Available online at www.sciencedirect.com

SciVerse ScienceDirect

Electronic Notes in Theoretical Computer Science 286 (2012) 117–137

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

Timed Sets, Functional Complexity, and Computability

Robin Cockett [a,1,5] Joaquín Díaz-Boïls [b,4,6]
Jonathan Gallagher[a,3,7] Pavel Hrubeš [a,2,8]

[a] *Department of Computer Science*
University of Calgary
Calgary, Canada

[b] *Departamento de Lógica y Filosophía*
Universidad de Valencia
Valencia, Spain

Abstract

The construction of various categories of "timed sets" is described in which the timing of maps is considered modulo a "complexity order". The properties of these categories are developed: under appropriate conditions they form discrete, distributive restriction categories with an iteration. They provide a categorical basis for modeling functional complexity classes and allow the development of computability within these settings. Indeed, by considering "program objects" and the functions they compute, one can obtain models of computability – i.e. Turing categories – in which the total maps belong to specific complexity classes. Two examples of this are introduced in some detail which respectively have their total maps corresponding to PTIME and LOGSPACE.

Keywords: Restriction categories, complexity measures, functional complexity, computability, Turing categories

1 Introduction

The goal of this paper is to provide a general construction of categories of "timed sets" which may be used as a categorical basis for modeling functional complexity

[1] Supported by NSERC
[2] Supported by PIMS
[3] Supported by the University of Calgary
[4] Universidad de Valencia
[5] Email: robin@ucalgary.ca
[6] Email: ximo104@hotmail.com
[7] Email: jdgallag@ucalgary.ca
[8] Email: pahrubes@gmail.com

doi:10.1016/j.entcs.2012.08.009

classes. A motivating example concerns the modeling of polynomially timed functions: this example is of additional interest as it is possible to show one obtains a full model of computability – a Turing category [2,10] – in which the total functions are precisely the PTIME functions in the usual complexity sense. One may therefore identify the computability theory of this Turing category with the theory of PTIME complexity.

To achieve such a modeling, a number of ingredients must be collected. The first and simplest ingredient is a notion of "timing": here we take a very basic approach supposing that a (partial) map between sets is timed if for every input (on which the map defined) there is a measurement of the "running time" of the function on that input. Of course, "time" here could be replaced by any resource which one might wish to measure. Indeed, at the end of the paper we include a discussion using space as a resource in order to obtain a Turing category whose total functions belong to LOGSPACE.

Formally, the measurement of the resources used by a partial function, f, when applied to an element, x, in its domain, is given by associating to that element a value in a size monoid. The canonical example of a size monoid is, of course, the natural numbers, $\mathbb{N}$, under addition. Thus, for measuring time complexity, we associate to each x in the domain of f, a time $|x|_f \in \mathbb{N}$. If one had wished to measure space, a natural size monoid might be the natural numbers under maximum as the space usage of the composite of two functions is often[9] the maximum of the individual space usages.

Sets with "timed maps" in this sense certainly form a category in which two timed maps are considered to be equal if and only if they have *exactly* the same timing. Typically, however, in complexity theory one wants to regard maps whose timings are "similar" to be equal. So for example, if the functions have timings which asymptotically differ only by a linear factor (e.g. as in "big O" notation), one may want to regard them, from a complexity standpoint, to be equivalent. To capture this idea, which is fundamental to complexity theory, it is clear that one must pass to a quotient of the basic category of timed sets. However, there is a technical problem: one must describe precisely when two timed maps should be equivalent.

Unfortunately, one cannot simply use asymptotic behavior (e.g. "big O" notation) as, *a priori*, it is only defined for maps from the natural numbers to the real numbers. The timing functions in which we are interested have domain an arbitrary set and codomain an arbitrary size monoid. Thus, some other mechanism must be sought: we introduce here the notion of a *complexity order* to serve this purpose. A complexity order is a set of monotone endomorphisms of a size monoid, $\mathcal{C} \subseteq \mathsf{Mon}(M, M)$ satisfying certain properties. Given a complexity order, $\mathcal{C}$ one can preorder timed maps: one map, f, being of better $\mathcal{C}$-complexity than another, g, if it is defined whenever the latter is, and its cost is bounded by the latter's cost modified by a map from the order $\mathcal{C}$, that is $|x|_f \leq F(|x|_g)$ for all x in the domain

[9] For LOGSPACE it turns out (see later) that the appropriate size monoid is actually the natural numbers under addition: however, for example for PSPACE it does seem that maximum is more appropriate.

of g and for some $F \in \mathcal{C}$. The maps equivalent under this preorder are then taken as being equal.

