order to obtain all Turing categories, one needs a form of relative computability (Subsection 4.3). This sets up a correspondence between Turing categories over a fixed base category and pairs of PCAs in the base category.

4.1. Applicative systems and combinatory completeness

Throughout this section, we will work in an arbitrary cartesian restriction category \mathcal{C} . **Definition 4.1** (Applicative system). *An applicative system $\mathbb{A} = (A, \bullet)$ in \mathcal{C} consists of an object A together with a morphism $A \times A \xrightarrow{\bullet} A$, called application.*

There are no axioms to be satisfied, and therefore this notion is rather wide. In order to select a class of interesting applicative systems, we focus on the morphisms which can be “computed” by an applicative system. Before we can define what we mean by this, we recall from 3.4 the family of iterated application morphisms $\bullet^n : A \times A^n \rightarrow A$, derived from \bullet , including the special case for $n = 0$, where $\bullet^{(0)}$ is the composite $\bullet \Delta$.

We now define:

Definition 4.2 (Computable morphism, absolute version). *Let $\mathbb{A} = (A, \bullet)$ be an applicative system.*

- (i) *For $n \geq 0$, a morphism $f : A^n \rightarrow A$ is said to be \mathbb{A} -computable (or computable whenever \mathbb{A} is clear from the context) when there exists a total point $p : 1 \rightarrow A$ such that*

$$\begin{array}{ccc} A \times A^n & \xrightarrow{\bullet} & A \\ p \times 1 \uparrow & \nearrow f & \\ A^n & & \end{array}$$

is commutative. (We suppress the isomorphism $A^n \cong 1 \times A^n$). Moreover, if $n > 1$, it is required that the morphism

$$1 \times A^{n-1} \xrightarrow{p \times 1} A \times A^{n-1} \xrightarrow{\bullet^{n-1}} A$$

is total.

In this situation, we call p a code for the morphism f .

- (ii) *More generally, a morphism $A^n \rightarrow A^m$ is computable when all components are; finally, we say that a morphism $A^n \rightarrow 1$ is computable if its domain is (as a map $A^n \rightarrow A^n$).*

The second clause in the definition which requires a to be total in its first $n - 1$ arguments can be justified in a few different ways; see the remarks at the end of the section.

Remark. One can, in the definition of a computable morphism, replace the condition that the relevant diagram is commutative on the nose, by the relaxed condition that it commutes up to inequality (i.e. that $f \leq \bullet(p \times 1)$). This is typically referred to as the *realizability* of a morphism f . Thus, computability is a stricter notion which requires the domain of f to be captured by the code p .

In order to single out interesting applicative structures we may ask for certain useful morphisms to be computable. For example, we may want the identity morphism on A , or more generally the projection morphisms to be computable; one could additionally ask that the computable maps are closed under composition. Ultimately, one wants to extract a category from \mathbb{C} , where the objects are the finite powers of A and the morphisms are the computable maps. This category, which will be denoted by $\mathbf{Comp}(\mathbb{A})$, should be constructed as follows:

Objects: Formal finite powers of A . We may identify these with the natural numbers $\underline{0}, \underline{1}, \dots$

Morphisms: A morphism $\underline{n} \rightarrow \underline{m}$ is an \mathbb{A} -computable morphism $A^n \rightarrow A^m$ in the base category \mathbb{C} .

One could directly impose the requirement on \mathbb{A} that this forms a category; however, we will take the more common approach of discussing combinatory completeness first and then showing that these two things are equivalent.

From $\mathbb{A} = (A, \bullet)$, we can build a subcategory of \mathbb{C} , called the \mathbb{A} -*polynomial category*, or simply the *polynomial category*; this is defined to be the smallest cartesian restriction subcategory of \mathbb{C} on the objects $1, A, A^2, \dots$ which contains all total points of A as well as the application map. Morphisms in this subcategory are called \mathbb{A} -*polynomial morphisms*; think of such a morphism $f : A^n \rightarrow A$ as a polynomial in n variables with coefficients in A .) The key definition is now:

Definition 4.3 (Combinatory Completeness). *An applicative system is combinatory complete when every polynomial morphism is computable.*

It is a classic result that combinatory completeness already follows from two instances (see, for example, [4]). Define the map k_0 to be the first projection $\pi_0 : A \times A \rightarrow A$, and define the map s_0 to be the composite

$$A \times A \times A \xrightarrow{\langle \pi_{13}, \pi_{23} \rangle} (A \times A) \times (A \times A) \xrightarrow{\bullet \times \bullet} A \times A \xrightarrow{\bullet} A.$$

Theorem 4.4. *For an applicative system $\mathbb{A} = (A, \bullet)$, the following are equivalent:*

- (i) \mathbb{A} is combinatory complete
- (ii) the polynomial morphisms k_0, s_0 are \mathbb{A} -computable.

The codes for k_0, s_0 are usually denoted by \mathbf{k} and \mathbf{s} , respectively. It is important to note that such combinators (as they are usually called) are by no means unique.

We now define a *Partial Combinatory Algebra* (PCA for short) to be a combinatory complete applicative system. Often, a PCA is taken to be an applicative system together with chosen combinators \mathbf{k} and \mathbf{s} ; in this work, we shall avoid making these choices as the constructions and morphisms we shall be considering will not rely on having such choice. However, for some further discussion, see Section 4.4.

Before we turn to examples of PCAs, we give the promised result (proved in the total case by Longo and Moggi) that combinatory completeness is precisely what is needed to guarantee the computable maps do form a category.

Theorem 4.5 (Categorical characterization of completeness). *For an applicative system $\mathbb{A} = (A, \bullet)$ in a category \mathcal{C} , the following are equivalent:*

- (i) \mathbb{A} is combinatory complete
- (ii) The \mathbb{A} -computable morphisms form a cartesian restriction category over \mathcal{C} .

Proof. We sketch the proof, which is a straightforward extension of the total case proved in [31].

Assume first that A is combinatory complete. By definition, this means that every polynomial map is computable. In particular, the application map is computable, as is every total element. Now if $f : A^n \rightarrow A$ is computable, we may express f as a polynomial, since it has a code a such that $f = \bullet^{(n)}(a \times 1)$, expresses f as a composite of polynomials. Thus the polynomial maps and the computable maps coincide, and hence the computable maps form a cartesian restriction category.

For the converse, assume that computable maps form a category; we first note that this implies that the identity map $A \rightarrow A$ has a code, say i . Then i is also a code for the application map, and hence the application map is computable. Now the maps k_0, s_0 defined above are built from the application and the cartesian structure, so are in the category of computable maps. By the previous theorem this implies combinatory completeness. \square

In this proof, one aspect has been hidden: given a morphism f , how do we see that \bar{f} is again computable? Consider the diagram

$$\begin{array}{ccc}
 & A \times A \times A & \xrightarrow{\bullet} A \\
 & \uparrow \mathbf{k} \times A \times A & \nearrow \pi_0 \\
 A & \xrightarrow{\langle 1, f \rangle} A \times A &
 \end{array}$$

By definition of \mathbf{k} , the triangle commutes; the composite $\pi_0 \langle 1, f \rangle$ equals the domain \bar{f} . Since all maps are computable, so is \bar{f} .

Of course, what is happening here is that in the presence of partial products, the restriction structure is definable in terms of the products, namely as $\bar{f} = \pi_0 1, f$

We pause for a moment to contemplate what the elements of a PCA are. We defined an element $p : 1 \rightarrow A$ to be computable if it has a $\bullet^{(0)}$ -index; an equivalent formulation would be: an element $p : 1 \rightarrow A$ is computable if the composite $A \xrightarrow{1} 1 \xrightarrow{p} A$ is computable. The first formulation has the advantage that it explains the relation between the collection of total and of partial elements of a PCA. We state this in a lemma:

Lemma 4.6. *Let $\mathbb{A} = (A, \bullet)$ be a PCA in \mathcal{C} . Then every partial element $p : 1 \rightarrow A$ factors through $\bullet^{(0)}$ via a total element.*

Proof. Of course, the desired total element is simply a $\bullet^{(0)}$ -index for p . \square

In set-theoretic notation, this means that every partial element p may be written $p = u \bullet u$, for some total u .

The main point is, that this reveals more of the role of the family $\bullet^{(n)}$ and why $\bullet^{(0)}$ fits into it: each member is universal in $\mathbf{Comp}(\mathbb{A})$, and hence every morphism $A^n \rightarrow A$ can be obtained by substituting a total element in the leftmost argument of $\bullet^{(n)}$. Thus the family provides (weak) normal forms for maps in the category.

