Turing Categories and Computability

Amittai Siavava

06/04/2024

Contents

Introduction

In this paper, we construct a turing category $\Bbbk$ and study the resulting implications on computability.

In mathematics, it is often the case that structures present in one field are closely mirrored in another. For example, the idea of sets, maps and bijective maps in set theory is mirrored by the notion of groups, homomorphisms and isomorphisms in group theory, topological spaces, continuous maps and homeomorphisms in topology, vector spaces, linear maps and linear isomorphisms in linear algebra, and so on.

$$\underbrace{S}_{set} \xrightarrow{f} \underbrace{T}_{set} \qquad \underbrace{G}_{group} \xrightarrow{\varphi} \underbrace{H}_{group} \qquad \underbrace{X}_{topological space} \xrightarrow{\sigma} \underbrace{Y}_{topological space}$$

Category theory as a field of mathematics studies such structures and relationships between them, with the aim that, often, withdrawing from the specifics within a particular field can reveal more general shared structures, properties, and relationships [2].

In this paper, we explore some of the core concepts in category theory, then hone in on the notion of a turing category—an abstract model of computation based on category theory—and study some of its properties and their implications on computability.

1 Preliminaries

1.1 Categories

Definition 1.1. A ***category*** $\mathscr{A}$ consists of:

1. A collection $\mathbf{ob}(\mathscr{A})$ of objects;
2. For each pair of objects $A, B \in \mathbf{ob}(\mathscr{A})$, a set $\mathscr{A}(A, B)$ of ***arrows*** or ***morphisms*** or ***maps*** from A to B;
3. For each $A, B, C \in \mathbf{ob}(\mathscr{A})$, a function

$$\begin{aligned} \circ_{A,B,C} : \mathscr{A}(B, C) \times \mathscr{A}(A, B) &\to \mathscr{A}(A, C) \\ (f, g) &\mapsto f \circ g \end{aligned}$$

   called ***composition***; where $(f \circ g)(x) = f(g(x))$ for all $x \in A$.
4. For each $A \in \mathbf{ob}(\mathscr{A})$, a morphism $\mathsf{id}_A \in \mathscr{A}(A, A)$ called the ***identity*** on A;

such that the following axioms hold:

1. **associativity**: for all $f \in \mathscr{A}(A, B)$, $g \in \mathscr{A}(B, C)$, and $h \in \mathscr{A}(C, D)$, $(h \circ g) \circ f = h \circ (g \circ f)$.
2. **identity laws**: for all $f \in \mathscr{A}(A, B)$, $f \circ \mathsf{id}_A = f = \mathsf{id}_B \circ f$.

Remark 1.2. As simplifications, we write:
(a) $A \in \mathscr{A}$ to mean $A \in \mathbf{ob}(\mathscr{A})$;
(b) $f : A \to B$ or $A \xrightarrow{f} B$ to mean $f \in \mathscr{A}(A, B)$;
(c) fg for $f \circ g$;

$\Diamond$

Examples 1.3. (Categories)

1. There is a category Set, where
   (a) $\mathbf{ob}(\mathsf{Set})$ is the collection of all sets;
   (b) $\mathsf{Set}(A, B)$ is the set of all functions from A to B;

    (c) composition is ordinary function composition;

    (d) the identity on A is the identity function on A.

2. There is a category Grp, where

    (a) $\mathbf{ob}(\mathsf{Grp})$ is the collection of all groups;

    (b) $\mathsf{Grp}(G, H)$ is the set of all group homomorphisms from G to H;

    (c) composition is ordinary function composition;

    (d) the identity on G is the identity homomorphism on G.

3. There is a category Top of topological space and continuous maps.

4. For each field k, there is a category Vect_k of vector spaces over k and linear maps between them.

$\Diamond$

Definition 1.4. A map $f : A \to B$ in a category $\mathscr{A}$ is an ***isomorphism*** if there exists a map $g : B \to A$ such that $fg = \mathsf{id}_A$ and $gf = \mathsf{id}_B$. Ee call g the ***inverse*** of f and write $f^{-1} = g$, and say that A and B are ***isomorphic*** if there exists an isomorphism between them.