Of course, for this to make good sense, $\mathcal{C}$ must satisfy some basic properties: it must certainly be closed to composition, but also, more technically, it must be "laxly additive." A canonical example of a complexity order is generated by the polynomial maps, $\mathcal{P} \subseteq \mathsf{Mon}(\mathbb{N}, \mathbb{N})$, on the natural numbers. Another important example, closely related to "big O" notation, is generated by the linear maps, $\mathcal{L} \subseteq \mathsf{Mon}(\mathbb{N}, \mathbb{N})$.

After the maps are quotiented by this $\mathcal{C}$-equivalence, a category of timed sets is obtained whose maps are partially ordered by their $\mathcal{C}$-complexity. If the complexity order, $\mathcal{C}$, is closed to addition – which certainly implies that $\mathcal{C}$ is laxly additive – something quite striking happens: the resulting category of timed sets naturally has the structure of a restriction category. Restriction categories are important as they provide a completely algebraic way of expressing partiality. The completeness theorem for restriction categories asserts that every restriction category is canonically a full subcategory of a partial map category: the canonical partial map category is actually formed by splitting the restriction idempotents.

In complexity theory, one always considers the running time of a function in terms of the size of its input. Yet, in the development so far, no mention of input size has been made. The effect of splitting the restriction idempotents elegantly corrects this defect. When one considers a timed map whose starting point is an idempotent the timing of that idempotent can be viewed as providing the "size" of its input. One can view the restriction idempotents as giving the cost of "reproducing" or "reading" the input both of which are, intuitively, measures of its size. In fact, these idempotents can provide a surprisingly sophisticated interpretation of size as witnessed by their use in modeling space complexity.

Splitting the idempotents has another remarkable effect. The *total* maps of this split category become exactly the maps whose complexity belongs to the complexity order, $\mathcal{C}$. Thus, for example, if $\mathcal{C} = \mathcal{P}$, then one obtains, as the total maps of the split category, exactly the polynomially timed maps.

This is striking: it is not at all obvious that there should be any relationship between complexity, given in this very concrete manner, and partiality. Yet, in a very straightforward way, one can exhibit a direct relationship. Of course, the polynomially timed maps, obtained in this manner, are by no means the PTIME maps as understood by complexity theorists. The category of timed sets allows *all* possible timings for all partial maps given between sets whether they are realizable by machine or not. To obtain the classical PTIME maps we need to link these ideas up with computability: for to be a PTIME map, one must also be able to realize the map by a computation on a machine.

This introduces a further technical problem: one must explain what a reasonable notion of computation is in timed sets. One approach to this is to view a computable map as one obtained by iterating a basic (total) machine step. For this approach to work requires that one can produce a well-defined notion of iteration – that is a trace on the coproduct – for categories of timed sets. To achieve this requires that the complexity order, $\mathcal{C}$, which determines the notion of equivalence, satisfies an additional requirement: namely, that the order is generated by maps which are "lax" with respect to addition. When this is the case there is a canonical notion of iteration which allows one to talk about iterating the state change of a machine (such as a Turing machine) which, in turn, allows one to "time" computations in a natural way.

In this manner one can obtain a categorical model of computability – in the sense of being a Turing category [2,10] – whose total maps are precisely the PTIME maps as standardly understood by complexity theorists. In particular, this suggests that one can identify the computability theory of this Turing category with the theory of PTIME complexity.

These results are explained in the last section: the development relies on a number of standard results from complexity theory and on a basic understanding of Turing categories. A similar technique allows one to obtain models of Turing categories of other low complexity classes such as ELEMENTARY, PRIM (primitive recursive). These techniques can also be applied when space is taken as the resource: to illustrate this we briefly discuss LOGSPACE computations which also organize themselves into a Turing category. Finally, we illustrate the possibility of interpreting complexity hierarchy problems as questions about functors between these Turing categories.

2 Timed Maps

A **size monoid**, $M = (U(M), +, 0, \leq)$ is an ordered commutative monoid (thus $(U(M), +, 0)$ is a commutative monoid, where $U(M)$ is the underlying set of M and $\leq$ a partial order on $U(M)$) such that

- $0 \leq m$ for all $m \in M$;
- if $a \leq a'$ and $b \leq b'$ then $a + b \leq a' + b'$.