So far, we have shown that a PCA $\mathbb{A} = (A, \bullet)$ in \mathbf{C} gives a faithful category over \mathbf{C} on the computable maps. Given the preceding discussion of combinatory completeness, it should not come as a surprise that $\mathbf{Comp}(\mathbb{A})$ is in fact a Turing category with Turing morphism \bullet , regarded as a morphism $\underline{2} \rightarrow \underline{1}$. The only thing left to verify is that every object \underline{n} is a retract of $\underline{1}$. We show this for $\underline{2}$, and the others will follow. The point is that combinatory completeness entails that one can construct a computable pairing map $p : A \times A \rightarrow A$, together with computable unpairings $p_0, p_1 : A \rightarrow A$ (see, e.g. [27]). Then $A \times A$ is a retract of A in \mathbf{C} via the pair of computable maps p and $\langle p_0, p_1 \rangle$. Thus we have shown:

Theorem 4.7. *For any PCA $\mathbb{A} = (A, \bullet)$ in \mathbf{C} , the category $\mathbf{Comp}(\mathbb{A}, \mathbf{C})$ is a Turing category.*

Although it is probably clear from the construction, it is important to remark that the Turing category $\mathbf{Comp}(\mathbb{A}, \mathbf{C})$ is faithful over \mathbf{C} , via the functor $\underline{n} \mapsto A^n$. As stated earlier, the world of “Turing categories over a fixed base category” is particularly important for us; thus, we have found a way of generating such Turing categories, namely by considering PCAs in the base category.

Remark. One can of course consider the category $\mathbf{Real}(\mathbb{A})$ of *realizable maps* (those maps f for which there exists a realizer p such that $h \leq \bullet(p \times 1)$). Everything above generalizes immediately, and we find that every PCA gives rise to a Kleene category (as discussed in Example 3.2.6). Moreover, $\mathbf{Real}(\mathbb{A})$ is obtained from $\mathbf{Comp}(\mathbb{A})$ by formally adding all domains from \mathbf{C} .

We end this section with an elementary observation which allows us to transport PCAs along cartesian functors. If $\mathbb{A} = (A, \bullet)$ is an applicative system in \mathbf{C} and $F : \mathbf{C} \rightarrow \mathbf{D}$ is a cartesian restriction functor, then $F\mathbb{A} = (FA, F\bullet)$ is an applicative system in \mathbf{D} . Moreover, if we can make a choice for \mathbf{k}, \mathbf{s} in \mathbb{A} , then $F\mathbf{k}, F\mathbf{s}$ will turn $F\mathbb{A}$ into a PCA. Thus we may say that cartesian restriction functors preserve PCAs as long as in the external world we are able to make a choice of combinators.

4.2. Examples of PCAs

Given the connection between Turing categories and PCAs, it is now clear that each example of a Turing object in a Turing category is an example of a PCA (in that Turing category). Nevertheless, we give some more to complete the picture.

Kleene’s first model. The prime example of a PCA is Kleene’s model \mathbb{K}_1 ; its underlying set is \mathbb{N} , and application is Kleene-application as defined in the previous section. Combinators \mathbf{k} and \mathbf{s} are found using the Parameter Theorem (see [6]) for a precise account). Of course, $\mathbf{Comp}(\mathbb{K}_1)$ is the classical recursion category (first example in the previous section).

Oracles. One may generalize this example by considering computability relative to

an oracle A ; the resulting model is denoted \mathbb{K}_1^A . It has the same underlying set \mathbb{N} , but application is given by $n \bullet^A m = \phi_{n,A}(m)$, where $\phi_{0,A}, \phi_{1,A}, \dots$ is a suitable enumeration of the class of unary partial functions which are computable with help of the oracle A .

Basic recursive function theory. Generalizing the situation even further, there is the notion of a so-called *Basic Recursive Function Theory*, BRFT for short (see [49,51]). Essentially, a BRFT consists of an infinite set B , a collection of partial functions \mathcal{F} on B , and, for each n , an enumeration function $\Phi^{(n)}$, which plays the role of the universal function of n variables. It is required that \mathcal{F} contains the identity, constant functions and the parametrization functions, as well as decision functions $d(x, y, u, v)$, which are supposed to satisfy

$$d(x, y, u, v) = \begin{cases} u & \text{if } x = y \\ v & \text{otherwise.} \end{cases}$$

Now we can use the enumeration function $\Phi^{(1)}$ to define an application on B , and use the parametrization functions to define the combinators \mathbf{k}, \mathbf{s} . Thus, each BRFT can be made into a PCA. It can be shown that if the underlying set of a BRFT is the natural numbers, then the class \mathcal{F} must contain all partial recursive functions. Thus, this example does generalize some important aspects of elementary recursion theory over \mathbb{N} ; indeed it has been shown that most basic aspects of recursion theory can be proved in this setting.

Effective applicative structures. When, in the definition of a BRFT, one omits the requirements that the decision functions be present, one arrives at what Asperti and Longo ([2]) call an *effective applicative structure*. It is easily seen that this notion coincides with that of a combinatory complete applicative structure which is non-trivial (because we insist on the underlying set being infinite). Note that the requirement that the identity function and the constant functions be present is already ensured by combinatory completeness. Thus, after a series of generalizations from the recursive structure on the natural numbers, we have come back to PCAs (in the category of sets, that is).

Syntactical models. Combinatory logic (CL) is an equational theory whose models are total combinatory algebras (i.e. applicative systems whose application map is total) which come equipped with a specific choice of combinators \mathbf{k}, \mathbf{s} . Both the open and closed term models of CL are therefore examples of (total) PCAs. It is well-known that CL admits a rewriting presentation; therefore it makes sense to look at strongly normalizing terms. The set of strongly normalizing terms modulo provable equality is a PCA; for details, see [6].

Partial combinatory logic (PCL) is a partial equational theory whose models are PCAs (in which one has chosen combinators). As indicated in Example 3.2.5, it gives rise to a classifying category \mathcal{C}_{PCL} containing a generic model; if we are given a PCA in an arbitrary cartesian restriction category \mathcal{C} then there is an essentially unique cartesian restriction functor which sends the generic PCA to the given model. In particular, we can apply the global sections functor to the generic PCA, and the result is (up to isomorphism) the PCA of strongly normalizing CL-terms.

Models of lambda calculus. Any model of the (partial) lambda calculus is a PCA; in particular, a reflexive object in a (partial) cartesian closed category is a PCA.

4.3. Idempotent splittings and relative computability

We have just seen that every PCA in \mathbf{C} gives rise to a Turing category over \mathbf{C} . Obviously, we would like to know whether every Turing category over \mathbf{C} arises in this manner. There are two aspects to the answer to this question; one has to do with splitting of idempotents, and the second with relative computability. We will discuss idempotent splittings first.

We have taken the objects of $\mathbf{Comp}(A)$ to be the formal finite powers of A . Clearly, there are more Turing categories which are “generated” by A , since we could formally split a class of idempotents in $\mathbf{Comp}(A)$ as to obtain a bigger Turing category with essentially the same Turing structure. The following lemma tells us that any Turing category is, up to idempotent splitting, equivalent to the category of computable maps (for any choice of Turing object).

Lemma 4.8. *Let \mathbf{D} be a Turing category with Turing object $\mathbb{A} = (A, \bullet)$. Then the categories $\mathbf{Comp}(\mathbb{A}, \mathbf{D})$ and \mathbf{D} are Morita-equivalent.*

Proof. In \mathbf{D} , we may, for each n , choose an embedding-retraction pair $(m_n, r_n) : A^n \triangleleft A$; the composite $m_n r_n$ is an idempotent on A , and when we split these idempotents we obtain a Turing category \mathbf{D}' , which has the property that $\mathbf{Comp}(\mathbb{A})$ embeds faithfully into it. Now because all the objects of \mathbf{D}' are retracts of A , they correspond to computable idempotents on A , i.e. are idempotents in $\mathbf{Comp}(\mathbb{A})$. Thus, if we split these idempotents, the resulting category will be equivalent to \mathbf{D}' . \square

This highlights the role of categories of the form $\mathbf{Comp}(\mathbb{A})$: they serve as a minimal environment in which PCAs and their computable maps live; other Turing categories are supposed to be viewed as (non-essential) inflations of such minimal categories.

We may summarize the discussion so far in the following theorem:

Theorem 4.9 (Structure theorem for Turing categories).

- (i) *When $\mathbb{A} = (A, \bullet)$ is a PCA in \mathbf{D} , then $\mathbf{Comp}(\mathbb{A})$ is a Turing category with Turing object (A, \bullet) .*
- (ii) *If \mathbf{D} is a Turing category and (A, \bullet) is a Turing object, then $\mathbb{A} = (A, \bullet)$ is a PCA in \mathbf{D} and $\mathbf{D} \simeq \mathcal{K}_E(\mathbf{Comp}(\mathbb{A}, \mathbf{D}))$ for some class E of idempotents.*

In particular, given two different choices \mathbb{A}, \mathbb{B} of Turing objects, we may get different categories $\mathbf{Comp}(\mathbb{A}, \mathbf{D}), \mathbf{Comp}(\mathbb{B}, \mathbf{D})$, but these are Morita-equivalent.

We now return to classifying Turing categories over a fixed base category \mathbf{C} . Suppose we are given a Turing category \mathbf{D} together with a faithful functor $F : \mathbf{D} \rightarrow \mathbf{C}$. Is there a PCA which generates \mathbf{D} ? The obvious candidate is the (any) Turing object in \mathbf{D} . Let us choose one such PCA $\mathbb{A} = (A, \bullet)$. Since any cartesian functor sends PCAs to PCAs, we find that $F(\mathbb{A}) = (FA, F\bullet)$ is a PCA in \mathbf{C} . We would hope that $F(\mathbb{A})$ generates \mathbf{D} , but there is one problem: the PCA $F(\mathbb{A})$ may have total elements $1 \rightarrow FA$ which are not in the image of F . That would imply that $F(\mathbb{A})$ -computable maps are not just the images of \mathbb{A} -computable maps under F , and hence that the category $\mathbf{Comp}(F(\mathbb{A}), \mathbf{C})$ would be too large.