Examples 1.5. (Isomorphisms)

1. In Set, isomorphisms are bijections.

2. In Grp and Ring, isomorphisms are group and ring isomorphisms respectively.

3. In Vect_k, isomorphisms are linear isomorphisms.

$\Diamond$

1.2 Functors

Definition 1.6. Let $\mathscr{A}$ and $\mathscr{B}$ be categories. A ***functor*** $F : \mathscr{A} \to \mathscr{B}$ consists of:

1. A function $\mathbf{ob}(F) : \mathbf{ob}(\mathscr{A}) \to \mathbf{ob}(\mathscr{B})$;

2. For each $A, B \in \mathbf{ob}(\mathscr{A})$, a function

$$F : \mathscr{A}(A, B) \to \mathscr{B}(F(A), F(B))$$
$$f \mapsto F(f)$$

    such that:

    (a) $F(\mathsf{id}_A) = \mathsf{id}_{F(A)}$;

    (b) $F(fg) = F(f)F(g)$.

Essentially, a functor is a map between categories that preserves the structure of the categories. Since different categories have potentially different objects *and* different morphisms between objects, functors must map each object to some corresponding object and each morphism to some corresponding morphism such that the new objects and morphisms remain "faithful" to the original structure.

Examples 1.7. (Functors)

1. The ***identity functor*** $\mathsf{id}_{\mathscr{A}} : \mathscr{A} \to \mathscr{A}$ maps each object to itself and each morphism to itself.
2. The ***forgetful functor*** $U : \mathsf{Grp} \to \mathsf{Set}$ maps each group to its underlying set and each group homomorphism to its underlying function.
3. The ***free functor*** $F : \mathsf{Set} \to \mathsf{Grp}$ maps each set to the free group on that set and each function to the unique group homomorphism extending that function.

$\Diamond$

1.3 Restriction Structures

Definition 1.8. For a category $\mathscr{A}$, a ***restriction structure*** on $\mathscr{A}$ is an operation operation that assigns to each morphism $f : A \to B$ an morphism $\bar{f} : A \to A$ such that:

1. $\bar{f} \circ f = f$;
2. $\bar{f} \circ \bar{g} = \bar{g} \circ \bar{f}$ whenever $\mathbf{dom}\,(f) = \mathbf{dom}\,(g)$;
3. $\overline{f \circ \bar{g}} = \bar{g} \circ \bar{f}$ whenever $\mathbf{dom}\,(f) = \mathbf{dom}\,(g)$;
4. $\bar{g} \circ f = \bar{g} \circ f \circ \bar{g}$ whenever $\mathbf{dom}\,(f) = \mathbf{range}\,(g)$.

When a category has a restriction structure, we call it a ***restriction category***.

Examples 1.9. Here are a few examples of restriction categories. [4]

1. All categories admit the trivial restriction operation that maps $f : A \to B$ to $\bar{f} = \mathsf{id}_A$.
2. The category Par of partial functions between sets admits a restriction operation that maps $f : A \to B$ to $\bar{f} = \mathsf{id}_{\mathbf{dom}(f)}$.

The second example is of particular interest, since it provides a mechanism to model partial functions in a category-theoretic setting and ensure that each function has a well-defined domain. $\Diamond$

2 Turing Categories

A *turing category* is a specific categorification[1] of ***partial combinatory algebras***[2] based on restriction categories. Precisely, a Turing category is a cartesian restriction category $\mathscr{T}$ equipped with:

1. cartesian products — to pair (the codes of) data and programs;
2. a restriction structure representing the notion of partiality, since programs may not terminate on all inputs;
3. and a ***Turing object*** A — as some execution context or universal machine that understands and executes codes (morphisms) within the category. For any $X, Y \in \mathbf{ob}(\mathscr{T})$, there is a universal application morphism $\tau_{X,Y} : A \times X \to Y$ that represents the application of a program (in A) to data (in X) to produce a result (in Y). [3]

Remark 2.1. For example, in a Turing category TM of Turing machines, the Turing object A would be a universal Turing machine that takes as input some code e and data x, and simulates $\varphi_e(x)$. $\lozenge$

Turing categories provide an abstract framework for computability: a "category with partiality" equipped with a "universal computer", whose codes constitute the objects of interest. [3]