A basic example of a size monoid is the natural numbers under addition. Of course, the natural numbers also form a size monoid under maximum and multiplication. In general, given any commutative monoid M we may place a preorder on it by setting $x \leq y$ if and only $\exists a.x + a = y$. The equivalence relation determined by $x \sim y \Leftrightarrow x \leq y \& y \leq x$ is then a congruence on the monoid: the quotient by this congruence is the universal size monoid associated with that commutative monoid. Formally this gives a left adjoint to the underlying functor, $\mathsf{SzMon} \to \mathsf{CMon}$, from size monoids to commutative monoids. Notice that under this adjunction the universal size monoid associated to a commutative group is always trivial: so size monoids are, in this sense, orthogonal to groups. In addition, this is a Galois adjunction: the category of size monoids such that $x \leq y \Leftrightarrow \exists a.x + a = y$ (one may think of these as "positive cones") forms a coreflective subcategory of sized monoids and a reflective subcategory of commutative monoids.

Given a size monoid M we can form a category $\mathsf{TSet}(M)$ [10] , the category of M-timed sets:

Objects: Sets;

Maps: A timed map $f : X \to Y$ is a pair $f = (U(f), |\_|_f)$ where $U(f) : X \to Y$ is a partial function and $|\_|_f : X \to M$ is the **timing** or **cost** of the map which is defined precisely when $U(f)$ is defined.

Composition: Given timed maps $f : X \to Y$ and $g : Y \to Z$ the composite fg has $U(fg) = U(f)U(g)$ and $|x|_{fg} = |x|_f + |f(x)|_g$;

Identities: The timed identity $1_X : X \to X$ has $U(1_X) = 1_X$ and $|x|_{1_X} = 0$ for all $x \in X$.

This is clearly a category whose maps have attached timings in the size monoid M. Two maps in this category are equal only if they have precisely the same timings. Our objective is to now relax this latter constraint: to do this we introduce the notion of a complexity order. A **complexity order** for a size monoid, M, is given by a class $\mathcal{C}$ of monotone endomorphisms (i.e., satisfying $P(x) \le P(y)$ whenever $x \le y$), $\mathcal{C} \subseteq \mathsf{Mon}(M, M)$, such that:

- $\mathcal{C}$ is **down closed**: if $P \in \mathcal{C}$ and $Q \in \mathsf{Mon}(M, M)$ with $Q(m) \le P(m)$ for every $m \in M$ then $Q \in \mathcal{C}$.
- $\mathcal{C}$ is closed to **composition**: the identity map, $I(x) = x$ is in $\mathcal{C}$ and, if $P, Q \in \mathcal{C}$ then $PQ \in \mathcal{C}$ (where $PQ(x) = Q(P(x))$ – as we are writing composition in diagrammatic order rather than applicative order).
- $\mathcal{C}$ is **laxly additive**: if $P_1, P_2 \in \mathcal{C}$ then there is a $Q \in \mathcal{C}$ such that for all x and y, $P_1(x) + P_2(y) \le Q(x + y)$.

We shall say a complexity order $\mathcal{C}$ is **additive** in case we replace the last axiom by the requirement that whenever $P, Q \in \mathcal{C}$ then $P + Q \in \mathcal{C}$. Clearly, being additive implies being laxly additive as $P_1(x)+P_2(y) \le P_1(x+y)+P_2(x+y) = (P_1+P_2)(x+y)$, so with this modification, it is still a complexity order. It is not hard to see, conversely, that a laxly additive order is additive if and only if it contains $I + I$.

Given a complexity order $\mathcal{C}$ and two M-timed partial maps maps $f, g : X \to Y$ we shall say f has **better $\mathcal{C}$-complexity** than g, denoted $f \le_{\mathcal{C}} g$, in case as partial maps $U(f) \ge U(g)$ (that is whenever $U(g)$ is defined $U(f)$ is defined and equal to $U(g)$) and there is a $F \in \mathcal{C}$ such that for all x for which g is defined $|x|_f \le F(|x|_g)$. [11] Intuitively, one should think that a map f has better complexity than g, when it is not only as least as defined as g, but also its timing is no worse than g's (up to a $\mathcal{C}$ increment). We say $f =_{\mathcal{C}} g$, that is f and g have the same $\mathcal{C}$-complexity if $f \le_{\mathcal{C}} g$ and $f \ge_{\mathcal{C}} g$. This means that $U(f) = U(g)$ and there are $P, Q \in \mathcal{C}$ such that $|x|_f \le P(|x|_g)$ and $|x|_g \le Q(|x|_f)$.