The remedy is to refine the construction of $\mathbf{Comp}(F(\mathbb{A}), \mathbf{C})$ by taking into account which total elements are considered to be computable. This is done in the following definition (where Γ is the global sections functor).

Definition 4.10 (Computable morphism, relative case). *Let $\mathbb{A} = (A, \bullet)$ be an applicative system in \mathbf{C} and let \mathcal{V} be a subset of $\Gamma(A)$.*

- (i) *A morphism is called $(\mathbb{A}, \mathcal{V})$ -computable when it has a code $a \in \mathcal{V}$.*
- (ii) *The $(\mathbb{A}, \mathcal{V})$ -polynomial subcategory is the smallest cartesian subcategory on the powers of A containing all points in \mathcal{V} , containing the application.*
- (iii) *The pair $(\mathbb{A}, \mathcal{V})$ is combinatory complete when every $(\mathbb{A}, \mathcal{V})$ -polynomial map is $(\mathbb{A}, \mathcal{V})$ -computable.*

The results in the previous section generalize. In particular, $(\mathbb{A}, \mathcal{V})$ is combinatory complete if and only if the $(\mathbb{A}, \mathcal{V})$ -computable maps form a cartesian restriction category which is faithful over \mathbf{C} . We can also characterize combinatory completeness in terms of a closure property on $(\mathbb{A}, \mathcal{V})$; to this end, we exploit the fact that the global sections functor Γ sends PCAs to PCAs in \mathbf{Par} (being a cartesian restriction functor).

Theorem 4.11. *For an applicative system $\mathbb{A} = (A, \bullet)$ in \mathbf{C} and a collection of total points \mathcal{V} of A , the following are equivalent:*

- (i) *$(\mathbb{A}, \mathcal{V})$ is combinatory complete;*
- (ii) *the $(\mathbb{A}, \mathcal{V})$ -computable maps form a cartesian restriction category over \mathbf{C} ;*
- (iii) *\mathbb{A} is a PCA, and $\mathcal{V} \subseteq \Gamma(\mathbb{A})$ is a sub-PCA of $\Gamma(\mathbb{A})$.*

Here, by a sub-PCA we mean that the application on $\Gamma(\mathbb{A})$ restricts to an application on \mathcal{V} (i.e. that the latter is closed under the application) and that \mathcal{V} contains a basis for the combinatory completeness of $\Gamma(\mathbb{A})$. Thus in particular, \mathcal{V} is itself a PCA.

We shall call a combinatory complete pair $(\mathbb{A}, \mathcal{V})$ a *relative PCA*. The adjective “relative” is supposed to remind the reader of relative realizability, not of oracle (relative) computability. Indeed, if we were to replace computable maps by realizable maps, we would arrive exactly at relative realizability.

The category of (A, \mathcal{V}) -computable morphisms shall be denoted $\mathbf{Comp}(\mathbb{A}, \mathcal{V})$.

Examples. Here are a few examples of relative PCAs. First, given any PCA \mathbb{A} there are two evident choices one can make for \mathcal{V} , one can take all total points (in which case we are back in the absolute case) or one can take a “smallest” set, by considering the sub-PCA generated by (a choice of) \mathbf{k}, \mathbf{s} . (This is sometimes called the kernel.)

Next, in many situations a given model will have an “effective submodel”. This occurs often in the context of domains: take for example the Graph model $\mathcal{P}\omega$, and recall that every element $U \subseteq \omega$ codes a continuous function $\mathcal{P}\omega \rightarrow \mathcal{P}\omega$. Among the continuous functions, there are those which are *effective*, i.e. whose modulus of continuity is, after coding, an r.e. subset. It turns out (see [27]), that the r.e. subsets of $\mathcal{P}\omega$ form a sub-PCA of the Graph model.

Finally, one may consider an extension T of combinatory logic, or of the lambda calculus; this always gives rise to a pair of syntactic combinatory algebras, namely the closed term model of T and the open term model of T ; such a pair is a relative PCA.

The additional parameter \mathcal{V} is exactly what is needed to recover an arbitrary Turing category (up to Morita-equivalence) from a PCA. This is the content of the following theorem, which generalizes Theorem 4.9.

Theorem 4.12 (Structure theorem for Turing categories over a base). *Let \mathbf{C} be a cartesian restriction category.*

- (i) *If $(\mathbb{A}, \mathcal{V})$ is a relative PCA in \mathbf{C} , then $\mathbf{Comp}(\mathbb{A}, \mathcal{V})$ is a Turing category which is faithful over \mathbf{C} ;*

(ii) If \mathbf{D} is a Turing category with a faithful functor $F : \mathbf{D} \rightarrow \mathbf{C}$, then for any Turing object $\mathbb{A} = (A, \bullet)$ of \mathbf{D} , the pair $(F(\mathbb{A}), \mathcal{V})$, where $\mathcal{V} = \{Fa : 1 \rightarrow FA \mid a \in \text{Tot}(\mathbf{D}(1, A))\}$, is a relative PCA in \mathbf{C} . Moreover, there is a class E of idempotents such that the categories \mathbf{D} and $\mathcal{K}_E(\text{Comp}(F(\mathbb{A}), \mathcal{V}))$ are equivalent.

Proof. Analogous to the absolute case. □

4.4. Remarks

In the literature on combinatory logic one defines a PCA to be an applicative system together with chosen combinators \mathbf{k}, \mathbf{s} . This is exactly a model of partial combinatory logic (PCL) as defined in Example 3.2.6. Indeed, in this context one studies models of combinatory logic as algebraic variety, and the kind of constructions and properties of PCAs one is interested in depend essentially on the choice of combinators.

However, we use PCAs for different purposes; our constructions and morphisms do not depend in any way of those choices, and it is only the property of combinatory completeness that matters. For us it is still useful to know that PCAs admit a *presentation* as models of PCL, but according to our definition (which is the one commonly found in the literature on realizability), PCAs are actually not algebraic, as they must satisfy a non-algebraic property!

There is one limitation to our definition: we have, in demanding that codes for computable morphisms be total elements, implicitly required that our PCAs have total support. Thus, in order to handle PCAs which do not have total support, we would have to slice by the support (an idempotent on the terminal object) and then work in that slice. Alternatively, it is possible to reformulate the theory while working, instead of with total elements, with *grounded* elements, which are of equal support as the PCA. We have nevertheless opted to present the theory in this simpler, less general, form, mainly because of simplicity of the exposition and because the general version can easily be deduced when the necessity would ever arise.

Relative realizability originated, on a syntactical level, in the work of Kreisel; the idea of treating this using pairs of PCAs seems to have originated in the Carnegie Mellon School [3] and was further developed by Birkedal in [7] and in Birkedal and van Oosten [8]. In [19], it was recognized that it was the extra generality of having a relative PCA (there called a filter of designated truth-values) which allows for describing realizability toposes purely in terms of this combinatorial data. The example of a model together with its effective submodel is especially important for computation at higher types; see [28] for examples and discussion. Finally, it is worth noting that a similar concept was already employed in Moschovakis' work on abstract computability [33].

The padding construction from Section 3.4 has another interesting implication. In [6], it is shown that under certain conditions, a PCA (in **Par**) can be embedded into a total PCA. The relevant condition (called *uniqueness of head normal forms*) is that the elements $\mathbf{k}, \mathbf{k}x, \mathbf{s}y, \mathbf{s}uv$ are all distinct, and that $\mathbf{s}xy = \mathbf{s}uv$ implies $x = u, y = v$. Now if we start with a PCA (A, \bullet) (again, in **Par**), then we can apply the padding construction and obtain a new PCA (A, \bullet') (and these two PCAs are isomorphic in the sense to be discussed in the next Section). Now in the PCA (A, \bullet') one can always choose combinators \mathbf{k} and \mathbf{s}

such that one has unique head normal forms. (This can be done thanks to the freedom one has in choosing the second “free” coordinate of codes.)

The requirement in the definition of computable map that the code is total in the first $n - 1$ arguments of course neatly agrees with the requirement that indices in Turing categories are total, but one may ask for some independent reason why such a condition should be imposed. In the case of a cartesian closed partial map category, exponential transposes are always total maps. Thus, one can use the process of Currying $n - 1$ times and obtain a total map.

In addition, John Longley has shown that if one has a conditional PCA (which need not satisfy the requirement that sxy is total, but does satisfy sx total, one can modify the application and obtain an equivalent PCA (in the sense to be discussed below).

Apart from the aspects mentioned earlier, there is another regard in which our presentation here differs from that in Longo and Moggi [31]: they prove that the category $\mathbf{Comp}(\mathbb{A})$ contains an object whose global sections form a combinatory algebra isomorphic to \mathbb{A} , while we observe directly that this object is in fact an internal PCA in $\mathbf{Comp}(\mathbb{A})$. Since global sections is a cartesian functor, it preserves (well-supported) PCAs, so our observation implies the result in [31].