2.1 Properties of Turing Categories

Definition 2.2. Given two objects $A, B \in \mathscr{C}$, A is a ***retract*** of B if there exist morphisms $s : A \to B$ and $r : B \to A$ such that $r \circ s = \mathsf{id}_A$.

$$\begin{tikzcd} A \arrow[r, bend left, "s"] & B \arrow[l, bend left, "r"] \end{tikzcd}$$

In this case, s is called a ***section*** or *right inverse* of r, and r is called a ***retraction*** or *left inverse* of s. [1]

Lemma 2.3. In a Turing category $\mathscr{C}$ with a Turing object A, then A is a universal object in the sense that every object $B \in \mathscr{C}$ is a retract of A.

Proof. To show that every object $B \in \mathscr{C}$ is a retract of A, we need to exhibit a section $s : B \to A$ and a retraction $r : A \to B$. Since A is a Turing object, there exists a morphism $\tau_{B,A} : A \times B \to A$ that represents the application of a program (in A) to data (in B) to produce a result (in A). Define $s = \pi_2 : B \to A$ and $r = \pi_1 : A \to B$ such that $r \circ s = \pi_1 \circ \pi_2 = \mathsf{id}_A$.

revisit $\square$

[1] Categorification is the generalization of set-theoretic structures and properties to category theory (or from category theory to higher category theory).

[2] A partial combinatory algebra is a generalization of untyped lambda calculus (equivalently, Turing machines), allowing for application to be only partially defined.

Theorem 1. Given a partial combinatory algebra $\mathbb{A}$, the computable maps in $\mathbb{A}$ form a Turing category.

Proof. The proof is by construction. Given a partial combinatory algebra $\mathbb{A}$, we construct a Turing category $\mathscr{T}$ as follows:

1. The objects of $\mathscr{T}$ are the elements of $\mathbb{A}$.
2. The morphisms of $\mathscr{T}$ are the computable maps in $\mathbb{A}$.
3. The Turing object A is the universal machine in $\mathbb{A}$.
4. The application morphism $\tau_{X,Y} : A \times X \to Y$ is the application of a program (in A) to data (in X) to produce a result (in Y).

The resulting category $\mathscr{T}$ is a Turing category. □

Theorem 2. (Recognition Criterion for Turing Categories) Given a category C, the following are equivalent:

1. C is a Turing category.
2. There exists an object $A \in \mathbf{ob}(C)$ of which every other Turing object is a retract

2.2 Examples of Turing Categories

2.2.1 Classical Recursion Category

Consider an enumeration of partial computable functions $f : \omega \to \omega$ as $\varphi_0, \varphi_1, \varphi_2, \ldots$ since each such function consists of a finite number of instructions that can be encoded as a natural number. Alternatively, since the partial computable functions are exactly the Turing machine-computable ones, one may consider a coding for Turing machines.

Similarly, one can consider an enumeration of n-ary functions $f : \omega^n \to \omega$ as $\varphi_0^n, \varphi_1^n, \varphi_2^n, \ldots$.

We define the classical recursion category $\mathscr{R}$ with:

1. $\mathbf{ob}(\mathscr{R}) = \{\omega, \omega^2, \omega^3, \ldots\}$;
2. The Turing object is a universal machine Φ;
3. Morphisms are codes for partial computable ("recursive") functions with each i representing φ_i.

To distinguish between unary, binary, ternary, etc., functions, we denote $\Phi^{(n)}(e, x_1, \ldots, x_n)$ when the universal machine Φ interprets e as a code for an n-ary function and applies it to the arguments $x_1, \ldots, x_n$.

Remark 2.4. For the category to be well-formed, the coding scheme must be consistent. For example, if the Turing object interprets codes as Turing machines, then each code must encode a turing machine. Mixing codes for Turing machines with codes for register machines or codes for function instructions, for example, would break the structure of the category. ◊

Remark 2.5. When convenient (and purely for semantic reasons), we sometimes interchange the functions and machines as the morphisms in $\mathscr{R}$. ◊

Key properties of classical recursion include:

(a) By the construction of the category, the ***Universality Theorem*** holds. That is: for each $e, n \in \omega$,

$$\Phi^{(n)}(e, x_1, x_2, \ldots, x_n) = \varphi_e(x_1, x_2, \ldots, x_n).$$

(b) The ***s-m-n Theorem*** also holds. That is, there exists computable and injective functions s^n_m for each $m, n \in \omega$ such that

$$\Phi^{(m+n)}(x_1, x_2, \ldots, x_m, y_1, y_2, \ldots, y_n) = \Phi^{(n)}(s^n_m(e, x_1, x_2, \ldots, x_m), y_1, y_2, \ldots, y_n).$$

2.2.2 Variations of Classical Recursion

A slight variation of the classical recursion category above is obtained when we consider the category obtained by considering the c.e. subsets of $\omega^n, k > 0$ as objects and tuples of partial computable functions as morphisms. This category is denoted $\mathscr{R}_n$.

2.2.3 A Non-Example

When does a category not form a Turing category? One of the important requirements is that the morphisms within the category have a *suitable* enumeration. For example, consider the subcategory of total computable functions within the classical recursion category. This subcategory does not form a Turing category because no suitable enumeration can be constituted for the total functions.

Proof. Suppose we enumerate the total computable functions as $\varphi_0, \varphi_1, \varphi_2, \ldots$, Then, by diagonalization, we can always find a total computable function not in the enumeration: $f(e) = \varphi_e(e) + 1$. □

2.3 (Some) Applications of Turing Categories

2.3.1 Turing Equivalences

Lemma 2.6. Given A is a Turing object in C, then B is a Turing object if and only if A is a retract of B.

Proof. Since all Turing-complete models of computation are equivalent, given any two Turing objects A and B, there exists some computable isomorphism between them.
For example, for each register machine simulating some function f, there exists some Turing machine that simulates the same sequence of instructions needed to compute f.
For each $i \in \omega$, suppose ϕ_i is the Turing machine with coding i, and ψ_i is the register machine with

coding i, then for each ϕ_i there exists some ψ_k (with k not necessarily equal to i) such that $\phi_i \equiv \psi_k$. Define s and t such that $s(\phi_i) = \psi_k$ and $t(\psi_k) = \phi_i$. Then, s and t satisfy the conditions for a section and retraction. $\square$

This means that the coding scheme for any Turing object has to be consistent with the coding scheme for other valid Turing objects, since the models of computation are equivalent.
Furthermore, even when a category contains a finite finite number of codes, the Turing object itself must be capable of representing an infinite number of codes for it to be Turing-complete.

Definition 2.7. Two categories $\mathscr{C}$ and $\mathscr{D}$ are ***equivalent*** if there exists a pair of functors $F : \mathscr{C} \to \mathscr{D}$ and $G : \mathscr{D} \to \mathscr{C}$ such that $F \circ G \cong \mathsf{id}_{\mathscr{D}}$ and $G \circ F \cong \mathsf{id}_{\mathscr{C}}$, and all the laws of functors are satisfied.

Following from Lemma 2.6, any two Turing categories A and B encoding the same set of morphisms are equivalent. This means that whenever $\mathbf{ob}(A) \equiv \mathbf{ob}(B)$[1] then all properties consistent in A are also consistent in B.

Turing equivalences provide a way to compare the computability of different structures and constructs, and effectively equate different models of computation.

3 Further Areas of Exploration

This paper explores the basic notions around Turing categories as an abstract model of computation. To do this, it needs to go through a decent amount of underlying concepts.

To explore turing categories in an accessible manner, this paper needed to delve into some of the underlying concepts in category theory. This is also a restriction. For instance, the theory of ***Partial Combinatory Algebras*** (PCAs) is a big part of what is needed to fully understand Turing categories, but the field is too vast and complicated to be covered here.

If interested in further discovery around Turing categories and some of the results that can be derived from them, PCAs could be a good place to start.

[1]Not necessarily the same codes, but simulating the same functions.

References

[1] Andrea Asperti and Agata Ciabattoni, *Effective applicative structures*, Category theory and computer science, 1995, pp. 81–95.

[2] Tai-Danae Bradley, *What is category theory anyway?*, Math3ma, 2017.

[3] J.R.B. Cockett and P.J.W. Hofstra, *Introduction to turing categories*, Annals of Pure and Applied Logic **156** (2008), no. 2, 183–209.

[4] Tom Leinster, *Basic category theory*, 2016.