[10] There is a monad involved in this construction for which $\mathsf{TSet}(M)$ is the Kleisli category. Using a monad to model timing was described by Doug Gurr in his PhD thesis [7].

[11] It would be more in the spirit of the asymptotic definition of order to require $|x|_f \le P(|x|_g)$ holds *almost everywhere* – that is it holds for all but a finite set of x. While the development certainly works with this definition, as we shall see, most complexity orders already accommodate finitely many exceptions thus the gain of this complication is limited.

A complexity order $\mathcal{C}$ is said to be **generated** by a subset $\mathcal{S}$ of its maps, denoted $\mathcal{C} = \langle \mathcal{S} \rangle$, when every map in the order is dominated by one in the subset. Here are some examples of complexity orders:

(i) The smallest complexity order for any size monoid M is generated by the identity map. In this case $f \leq_{\mathcal{C}} g$ if and only if f is more defined than g and $|x|_f \leq |x|_g$ whenever $g(x)$ is defined.

(ii) Given any size monoid M there is always the **constant** complexity order generated by translations:

$$\mathcal{K}_M = \langle K_m | K_m = \lambda x.x + m, m \in M \rangle.$$

Here $f \leq_{\mathcal{K}_M} g$ if and only if f is more defined than g and there is an $m \in M$ such that for all $x \in X$ with $g(x)$ defined, $|x|_f \leq |x|_g + m$.

(iii) Given any size monoid M there is always the **linear** complexity order:

$$\mathcal{L}_M = \langle \lambda x.n \cdot x | n \cdot x = \underbrace{x + \ldots + x}_{n \text{ times}}, n \in \mathbb{N} \rangle.$$

When $M = \mathbb{N}$, note that $f \leq_{\mathcal{L}} g$ implies that, using "big O" notation, $f \in \mathcal{O}(g)$.

(iv) When $M = \mathbb{N}$ (or $M = \mathbb{R}_{\geq 0}$) then we may consider the orders:

$$\mathcal{P} = \langle \lambda x. \sum_{i=0}^{n} a_i x^i | a_i, n \in \mathbb{N} \rangle \qquad \mathcal{P}^* = \langle \lambda x. \sum_{i=1}^{n} a_i x^i | a_i, n \in \mathbb{N} \rangle.$$

These are the polynomial complexity orders. Note that the second does not include the constant functions. If $M = \mathbb{N}$, the sums are not necessary as they are dominated by the largest power. Hence, for example, $\mathcal{P}^* = \left\langle \lambda x.ax^i | a, i \in \mathbb{N}, i > 0 \right\rangle$.

We start by observing:

Lemma 2.1 *Given any complexity order, $\mathcal{C}$, on a size monoid M:*

(i) Each homset of $\mathsf{TSet}(M)(X,Y)$ *is preordered by $f \leq_{\mathcal{C}} g$;*

(ii) If $f \leq_{\mathcal{C}} g$ and $h \leq_{\mathcal{C}} k$ in $\mathsf{TSet}(M)$ *then , $fh \leq_{\mathcal{C}} gk$ (i.e. this is a preorder enrichment for* $\mathsf{TSet}(M)$*);*

(iii) $\_ =_{\mathcal{C}} \_$ is a congruence on $\mathsf{TSet}(M)$.

Proof.

(i) It is immediate that $f \leq_{\mathcal{C}} f$ the only difficulty is to show transitivity. So suppose $f \leq_{\mathcal{C}} g \leq_{\mathcal{C}} h$ then for $P, Q \in \mathcal{C}$ we have $|x|_f \leq P(|x|_g)$ and $|x|_g \leq Q(|x|_h)$ but this means $|x|_f \leq P(|x|_g) \leq P(Q(|x|_h))$ as P is monotone. But $QP \in \mathcal{C}$ so we are done.

(ii) Suppose $|x|_f \leq P_1(|x|_g)$ and $|y|_h \leq P_2(|y|_k)$ whenever they are defined. Then, as $\mathcal{C}$ is laxly additive there is a $Q \in \mathcal{C}$ with $P_1(x) + P_2(y) \leq Q(x + y)$ and we have:

$$\begin{aligned} |x|_{fh} &= |x|_f + |f(x)|_h \\ &\leq P_1(|x|_g) + P_2(|f(x)|_k) \\ &\leq Q(|x|_g + |f(x)|_k) \\ &= Q(|x|_{gk}). \end{aligned}$$

(iii) $\_ =_{\mathcal{C}} \_$ is a congruence on $\mathsf{TSet}(M)$ as this is so for the equivalences in any preorder enriched category.