The possibility of regarding PCAs in categories different from the category of Sets has, in our opinion, not been sufficiently exploited. In the present context, we have seen already that this extra generality gives rise to cleaner formulations; also, there are various natural questions, such as what are the PCAs in a given Turing category, and what are the relations between them. Moreover, the generic model is another important example of a PCA in a non-concrete category. Finally, especially in the literature on models of the lambda calculus many authors, when presented with a reflexive object, seem to insist on immediately applying global sections, as if the reflexive object isn’t the real model at hand, but merely a means of getting hold of a model of the lambda calculus in Sets. But global sections is not cartesian closed (and hence it is, from a general categorical point of view, rather odd to apply with great expectations to a reflexive object) and, moreover, it need not be faithful. So unless we’re desperate to study the image of the reflexive object *qua* PCA in Sets, why would we be merely looking at this Platonic shadow?

5. Equivalences and further results

In the last Section we obtained a correspondence between (relative) PCAs in a fixed base category on the one hand, and Turing categories with faithful functors into the base (up to Morita-equivalence) on the other hand. Naturally, one would like to know if and how the passage from PCAs to Turing categories is functorial, and whether the above correspondence could be expressed as an equivalence of categories.

These questions can all be answered in a quite satisfactory way, which involves defining suitable notions of morphisms for PCAs and transformations between those (such that we obtain a 2-category of PCAs) and defining suitable notions of morphisms of Turing categories and their transformations (giving a 2-category of Turing categories over the base); then one can show that the assignment $(\mathbb{A}, \mathcal{V}) \mapsto \mathbf{Comp}(\mathbb{A}, \mathcal{V})$ is a 2-functor which induces the desired 2-equivalence.

In this paper we will not prove these results, which will appear as [11], since they involve

more machinery than is appropriate here. Rather, we will discuss morphisms of relative PCAs and use those to obtain various results, such as the characterization of PCAs in a Turing category. We will also prove that Turing categories are a suitable invariant for PCAs, in the sense that two PCAs are equivalent if and only if their corresponding Turing categories are Morita-equivalent over the base category.

We start (Subsection 5.1) by explaining what simulations of PCAs are, and by organizing (relative) PCAs in a 2-category. This is mainly a review, although the generalization to the relative case has, so far, only been addressed in [19].

5.1. Simulations

In this section we review the basic theory of simulations of PCAs and relative PCAs. We will also define transformations of simulations, and the upshot will be that we obtain a 2-category of PCAs, simulations and transformations. For clarity, we first deal with the absolute case.

Definition 5.1 (Simulation of PCAs, absolute version). *Let $\mathbb{A} = (A, \bullet_A), \mathbb{B} = (B, \bullet_B)$ be PCAs and let $\phi : A \rightarrow B$ be a morphism on the underlying objects.*

- (i) *Given an \mathbb{A} -computable morphism $f : A^n \rightarrow A$, we say that f is ϕ -computable if there exists a \mathbb{B} -computable morphism $h : B^n \rightarrow B$ such that the following diagram commutes:*

$$\begin{array}{ccc} A^n & \xrightarrow{f} & A \\ \phi^n \downarrow & & \downarrow \phi \\ B^n & \xrightarrow{h} & B \end{array}$$

In the above diagram, we say that h (strictly) simulates f .

- (ii) *The map $\phi : A \rightarrow B$ is a (strict) simulation when every \mathbb{A} -computable f is ϕ -computable.*

A few remarks are in order. First, as said above, this definition is really one which applies to applicative systems, since it does not mention combinatory completeness at all. In some situations it is indeed desirable to consider simulations from an arbitrary applicative system to a PCA, but in this paper we will not need that.

Second, we note that the assignment $f \mapsto h$ is not functorial in any way. In certain cases it may be possible to make a choice of simulating morphisms which turns this into a functor of some kind, but in general there is a variety of reasons which prevent this.

Third, if one replaces the commutativity of the above diagram by an inequality $f \circ \phi \leq \phi^n \circ h$, one says that f is ϕ -realizable; if every computable map can be ϕ -realized one speaks of a *lax simulation*. Note that a lax simulation will not be able to “detect” domains; as a consequence, one can have a lax simulation from a partial to a total structure, while this is generally not possible for a strict simulation.

The definition given above can be normalized by exploiting the specific structure of PCAs.

Lemma 5.2. *For PCAs $\mathbb{A} = (A, \bullet_A), \mathbb{B} = (B, \bullet_B)$ and a morphism $\phi : A \rightarrow B$ the following are equivalent:*

- (i) *ϕ is a simulation;*
(ii) *there exists a code $u : 1 \rightarrow B$ with the property that*

$$\begin{array}{ccc}
A^2 & \xrightarrow{\bullet} & A \\
u \times \phi \times \phi \downarrow & & \downarrow \phi \\
B^3 & \xrightarrow{\bullet} & B
\end{array}$$

commutes;

(iii) there exists a \mathbb{B} -computable $w : B \rightarrow B$ such that

$$\begin{array}{ccc}
A^2 & \xrightarrow{\bullet} & A \\
w\phi \times \phi \downarrow & & \downarrow \phi \\
B^2 & \xrightarrow{\bullet} & B
\end{array}$$

commutes.

The proof is left as an easy exercise.

It is the second condition which is normally found in texts on realizability; in the internal language, it reads $u \bullet \phi(x) \bullet \phi(y) = \phi(x \bullet y)$. In this form (actually, a relational version of this form), the concept is due to Longley [27], who coined them “applicative morphisms”. Adventurous as that may sound, we have shifted terminology to “simulations” in order to stress our intuition from the main examples as well as the fact that the concept applies to more types of structure than just applicative systems.

We wish to emphasize that, while for PCAs the above Lemma tells us that the second formulation is adequate, this formulation is not available in other contexts, as it relies on the application operation (as well as the combinatory completeness). One can readily extend the first formulation to define simulations between any kinds of partial algebras, since it only relies on a notion of polynomial. This was also used in [19] where it was used for structures weaker than PCAs, and it will also be investigated in [11].

Examples. We briefly sketch a few examples here in order to give the reader some intuition for the concept.

- (i) First, any constant morphism (factoring through a total element) is a simulation. Such a simulation will be considered degenerate. Note that a constant morphism will never preserve the application (unless the codomain is trivial). Note also that this is always a lax simulation if the domain is not a total PCA.
- (ii) Next, one may take a PCA \mathbb{A} , and use the combinatory completeness to define the Church numerals in \mathbb{A} . In case \mathbb{A} is a PCA in \mathbf{Par} , this gives a lax simulation of \mathbb{K}_1 into \mathbb{A} . This simulation is in general not strict (for example, when \mathbb{A} is total).
- (iii) If $A^A \triangleleft A$ is a reflexive object, then the PCA structure on A depends on the choice of embedding-retraction pair. Two different choices lead to different application maps; however, there will always be simulations between the two PCAs.
- (iv) Any homomorphism of applicative structures (by which we mean an application preserving map) is a simulation. This includes the homomorphisms of PCAs used in the model-theoretic context. Thus, any inclusion of PCAs (preserving a chosen basis) is a simulation, as is any quotient map (in the sense of [5]).
- (v) If \mathbf{SN} denotes the PCA of strongly normalizing terms of combinatory logic, then there is a strict simulation $\mathbf{SN} \rightarrow \mathbb{K}_1$; this is essentially the same as saying that there is an effective Gödel-numbering of combinatory logic terms, and that syntactic application and normalization can be performed effectively on the level of codes.

The following Lemma gives an (atypical) example of how a morphism of PCAs can arise. It will be of use in Section 5.2.

Lemma 5.3. *Let $F : \mathbb{C} \rightarrow \mathbb{D}$ be a cartesian restriction functor and $\mathbb{A} = (A, \bullet)$ a PCA in \mathbb{C} . Then $F\mathbb{A} = (FA, F\bullet)$ is a PCA in \mathbb{D} . Moreover, if $\alpha : F \rightarrow G$ is a natural transformation between such functors, then there is an induced morphism of PCAs $\alpha : F\mathbb{A} \rightarrow G\mathbb{A}$.*

Proof. The first part was already mentioned in Section 4.1: we only need to choose a basis for the combinatory completeness of \mathbb{A} ; then F will send this to a basis for the completeness of $F\mathbb{A}$.

For the second part, we consider the naturality of α at the application, which immediately gives

$$\begin{array}{ccc} FA \times FA & \xrightarrow{\alpha \times \alpha} & GA \times GA \\ F\bullet \downarrow & & \downarrow G\bullet \\ FA & \xrightarrow{\alpha} & GA \end{array}$$

That is, α is a strict simulation from $F\mathbb{A}$ to $G\mathbb{A}$. □

Of course, this is simply an instance of the general framework of the functorial semantics for (partial) equational theories.

Simulations for relative PCAs. In the previous section we encountered the need for replacing PCAs by PCAs together with a chosen collection of total elements; a pair $(\mathbb{A}, \mathcal{V})$ where $\mathcal{V} \subseteq \Gamma(\mathbb{A})$ is a sub-PCA of $\Gamma(\mathbb{A})$ was called a relative PCA.