□

This means that we can pass to the category $\mathsf{TSet}(M)/\mathcal{C}$ where partial map equality is determined by $f =_{\mathcal{C}} g$. Thus, in $\mathsf{TSet}(\mathbb{N})/\mathcal{K}$ two timed maps will be counted as being equal if they are equal as partial maps and their timings differ by at most a constant. Similarly, in $\mathsf{TSet}(\mathbb{N})/\mathcal{L}$ two maps are counted to be equal if their timings are linearly comparable.

3 Timed Sets and Partial Map Categories

In this section we consider $\mathsf{TSet}(M)/\mathcal{C}$, where $\mathcal{C}$ is an additive complexity order. Some of the orders mentioned above are not additive: in particular, neither the order generated by the identity map, nor the order $\mathcal{K}_M$ generated by translations, is additive. On the other hand, the linear and polynomial orders, $\mathcal{L}, \mathcal{P}$ and $\mathcal{P}^*$, are additive.

The purpose of this section is to establish:

Proposition 3.1 *When $\mathcal{C}$ is additive then $\mathsf{TSet}(M)/\mathcal{C}$ is a discrete distributive restriction category with the restriction given by:*

$$\frac{f : X \to Y}{\overline{f} = (\overline{U(f)}, |\_|_f) : X \to X}$$

A **distributive restriction category** is a Cartesian restriction category – that is it has finite products in an appropriate partial sense (explained below) – with coproducts over which these products distribute (see [4]). It is **discrete** in case all diagonal maps $\Delta : A \to A \times A$ have partial inverses[12] $\Delta^{(-1)}$.

The significance of Proposition 3.1 lies in the fact that restriction categories are abstract categories of partial maps. In particular, *every* restriction category is a full subcategory of a (real) partial map category (see [3,11]). Furthermore, the canonical partial map category, of which it is a full subcategory, is obtained by splitting the "restriction" idempotents to obtain a larger restriction category. From this one can extract the subcategory of total maps which can be used to form the partial map category in which the original restriction category sits (see [3]).

[12] A map $f : A \to B$ in a restriction category has a partial inverse if there is a map $f^{(-1)} : B \to A$ with $ff^{(-1)} = \overline{f}$ and $f^{(-1)}f = \overline{f^{(-1)}}$: partial inverses are unique.

In section 5 we carry out these formal steps for $\mathsf{TSet}(M)/\mathcal{C}$. While abstractly these steps are quite routine, their realization in this case is quite striking. As we shall see, this allows the construction of a restriction category whose total maps are precisely those with timings bounded by the complexity order $\mathcal{C}$. This provides a direct link between complexity and partiality.

A **restriction category** is a category with a restriction combinator which, to any map, assigns an endomorphism of its domain:

$$\frac{f : A \to B}{\overline{f} : A \to A}$$

This must satisfy just four identities:

$$[\mathbf{R.1}]\ \overline{f}f = f \qquad [\mathbf{R.2}]\ \overline{f}\,\overline{g} = \overline{g}\,\overline{f}$$
$$[\mathbf{R.3}]\ \overline{f}\,\overline{g} = \overline{\overline{g}\,f} \quad [\mathbf{R.4}]\ f\,\overline{g} = \overline{fg}\,f$$

As $\overline{f}\ \overline{f} = \overline{\overline{f}\,f} = \overline{f}$, the restriction of a map is always an idempotent, called a **restriction idempotent**, which should be thought of as a partial identity. In a restriction category a map $f : A \to B$ is said to be **total** in case $\overline{f} = 1_A$: the total maps always form a subcategory.

The proof of proposition 3.1 can be broken into parts: proving that it is a restriction category, that it is Cartesian, distributive, and discrete. The verification that it is a restriction category is straightforward: we shall prove [**R.1**] and [**R.4**] as these show why the additive assumption is needed:

[**R.1**] We know already that the behavior at the level of partial maps is correct: the problem is to provide a bounding function from $\mathcal{C}$ for $\overline{f}f$ in terms of f. We have

$$|x|_{\overline{f}f} = |x|_{\overline{f}} + |x|_f = |x|_f + |x|_f \le (I+I)(|x|_f)$$

where we use the additivity to produce the bound.