We will now extend the notion of simulation to relative PCAs, and then organize these in a 2-category.

Definition 5.4 (Simulation of PCAs, relative version). *Let $(\mathbb{A}, \mathcal{V}), (\mathbb{B}, \mathcal{W})$ be relative PCAs and let $\phi : A \rightarrow B$ be a morphism on the underlying objects.*

- (i) *Given an $(\mathbb{A}, \mathcal{V})$ -computable morphism $f : A^n \rightarrow A$, we say that f is ϕ -computable if there exists a $(\mathbb{B}, \mathcal{W})$ -computable morphism $h : B^n \rightarrow B$ such that the following diagram commutes:*

$$\begin{array}{ccc} A^n & \xrightarrow{f} & A \\ \phi^n \downarrow & & \downarrow \phi \\ B^n & \xrightarrow{h} & B \end{array}$$

- (ii) *The map $\phi : A \rightarrow B$ is a (strict) simulation when every $(\mathbb{A}, \mathcal{V})$ -computable f is ϕ -computable and moreover, $\phi(v) \in \mathcal{W}$ for each $v \in \mathcal{V}$.*

Thus, the only new ingredients are that the simulating morphism h should not just be \mathbb{B} -computable, but $(\mathbb{B}, \mathcal{W})$ -computable (i.e. it must have a code in \mathcal{W}), and that ϕ preserves the designated global elements.

Again, one can normalize, and Lemma 5.2 generalizes immediately:

Lemma 5.5. *For relative PCAs $(\mathbb{A}, \mathcal{V}), (\mathbb{B}, \mathcal{W})$ and a morphism $\phi : A \rightarrow B$ the following are equivalent:*

- (i) *ϕ is a simulation;*
(ii) *there exists a code $u : 1 \rightarrow B$ in \mathcal{W} with the property that*

$$\begin{array}{ccc}
A^2 & \xrightarrow{\bullet} & A \\
u \times \phi \times \phi \downarrow & & \downarrow \phi \\
B^3 & \xrightarrow{\bullet} & B
\end{array}$$

commutes;

(iii) there exists a $(\mathbb{B}, \mathcal{W})$ -computable $w : B \rightarrow B$ such that

$$\begin{array}{ccc}
A^2 & \xrightarrow{\bullet} & A \\
w\phi \times \phi \downarrow & & \downarrow \phi \\
B^2 & \xrightarrow{\bullet} & B
\end{array}$$

commutes.

It is easily verified that relative PCAs in a fixed category \mathcal{C} and simulations form a category, denoted $\mathfrak{PCA} = \mathfrak{PCA}(\mathcal{C})$. (For the absolute case, proofs can be found in, e.g., [27,20]; the relative case is hardly different and can be found in [19].)

The category \mathfrak{PCA} is preorder-enriched. For two parallel simulations $\phi, \psi : (\mathbb{A}, \mathcal{V}) \rightarrow (\mathbb{B}, \mathcal{W})$, we define $\phi \vdash \psi$ if and only if there exists a code $w \in \mathcal{W}$ such that the following diagram commutes:

$$\begin{array}{ccc}
A & & \\
\langle w, \phi \rangle \downarrow & \searrow \psi & \\
B \times B & \xrightarrow{\bullet} & B
\end{array}$$

In this case, we say that w transforms the simulation ϕ into the simulation ψ . Note that it doesn't matter which w does this, only that there exists one. In set-theoretic notation, the element w must satisfy $w \bullet \phi(x) = \psi(x)$ (and in particular, w must always be defined on elements of the form $\phi(x)$).

This definition turns \mathfrak{PCA} into a 2-category; thus, it makes sense to speak of adjunctions, or equivalences between (relative) PCAs. Equivalences are of particular importance to us; the following Lemma gives a criterion for two maps to constitute an equivalence.

Lemma 5.6. *Whenever $\phi : \mathbb{A} \rightarrow \mathbb{B}, \psi : \mathbb{B} \rightarrow \mathbb{A}$ are two morphisms of PCAs, then the following are equivalent:*

- (i) ϕ, ψ constitute an equivalence of PCAs
- (ii) the composite $\psi\phi$ is \mathbb{A} -computable and has an \mathbb{A} -computable retraction, and the composite $\phi\psi$ is \mathbb{B} -computable and has a \mathbb{B} -computable retraction.

Proof. The computability of the composite $\psi\phi$ corresponds to the existence of a transformation $1_A \vdash \psi\phi$; the existence of a computable retraction to a transformation $\psi\phi \vdash 1_A$. \square

Examples. We shall conclude this section by giving a few examples of 2-categorical aspects of PCAs.

- (i) First, recall that any constant map is in fact a simulation; any two such constant maps will be isomorphic.
- (ii) More interestingly, we mentioned that any PCA \mathbb{A} in \mathbf{Par} can simulate \mathbb{K}_1 , Kleene's first model (via a lax simulation, but not necessarily via a strict one). A particular choice of numerals in \mathbb{A} amounts to a particular (injective!) simulation $\phi : \mathbb{K}_1$

- $\rightarrow \mathbb{A}$. Given two such choices of numerals ϕ, ψ , one can show that these are always isomorphic.
- (iii) One may ask about self-equivalences of a given PCA; intuitively, these correspond to ways in which the PCA can encode itself. For example, one can show that a function $\phi : \mathbb{N} \rightarrow \mathbb{N}$ is the underlying function of a self-equivalence $\mathbb{K}_1 \rightarrow \mathbb{K}_1$ if, and only if, ϕ is a recursive injective function. One also observes that if a simulation $\mathbb{K}_1 \rightarrow \mathbb{K}_1$ is not injective, then it must be constant. Thus there are exactly two isomorphism classes of simulations $\mathbb{K}_1 \rightarrow \mathbb{K}_1$, namely the class of the identity and the class of the constant maps.
 - (iv) As we noted in Section 3.3, any two Turing morphisms in a Turing category (regarded as PCAs) are equivalent. Thus the construction in that Section of a new Turing morphism in fact shows that the property of having monic indices is not stable under equivalence; the same goes for extensionality.
 - (v) An example of an equivalence of PCAs is obtained by considering Graph models; it is well-known that these are completely determined once one chooses a total injective function $p : \mathbf{P}_f(A) \times A \rightarrow A$. When p, q are two such coding maps, the corresponding Graph models $G(p)$ and $G(q)$ are related via a simulation (which is in fact an isomorphism). Thus, while $G(p)$ and $G(q)$ are different *qua lambda models* they are isomorphic *qua PCAs* under simulation. See [27] for details.
 - (vi) Equivalences via lax simulations are weaker than via strict simulations; for example, in [50] an example of an equivalence between a total and a non-total PCA was given. This is impossible via strict simulations.

5.2. Equivalences

We now have a notion of equivalence of (relative) PCAs; intuitively, two PCAs are equivalent if they can be faithfully encoded into each other. One would hope, that equivalent PCAs generate the same notion of computability. This will be proved now, provided one takes “the same” to mean: Morita-equivalent. We will proceed by working in the absolute case instead of with relative PCAs, in order to reduce clutter; the proofs in the relative case are virtually the same.

Now in Section 3.3, we already established the following: any Turing object is a PCA (once we choose a Turing morphism); whenever we have two Turing objects $\mathbb{A} = (A, \bullet)$ and $\mathbb{B} = (B, \bullet)$, then there are simulations $\phi : \mathbb{A} \rightarrow \mathbb{B}$ and $\psi : \mathbb{B} \rightarrow \mathbb{A}$; indeed, to obtain ϕ , one picks any embedding-retraction pair $(\phi, \phi') : A \triangleleft B$ and shows that it is a simulation, and similarly for $(\psi, \psi') : B \triangleleft A$. Each of the maps involved are both \mathbb{A} -computable and \mathbb{B} -computable, since both \mathbb{A}, \mathbb{B} are Turing objects by assumption.

This proves the following coherence result for Turing morphisms:

Proposition 5.7. *Any two Turing morphisms in a given Turing category are equivalent as PCAs.*

We will now embark on the proof of the converse result, namely that equivalent PCAs generate equivalent Turing categories.

First we should remark that the assignment $\mathbb{A} \mapsto \mathbf{Comp}(\mathbb{A})$ is by no means functorial, if one takes that to mean that every simulation $\phi : \mathbb{A} \rightarrow \mathbb{B}$ would induce a *functor* $\Phi : \mathbf{Comp}(\mathbb{A}) \rightarrow \mathbf{Comp}(\mathbb{B})$; the notion of simulation is too incoherent for that. The morphisms between the computability categories which *are* induced by simulations will be called

categorical simulations; their theory will be studied in [11]. It will follow from that work that the 2-category of PCAs in \mathbf{C} with simulations and their transformations is in fact 2-equivalent to the 2-category of Turing categories over \mathbf{C} with categorical simulations and their transformations. So, an equivalence of PCAs will induce an equivalence of Turing categories, via categorical simulations. As it happens, splitting of idempotents will allow us to “straighten out” this equivalence and make it functorial. In this section, we will sketch a direct proof, which is based on the idea that, while not every simulation of PCAs induces a functor, certain well-behaved simulations do. Equivalences may be taken to consist of such well-behaved simulations, and then the result will follow.