[**R.4**] Again it is obtaining the bound for $\overline{fg}f$ in terms of that of $f\overline{g}$ which is the only difficulty. We have:

$$\begin{aligned}|x|_{\overline{fg}f} &= |x|_{\overline{fg}} + |x|_f = |x|_{fg} + |x|_f = 2\cdot|x|_f + |f(x)|_g \\ &\le 2\cdot(|x|_f + |f(x)|_g) = (I+I)(|x|_f + |f(x)|_g)\end{aligned}$$

Clearly, again, we have to use the fact that $\mathcal{C}$ is additive.

Thus, $\mathsf{TSet}(M)/\mathcal{C}$ is a restriction category. A restriction category is Cartesian when it has a restriction terminal object and restriction (binary) products. A **restriction terminal object** is an object 1 with, for each object A, a unique total map $!_A : A \to 1$ such that any map $f : A \to 1$ has $f = \overline{f}!_A$.

In $\mathsf{TSet}(M)/\mathcal{C}$ the terminal object is the one element set $1 = \{()\}$ and $!_A$ is determined by setting $U(!_A)(a) = ()$ for every $a \in A$ and $|a|_{!_A} = 0$. One then has, given an $f : A \to 1$, that whenever f is defined on a that $|a|_{\overline{f}!_A} = |a|_{\overline{f}} + |a|_{!_A} = |a|_f$.

An object, $A \times B$, with two total projections, $\pi_0 : A \times B \to A$ and $\pi_1 : A \times B \to B$, is a **restriction product** in case for any pair of maps $f : X \to A$ and $g : X \to B$

there is a unique map $\langle f,g\rangle : X \to A\times B$ such that $\langle f,g\rangle\,\pi_0 = \overline{g}f$ and $\langle f,g\rangle\,\pi_1 = \overline{f}g$.

In $\mathsf{TSet}(M)/\mathcal{C}$, the restriction product $A\times B$ is the Cartesian product of the sets, where π_i, $i = 0$ or 1, has $U(\pi_i)$ the appropriate projection in sets, and $|(x,y)|_{\pi_i} = 0$. We set $U(\langle f,g\rangle)$ to be the pairing map, which, on $x \in X$, is only defined when both f and g are defined, and set $|x|_{\langle f,g\rangle} = |x|_f + |x|_g$. To show that this is well-defined, that is that if $f =_{\mathcal{C}} f'$ and $g =_{\mathcal{C}} g'$ that $\langle f,g\rangle =_{\mathcal{C}} \langle f',g'\rangle$, it suffices to show that if $f \leq_{\mathcal{C}} f'$ and $g \leq_{\mathcal{C}} g'$ then $\langle f,g\rangle \leq_{\mathcal{C}} \langle f',g'\rangle$. So suppose $P_1, P_2 \in \mathcal{C}$ with $|x|_f \leq P_1(|x|_{f'})$ and $|x|_g \leq P_2(|x|_{g'})$ then

$$|x|_{\langle f,g\rangle} = |x|_f + |x|_g \leq P_1(|x|_{f'}) + P_2(|x|_{g'}) \leq Q(|x|_{f'} + |x|_{g'})\,,$$

where we are using the fact that $\mathcal{C}$ is laxly additive.

To verify that $\langle f,g\rangle\,\pi_0 = \overline{g}f$, first recall that this is so for the underlying partial maps between sets, so we need only check the bounds. However, for this we have:

$$|x|_{\langle f,g\rangle\pi_0} = |x|_{\langle f,g\rangle} + |(f(x),g(x))|_{\pi_0} = |x|_{\langle f,g\rangle} = |x|_f + |x|_g = |x|_{\overline{g}} + |x|_f = |x|_{\overline{g}f}.$$

To establish uniqueness, suppose $h : X \to A\times B$ satisfies these conditions, then certainly $U(h) = U(\langle f,g\rangle)$; so that it remains to show that there are $P, Q \in \mathcal{C}$ with $|x|_{\langle f,g\rangle} \leq P(|x|_h)$ and $|x|_h \leq Q(|x|_{\langle f,g\rangle})$. We have $h\pi_0 = \overline{g}f$ so we have a P and Q such that $|x|_{h\pi_0} \leq P(|x|_{\overline{g}f})$ and $|x|_{\overline{g}f} \leq Q(|x|_{h\pi_0})$. Now observe $|x|_{\overline{g}f} = |x|_f + |x|_g = |x|_{\langle f,g\rangle}$ and $|x|_{h\pi_0} = |x|_h + |h(x)|_{\pi_0} = |x|_h$, so that these bounds suffice.