Let us, for the rest of this section, say that a simulation $\phi : \mathbb{A} \rightarrow \mathbb{B}$ is *faithful* if it has a retraction ϕ' such that $\phi\phi'$ is \mathbb{B} -computable. Faithful simulations compose, so we obtain a 2-category $\mathfrak{P}\mathfrak{C}\mathfrak{A}_f(\mathbf{C})$ of PCAs in \mathbf{C} , faithful simulations and transformations. We first note that this includes all the equivalences:

Lemma 5.8. *If $\phi : \mathbb{A} \rightarrow \mathbb{B}, \psi : \mathbb{B} \rightarrow \mathbb{A}$ constitute an equivalence, then both ϕ and ψ are faithful.*

Proof. We show the result for ϕ . First note that there is an \mathbb{A} -computable f such that $f(\psi\phi(x)) = x$ for all x . Then let the desired retraction be defined by $\phi'(b) = f\psi(b)$. Clearly $\phi'\phi(x) = f\psi\phi(x) = x$, so that ϕ' is indeed a retraction of ϕ . Finally, $\phi\phi'$ is \mathbb{B} -computable since $\phi\phi'(b) = \phi f\psi(b) = h\phi\psi(b)$ for some \mathbb{B} -computable h (which can be found since ϕ is a simulation). Since both h and $\phi\psi$ are \mathbb{B} -computable, so is $\phi\phi'$. \square

The following technical result is the key to the main result:

Lemma 5.9. *Let \mathbb{A}, \mathbb{B} be PCAs in \mathbf{C} , and suppose that there is a faithful simulation $\phi : \mathbb{A} \rightarrow \mathbb{B}$. Then there is an induced cartesian restriction functor*

$$\mathcal{K}(\mathbf{Comp}(\mathbb{A})) \xrightarrow{\Theta} \mathcal{K}(\mathbf{Comp}(\mathbb{B})).$$

Proof. We will define Θ by specifying what it does on the subcategory $\mathbf{Comp}(\mathbb{A})$, since it then uniquely extends to the idempotent splitting.

Pick a retraction ϕ' , and note that $e = \phi\phi'$ is a \mathbb{B} -computable idempotent on B , so that $(\underline{1}, e)$ is an object of $\mathcal{K}(\mathbf{Comp}(\mathbb{B}))$. We now define

$$\Theta(\underline{n}) = (\underline{1}, e)^n.$$

Given a morphism $f : \underline{n} \rightarrow \underline{1}$ in $\mathbf{Comp}(\mathbb{A})$, we form $\Theta(f) = \phi \circ f \circ (\phi')^n$. This is a morphism in $\mathcal{K}(\mathbf{Comp}(\mathbb{B}))$ since ϕ is a simulation and since $\phi\phi'$ is computable. Clearly, Θ is functorial, because $\phi'\phi = 1$. Moreover, Θ preserves products by construction, so all that remains to be done is to check that it preserves restrictions.

Let us take a morphism $f : A \rightarrow A$, and compare $\overline{\Theta f}$ and $\Theta(\overline{f})$. The former is defined as $\overline{\phi f \phi' \phi \phi'}$, while the latter is $\overline{\phi f \phi'}$. Compute

$$\begin{aligned} \overline{\phi f \phi' \phi \phi'} &= \overline{\phi \phi' \overline{\phi f \phi' \phi \phi'}} \\ &= \overline{\phi \phi' \overline{\phi f \phi'}} \\ &= \overline{\phi \phi' \overline{f \phi'}} \\ &= \overline{\phi f \phi'} \end{aligned}$$

where we have used totality of ϕ to remove it from the restriction. \square

It is not difficult to show that the assignment described in the Lemma is in fact functorial. Moreover, if (ϕ, ϕ') and (ψ, ψ') are both faithful simulations $\mathbb{A} \rightarrow \mathbb{B}$ for which $\phi \vdash \psi$, then we have a \mathbb{B} -computable l such that $l \circ \phi = \psi$. Thus it satisfies $\psi \psi' l \phi \phi' = \psi \phi'$, which then is \mathbb{B} -computable. But that means that $\psi \phi'$ constitutes a morphism $(\underline{1}, \phi \phi') \rightarrow (\underline{1}, \psi \psi')$ in the category $\mathcal{K}(\text{Comp}(\mathbb{B}))$. This will be the component of the induced natural transformation at $\underline{1}$, and the same idea works to define the components at other objects.

We can now derive the desired result:

Proposition 5.10. *If two PCAs \mathbb{A} and \mathbb{B} are equivalent (as objects in the 2-category of PCAs, simulations and transformations over a fixed base category \mathcal{C}) then the corresponding Turing categories $\text{Comp}(\mathbb{A})$ and $\text{Comp}(\mathbb{B})$ are Morita-equivalent as categories.*

Proof. Lemma 5.6 and 5.8 together tell that we may take ϕ, ψ to be faithful; Lemma 5.9 then gives induced functors $\Theta : \mathcal{K}(\text{Comp}(\mathbb{A})) \rightarrow \mathcal{K}(\text{Comp}(\mathbb{B}))$ and $\Sigma : \mathcal{K}(\text{Comp}(\mathbb{B})) \rightarrow \mathcal{K}(\text{Comp}(\mathbb{A}))$; since this is 2-functorial the result is immediate. \square

This shows that the Turing category of a PCA is a good representation when we are interested in PCAs up to (strict) equivalence.

Note that, when we assume that the base category \mathcal{C} is split we have a diagram

$$\begin{array}{ccc} \mathcal{K}(\text{Comp}(\mathbb{A})) & \xrightarrow{\Theta} & \mathcal{K}(\text{Comp}(\mathbb{B})) \\ & \searrow F & \swarrow G \\ & \mathcal{C} & \end{array}$$

in which F and G are the obvious faithful functors, and which commutes up to a natural isomorphism $F \cong \Theta G$. (This natural isomorphism arises because in the base category \mathcal{C} there may be different objects which split the idempotent corresponding to $\phi \phi'$.) We shall say that the equivalence Θ is an *equivalence over \mathcal{C}* .

We have therefore shown that every equivalence of PCAs induces an equivalence over \mathcal{C} of the relative splittings of the Turing categories associated to the PCAs. The next question is obviously whether the equivalence of these categories is sufficient to conclude that the PCAs must be equivalent. Again the answer will follow from more abstract considerations presented in [11], and we will sketch a direct proof here. But before that, let us give an example in order to illustrate why it is important to regard equivalences over the base category as opposed to equivalences per se.

Let $\mathcal{C} = \text{Par} \times \text{Par}$. Given any PCA \mathbb{A} in Par , we can consider the PCAs $(\mathbb{A}, 1)$ and $(1, \mathbb{A})$ in \mathcal{C} . Now the associated categories $\text{Comp}(\mathbb{A}, 1)$ and $\text{Comp}(1, \mathbb{A})$ are not only equivalent, they are isomorphic (both are isomorphic to $\text{Comp}(\mathbb{A})$). But in \mathcal{C} , the PCAs $(\mathbb{A}, 1)$ and $(1, \mathbb{A})$ are not equivalent! Therefore, it is imperative that we are explicit about the base category.

The converse to Proposition 5.10 now reads:

Proposition 5.11. *Suppose \mathbb{A}, \mathbb{B} are PCAs in a split base category \mathcal{C} , inducing faithful functors $F : \mathcal{K}(\text{Comp}(\mathbb{A})) \rightarrow \mathcal{C}$ and $G : \mathcal{K}(\text{Comp}(\mathbb{B})) \rightarrow \mathcal{C}$. Then if Θ is an equivalence over \mathcal{C} , the PCAs \mathbb{A} and \mathbb{B} are equivalent.*

Proof. Suppose the equivalence over \mathcal{C} is given by a pair of cartesian restriction functors $\Theta : \mathcal{K}(\text{Comp}(\mathbb{A})) \rightarrow \mathcal{K}(\text{Comp}(\mathbb{B}))$ and $\Sigma : \mathcal{K}(\text{Comp}(\mathbb{B})) \rightarrow \mathcal{K}(\text{Comp}(\mathbb{A}))$, and suppose

that $\alpha : F \rightarrow G\Theta$ is a natural isomorphism. The Turing object \mathbb{A} in $\mathcal{K}(\text{Comp}(\mathbb{A}))$ (we really should write $\underline{1}$ for this object, but we will consider \mathbb{A} both as a PCA in \mathcal{C} and as a Turing object in $\mathcal{K}(\text{Comp}(\mathbb{A}))$) is sent by Θ to a PCA \mathbb{A}' in $\mathcal{K}(\text{Comp}(\mathbb{B}))$, and then by G to a PCA \mathbb{A}'' in \mathcal{C} . By Lemma 5.3, there is an induced isomorphism of PCAs $\alpha : \mathbb{A} \rightarrow \mathbb{A}''$. Since we are trying to prove that \mathbb{A} and \mathbb{B} are equivalent, we may just as well assume that $\mathbb{A} = \mathbb{A}''$ since they are isomorphic.