We have now established that $\mathsf{TSet}(M)/\mathcal{C}$ is a Cartesian restriction category.

Given this structure, the diagonal map $\Delta = \langle 1_A, 1_A\rangle : A \to A\times A$ is clearly a zero cost map and so has a zero cost partial inverse whose domain of definition is exactly the diagonal elements and which has $\Delta^{(-1)}(x,x) = x$.

Finally, we must show that $\mathsf{TSet}(M)/\mathcal{C}$ has coproducts: the empty set is clearly the initial object. The binary coproduct $A+B$ is just the disjoint union of the sets: the injections σ_0 and σ_1 are the inclusion maps in sets with a zero timing. The copairing map for $f : A \to X$ and $g : B \to X$ is the map $\langle f|g\rangle : A+B \to X$, where $U(\langle f|g\rangle)$ is the usual copairing for sets and partial functions ($U(\langle f|g\rangle)(a) = f(a)$ for $a \in A$ and $U\,\langle f|g\rangle\,(b) = g(b)$ for $b \in B$) with timing $|a|_{\langle f|g\rangle} = |a|_f$ for $a \in A$ and $|b|_{\langle f|g\rangle} = |b|_g$ for $b \in B$. As before we must show this is well-defined: this requires finding a bounding function which works for both components[13] – and the sum of the bounding functions of the individual components will clearly work. It is now routine to show one has a coproduct. Furthermore, as the canonical map

$$\langle 1_A\times\sigma_0 | 1_A\times\sigma_1\rangle : A\times B + A\times C \to A\times(B+A)$$

has zero cost and is an isomorphism in sets, it follows easily that $\mathsf{TSet}(M)/\mathcal{C}$ is distributive.

This completes the proof of proposition 3.1.

[13] In order to make the product and coproduct well-defined, a simultaneous bound for *all* the components is required. While for finite products and coproducts this is simply given by adding the individual bounds, this technique does not work for infinite products or coproducts and, indeed, $\mathsf{TSet}(M)/\mathcal{C}$ does not have these limits.

It should be noted that in $\mathsf{TSet}(M)/\mathcal{C}$, the total maps are (up to equivalence) zero cost maps. Thus, even when $\mathcal{C} = \mathcal{P}$ this allows only maps with timings which are bounded by a constant to be total. Furthermore, these maps take no account of the size of their inputs (or outputs for that matter). These are major defects from the complexity perspective: however, as we shall see shortly, they are all corrected by the formal construction of splitting idempotents (see section 5).

4 Joins, Ranges, and Iteration

Given any restriction category there is always an induced restriction partial order on parallel maps; it is defined by $f \leq g$ if and only if $\overline{f}g = f$. This makes any restriction category a partial order enriched category. Observe first that this restriction ordering is actually the converse of the complexity ordering:

Lemma 4.1 *In $\mathsf{TSet}(M)/\mathcal{C}$, for additive $\mathcal{C}$, $f \leq g$ in the restriction order if and only if $g \leq_{\mathcal{C}} f$ in the complexity order.*

Proof. For suppose $f \leq g$ then $\overline{f}g = f$ so $U(g)$ is at least as defined as $U(f)$ but, furthermore, on the timing we have a $P \in \mathcal{C}$ such that

$$|x|_g \leq |x|_f + |x|_g = |x|_{\overline{f}g} \leq P(|x|_f)$$

it follows that g has better $\mathcal{C}$-complexity than f (i.e. $g \leq_{\mathcal{C}} f$). Conversely, suppose $g \leq_{\mathcal{C}} f$ then $U(f) \leq U(g)$ and there is a $P' \in \mathcal{C}$ such that $|x|_g \leq P'(|x|_f)$. It follows then that

$$|x|_{\overline{f}g} = |x|_f + |x|_g \leq |x|_f + P'(|x|_f) = (I + P')(|x|_f)$$

where the desired bound is $I + P'$ which, as $\mathcal{C}$ is additive, is certainly in $\mathcal{C}$. □

In any restriction category we say that two parallel maps f and g are **compatible**, $f \smile g$, in case $\overline{f}g = \overline{g}f$. In sets and partial maps this means that they take the same value when they are both defined: thus, clearly, the join of the partial maps in sets, $f \vee g$, exists. It is not the case that in $\mathsf{TSet}(M)/\mathcal{C}$, for an arbitrary M, that one can join maps in this manner. Technically a restriction category has **joins** if whenever f and g are parallel maps with $f \smile g$ then there is a parallel map $f \vee g$ such that, with respect to the restriction order, it is the join – that is $f \leq f \vee g$, $g \leq f \vee g$, and if $f \leq h$ and $g \leq h$ then $f \vee g \leq h$ – and that join is stable – that is $h(f \vee g) = (hf) \vee (hg)$: in a restriction category this then implies universality, $(f \vee g)k = (fk) \vee (gk)$.