Now in $\mathcal{K}(\text{Comp}(\mathbb{B}))$, the PCA \mathbb{A}' is a retract of the Turing object and hence is simulated by the Turing object. Applying G to this simulation gives a simulation $\phi : \mathbb{A} \rightarrow \mathbb{B}$ in \mathcal{C} . There is a retraction map $\phi' : B \rightarrow A$, which already lives in $\mathcal{K}(\text{Comp}(\mathbb{B}))$ and thus is \mathbb{B} -computable.

Switching the roles of \mathbb{A} and \mathbb{B} , we find a faithful simulation $\psi : \mathbb{B} \rightarrow \mathbb{A}$, with retraction ψ' .

It remains to be proved that the composites $\psi\phi : \mathbb{A} \rightarrow \mathbb{A}$ and $\phi\psi : \mathbb{B} \rightarrow \mathbb{B}$ are isomorphic to the identity. We prove this for $\psi\phi$, the other case being the same.

First, $\psi\phi$ is \mathbb{A} -computable: applying Σ to ϕ , we get a map $\Sigma\Theta(\mathbb{A}) \rightarrow \Sigma(\mathbb{B})$, which is \mathbb{A} -computable. But then also the composite $\psi\Sigma(\phi)$ is \mathbb{A} -computable, and the image under F of this composite is (modulo computable isomorphism) $\psi\phi$.

Finally, to see that $\psi\phi$ has an \mathbb{A} -computable retraction, we consider the retraction ϕ' of ϕ and apply Σ : composing with the retraction ψ' of ψ we get an \mathbb{A} -computable retraction $\Sigma(\phi')\psi'$ of $\psi\Sigma(\phi)$, whose image under F is (again modulo computable isomorphism) a retraction of $\psi\phi$. \square

5.3. PCAs in a Turing category

We have seen that every PCA in a given Turing category is necessarily simulated by the Turing object, and that, in addition, one can always find a simulation which is part of a computable embedding-retraction pair. But clearly, not every PCA in a Turing category will also be a Turing object (for example, the terminal object is trivially a PCA but will in general not be a Turing object). Thus we wish to characterize which PCAs are Turing objects.

Our next result gives necessary and sufficient conditions for this to happen. Note that the point of the result is not to characterize when a given object which happens to be a PCA turns out to be a Turing object (this was already taken care of by Lemma 3.3), but to characterize when a given morphism is a Turing morphism.

Proposition 5.12. *Let \mathcal{C} be a Turing category, $\mathbb{A} = (A, \bullet_A)$ be a Turing object and let $\mathbb{B} = (B, \bullet_B)$ be a PCA in \mathcal{C} . Then \mathbb{B} is a Turing object (with Turing morphism \bullet_B) if and only if the following conditions are met:*

- there is an embedding-retraction pair $(\psi, \psi') : B \triangleleft A$;
- there is a simulation $\phi : \mathbb{A} \rightarrow \mathbb{B}$;
- the composite $\phi \circ \psi$ is \mathbb{B} -computable;
- there is a \mathbb{B} -computable retraction $\gamma : B \rightarrow B$ of $\phi \circ \psi$.

Proof. We begin by noting that all conditions are necessary, because by Lemma 5.6, the PCAs \mathbb{A} and \mathbb{B} are equivalent via a pair ϕ, ψ satisfying all the conditions.

Thus, assume that all conditions are satisfied, and let us prove that \bullet_B is a Turing morphism.

First, since the composite $\phi \circ \psi$ is \mathbb{B} -computable there is an index t such that $\bullet_B(t \times B) = \bullet_B(t \times \phi\phi')$. Next, since γ is assumed to be computable, there is an index $v : B \rightarrow B$ for $\gamma \bullet_B$. Finally, because ϕ is a simulation, there must be an index w such that $\bullet_B(w\phi \times \phi) = \phi(\bullet_A)$. All of t , v and w are total maps.

Now suppose we are given $f : X \times B \rightarrow B$ and we need to find an index for f relative to \mathbb{B} ; consider the diagram

$$\begin{array}{ccccc}
X \times B & \xrightarrow{f} & B & & \\
\uparrow X \times \psi' & & \downarrow \psi & & \\
X \times A & \xrightarrow{g \times A} & A \times A & \xrightarrow{\bullet} & A \\
\uparrow X \times \psi & & \downarrow w\phi \times \phi & & \downarrow \phi \\
X \times B & \xrightarrow{(w\phi g) \times (\phi\phi')} & B \times B & \xrightarrow{\bullet} & B \\
\parallel & & \downarrow v \times B & & \downarrow \gamma \\
X \times B & \xrightarrow{(vw\phi g) \times (\phi\phi')} & B \times B & \xrightarrow{\bullet} & B
\end{array}$$

(3) (4) (2) (1)

Diagram (1) commutes because we took v to be an index for $\gamma \bullet_B$, and diagram (2) expresses the fact that ϕ is a simulation via w , so this commutes as well. Now we use the fact that A is a Turing object to find an index g for the composite $\psi f(X \times \psi')$; thus by construction diagram (3) commutes. The commutativity of (4) and (5) is obvious.

Since $\psi' \circ \psi = 1$ and $\gamma \circ \phi \circ \phi' = 1$, the outermost composite from $X \times B$ to B is equal to f . Thus the diagram shows that we have factored f as $\bullet_B \circ ((vw\phi g) \times (\phi\phi'))$. However, we need to factor f as $\bullet_B \circ (h \times B)$, for some total h . Now use the index t defined above, to define h as the composite

$$X \times B \xrightarrow{(vw\phi g) \times B} B \times B \xrightarrow{t \times B} B \times B.$$

This composite is a total map since each of the constituents is. Moreover, we have the equalities

$$\bullet_B(h \times B) = \bullet_B(t(vw\phi g) \times B) = \bullet_B((vw\phi g) \times (\phi\phi')) = f,$$

as required. \square

Note that we have used combinatory completeness of \mathbb{B} in an essential way, so that this result does not characterize which applicative systems are Turing objects.

6. Conclusion and further work

We have attempted to outline the basic structural theory of Turing categories and the close connection with PCAs. However, even though we introduced morphisms of PCAs, we have not said anything about functoriality of the construction of a Turing category from a PCA. This issue will be studied in full detail in [11]; there, we will define categorical simulations, which directly generalize simulations of PCAs. Interestingly, these can be described in terms of morphisms of generalized categories of assemblies (which, in turn, are constructed using a generalization of the \mathcal{F} -construction by Robinson and Rosolini in [41]). It can then be shown that the 2-category of relative PCAs in a given base category is equivalent to the 2-category of Turing categories over the base.

Of course, one of the central points of Turing categories is the possibility of doing recursion theory in them. We will present an account of elementary recursion theory (basic undecidability and inseparability results, Rice’s theorem, etc.) in a forthcoming paper; to a certain extent, this will be a new presentation of the work in [38], but with a number of noticeable differences: first, we will advocate the use of the internal language of Turing categories in order to make the proofs more perspicuous. Indeed, many complicated diagram chases reduce to the kind of equational reasoning one may find in an introductory text on recursion theory. Moreover, we can present strengthenings of some of the results by looking carefully at what additional structure or properties is required of the ambient Turing category. It has to be noted that there are inherent limitations to the kinds of results one can obtain: for example, even in the classical Turing category over \mathbb{N} , we can only work with r.e. sets and computable functions. So, if we wish to study results about, say, non-r.e. sets or oracle computations then we have to move to a bigger category.

A similar inherent limitation to Turing categories lies in its essential untypedness. Recently, Longley ([28]) has advocated the use of typed PCAs in order to clarify notions of computation at higher types; it is not unimaginable that the corresponding generalization of Turing categories can be of interest if we wish to handle such notions of computation in our setting. Such a generalization would essentially bring us back to Birkedal’s weakly closed partial cartesian categories. At any rate, the banal observation that within the classical computability category $\mathbf{Comp}(\mathbb{K}_1)$ it is already impossible to talk about functionals at type 2, simply because the necessary objects aren’t there, indicates that we will have to consider widenings of the notion of Turing category once we wish to give a categorical account of such functionals.

There is a line of research, originated by Lambek, in which one takes natural number objects as a basic notion, and tries to characterize what functions are computable in a given category with NNO. Of special interest are free categories, such as the free cartesian closed category with NNO, or the free topos with NNO. In this context, we should also mention Joyal’s arithmetic universes, which are free categories based on primitive recursive arithmetic.

While this approach is distinct from the one discussed in this paper, we do think that there are worthwhile connections. In particular, one may always ask whether a natural number object in a given category is a PCA; so at the very least, the situations studied should give rise to interesting examples of Turing categories which live over categories other than \mathbf{Par} .

The classifying category for partial combinatory logic seems to be of interest not just because of the fact that it is, in a certain sense, the generic Turing category, but also because it has structural properties which are worthwhile in themselves. In order to study this category we will develop the internal language for various classes of restriction categories. In the case of PCL, we also look into the rewriting theory. There is an important structural result which expresses the classifying category of PCL as a semidirect product of the classifying category for ordinary CL with a suitable fibration, which arises from the rewriting theory.

Finally, the case of cartesian closed Turing categories is of interest; as indicated before,

this is, in essence, the study of models of the (partial) lambda calculus; such models may be called partial lambda algebras. While the basic theory of these (including the relation to PCAs) has been worked out, little seems to be known about specific examples.

References

- [1] S. Abramsky, A. Jung, Handbook for Logic in Computer Science, vol. 3, chap. Domain theory, Clarendon Press, Oxford, 1994.
- [2] A. Asperti, A. Ciabatonni, Effective applicative structures, in: Category Theory and Computer Science, 1995.
URL citeseer.ist.psu.edu/asperti95effective.html
- [3] S. Awodey, L. Birkedal, D. Scott, Local realizability toposes and a modal logic for computability, presented at *Tutorial Workshop on Realizability Semantics, FLoC'99*, Trento, Italy 1999. (1999).
- [4] I. Bethke, Notes on partial combinatory algebras, Ph.D. thesis, Universiteit van Amsterdam (1988).
- [5] I. Bethke, J. W. Klop, Collapsing partial combinatory algebras, in: G. Dowek, J. Heering, K. Meinke, B. Moller (eds.), Higher-Order Algebra, Logic, and Term Rewriting, vol. 1074 of LNCS, Springer Verlag, 1996, pp. 57–73.
URL citeseer.ist.psu.edu/bethke96collapsing.html
- [6] I. Bethke, J. W. Klop, R. de Vrijer, Completing partial combinatory algebras with unique head-normal forms, in: Logic in Computer Science, IEEE Computer Society Press, 1996, pp. 448–545.
URL citeseer.ist.psu.edu/bethke96completing.html
- [7] L. Birkedal, Developing theories of types and computability via realizability, Electronic Notes in Theoretical Computer Science 34, available at <http://www.elsevier.nl/locate/entcs/volume34.html>. The pdf version has active hyperreferences and is therefore the preferred version for reading online.
- [8] L. Birkedal, J. van Oosten, Relative and modified relative realizability, Annals of Pure and Applied Logic 118 (2001) 115–132.
- [9] J. Cockett, S. Lack, Restriction categories I, Theoretical Computer Science 270 (2002) 223–259.
- [10] J. Cockett, S. Lack, Restriction categories II: partial map classification, Theoretical Computer Science 294 (2003) 61–102.
- [11] J. R. B. Cockett, P. J. W. Hofstra, Categorical simulations, in preparation.
- [12] A. Corradini, F. Gadducci, A functorial semantics for multi-algebras and partial algebras, with applications to syntax, Theoretical Computer Science 286 (2) (2002) 293–322.
- [13] P.-L. Curien, A. Obtulowitz, Partiality, cartesian closedness and toposes, Information and Computation 80 (1989) 50–95.
- [14] S. Eilenberg, C. C. Elgot, Recursiveness, Academic Press, 1970.
- [15] J. E. Fenstad, General recursion theory, Springer Verlag, 1980.
- [16] M. Fitting, Fundamentals of generalized recursion theory, vol. 105 of Studies in Logic and the Foundations of Mathematics, North-Holland, 1981.
- [17] E. R. Griffor (ed.), Handbook of Computability Theory, vol. 140 of Studies in Logic, North-Holland, 1999.
- [18] A. Heller, An existence theorem for recursion categories, Journal of Symbolic Logic 55 (3) (1990) 1252–1268.
- [19] P. Hofstra, All realizability is relative, Mathematical Proceedings of the Cambridge Philosophical Society 141 (2006) 239–264.
- [20] P. Hofstra, J. van Oosten, Ordered partial combinatory algebras, Mathematical Proceedings of the Cambridge Philosophical Society 134 (2003) 445–463.
- [21] J. Hyland, The effective topos, in: A. Troelstra, D. V. Dalen (eds.), The L.E.J. Brouwer Centenary Symposium, North Holland Publishing Company, 1982.
- [22] G. Kreisel, Some reasons for generalizing recursion theory, in: G. et. al. (ed.), Logic Colloquium '69, North-Holland, 1971.
- [23] J. Lambek, P. J. Scott, Introduction to higher order categorical logic, vol. 7 of Cambridge studies in advanced mathematics, Cambridge University Press, 1986.
- [24] F. Lengyel, More existence theorems for recursion categories, Annals of Pure and Applied Logic 125 (1–3) (2004) 1–41.

- [25] F. Lengyel, Locally connected recursion categories, Tech. rep., CUNY Ph.D. program in computer science (2006).
- [26] P. Lietz, T. Streicher, Impredicativity entails untypedness, in: L. Birkedal, J. van Oosten, G. Rosolini, D. Scott (eds.), Tutorial workshop on realizability semantics, FloC'99, Trento, Italy, No. 23 in Electronic notes in theoretical computer science, Elsevier, 1999.
- [27] J. Longley, Realizability toposes and language semantics, Ph.D. thesis, University of Edinburgh (1994).
- [28] J. Longley, On the ubiquity of certain total type structures, to appear in: *Mathematical Structures in Computer Science* (2006).
- [29] G. Longo, E. Moggi, Cartesian closed categories of enumerations and effective type structures, in: Khan, MacQueen, Plotkin (eds.), *Symposium on semantics of data types*, vol. 173 of LNCS, Springer Verlag, 1984.
- [30] G. Longo, E. Moggi, Gödel numberings, principal morphisms, combinatory algebras, in: Chytil, Koubek (eds.), *Mathematical Structures in Computer Science*, vol. 176 of LNCS, Springer Verlag, Prague, 1984, preliminary version.
- [31] G. Longo, E. Moggi, A category theoretic characterization of functional completeness, *Theoretical Computer Science* 70 (2) (1990) 193–211.
- [32] E. Moggi, The partial lambda calculus, Ph.D. thesis, Edinburgh University (1988).
- [33] Y. N. Moschovakis, Abstract first order computability I, *Transactions of the American Mathematical Society* 138 (1969) 427–464.
- [34] Y. N. Moschovakis, Axioms for computation theories, in: G. et. al. (ed.), *Logic Colloquium '69*, North-Holland, 1971.
- [35] P. S. Mulry, Generalized Banach-Mazur functionals in the topos of recursive sets, *Journal of Pure and Applied Algebra* 26 (71–83).
- [36] P. Odifreddi, *Classical recursion theory*, vol. 125 of *Studies in Logic*, North-Holland, 1989.
- [37] E. Palmgren, S. Vickers, Partial Horn logic and cartesian categories, *Annals of Pure and Applied Logic* 145 (3) (2007) 314–353.
- [38] R. D. Paola, A. Heller, Dominical categories: recursion theory without elements, *Journal of Symbolic Logic* 52 (1987) 595–635.
- [39] R. D. Paola, F. Montagna, Some properties of the syntactic p-recursion categories generated by consistent, recursively enumerable extensions of Peano arithmetic, *Journal of Symbolic Logic* 56.
- [40] E. Robinson, G. Rosolini, Categories of partial maps, *Information and Computation* 79 (1988) 94–130.
- [41] E. Robinson, G. Rosolini, An abstract look at realizability, in: L. Fribourg (ed.), *Computer Science Logic, 15th International Workshop, CSL 2001. 10th Annual Conference of the EACSL*, Paris, France, September 10-13, 2001, *Proceedings*, vol. 2142 of *Lecture Notes in Computer Science*, Springer, 2001.
- [42] G. Rosolini, Continuity and effectiveness in topoi, Ph.D. thesis, Oxford University (1986).
- [43] G. Rosolini, Representation theorems for special p-categories, in: F. Borceux (ed.), *Categorical Algebra and its Applications*, vol. 1348 of *Lecture Notes in Mathematics*, Springer Verlag, 1988, pp. 307–315.
- [44] L. E. Sanchis, *Reflexive structures*, Springer Verlag, 1988.
- [45] L. Schröder, Classifying categories for partial equational logic, in: R. Blute (ed.), *Category Theory and Computer Science (CTCS02)*, vol. 69 of *Electronic Notes in Theoretical Computer Science*, Elsevier Science, 2003.
- [46] L. Schröder, The logic of the partial λ -calculus with equality, in: J. Marcinkowski, A. Tarlecki (eds.), *Computer Science Logic (CSL 04)*, vol. 3210 of *Lecture Notes in Computer Science*, Springer Verlag, 2004.
URL
<http://www.springerlink.com/openurl.asp?genre=article&issn=0302-9743&volume=3210&page=385>
- [47] S. Simpson, Short course on admissible recursion theory, in: Fenstad, Gandy, Sacks (eds.), *Generalized Recursion Theory II*, North-Holland, 1978.
- [48] S. Stefani, A relativization mechanism in recursion categories, *Journal of Symbolic Logic* 58 (4) (1993) 1251–1267.
- [49] H. R. Strong, Algebraically generalized recursive function theory, *IBM journal of Research and Development* 12 (1968) 465–475.

- [50] J. van Oosten, A general form of relative recursion, *Notre Dame Journal of Formal Logic* 46 (3) (2006) 311–318.
- [51] E. G. Wagner, Uniform reflexive structures: on the nature of Gödelization and relative computability, *Transactions of the American Mathematical Society* 144 (1969) 1–41.