In fact, from the general theory [4], *any* distributive restriction category is an "extensive" restriction category. Extensive restriction categories always have *disjoint* joins of partial maps. This provides:

Proposition 4.2 *When $\mathcal{C}$ is additive, $\mathsf{TSet}(M)/\mathcal{C}$ has finite disjoint joins of parallel maps.*

This means that when $f, g : A \to B$ are parallel maps with disjoint domains, that is $\overline{f}g = \emptyset = \overline{g}f$, where $\emptyset$ is the empty partial map then the join of f and g exists: it is written $f \sqcup g$ to emphasize the disjointness. For a general M it will not be the case that joins of compatible maps exist, however, if we assume M has *binary* infima, which are preserved by addition (in the sense that $(x \wedge y) + z = (x + z) \wedge (y + z)$), then we have:

Proposition 4.3 *When M has binary infima which are preserved by addition and $\mathcal{C}$ is additive, then $\mathsf{TSet}(M)/\mathcal{C}$ has finite joins of all compatible maps.*

Proof. We define $f \vee g$ to have $U(f \vee g) = U(f) \vee U(g)$ and

$$|x|_{f \vee g} = \begin{cases} |x|_f & f(x)\downarrow \text{ and } g(x)\uparrow \\ |x|_g & f(x)\uparrow \text{ and } g(x)\downarrow \\ |x|_f \wedge |x|_g & f(x)\downarrow \text{ and } g(x)\downarrow \end{cases}$$

It is left to the reader to check this is well-defined. That it has all the required properties is also straightforward to check. □

Certainly when $M = \mathbb{N}$ the requirements[14] of Proposition 4.3 are satisfied so, in this case, we have finite joins of compatible maps. As $\mathbb{N}$ is well ordered, we may also form "ranges" [10,5]: the range associates to each map $f : A \to B$ a restriction idempotent $\widehat{f} : B \to B$ such that $f\widehat{f} = f$, $\widehat{fg} = \widehat{\widehat{f}g}$, and $\widehat{f\overline{g}} = \widehat{f}\overline{g}$. A range operator provides an algebraic description of the image of a map. Sets and partial maps have ranges because every map has a partial section. For an additive $\mathcal{C}$, $\mathsf{TSet}(\mathbb{N})/\mathcal{C}$, also has partial sections for every map: however, the section must be chosen carefully!

We briefly outline these ideas. Given a partial map $f : A \to B$ a partial section is a map $g : B \to A$ such that $gf = \overline{g}$ and $fgf = f$: if a map has a partial section g then $\widehat{f} = \overline{g}$. Given $f : A \to B$ in $\mathsf{TSet}(\mathbb{N})/\mathcal{C}$ we may define a zero cost section g so that $g(y)$ is defined only if $y = f(x)$ for some $x \in X$, further, when it is defined, $f(g(y)) = y$ and $|g(y)|_f = \mathsf{min}\{|x|_f \mid f(x) = y\}$ (here we use the well-ordering of $\mathbb{N}$). This is last requirement is necessary as can be seen from the proof that $fgf = f$. The key steps in the bounding argument are

$$|x|_{fgf} = |x|_f + |f(x)|_g + |g(f(x))|_f = |x|_f + 0 + |g(f(x))|_f \le |x|_f + |x|_f$$

where the last step is possible because of the way we chose $g(y)$ to have $|g(y)|_f$ minimal. We state without further justification – because it is quite remarkable that these categories are so well-structured:

Proposition 4.4 *When $\mathcal{C}$ is additive, $\mathsf{TSet}(\mathbb{N})/\mathcal{C}$ is a discrete range restriction category with finite joins.*

The last aspect of structure we require from these categories is that it should be possible to iterate maps. A category has iteration precisely when it is traced on

[14] In this case, as $\mathbb{N}$ is well-ordered, it may be tempting to think that as we have arbitrary non-empty infima that we will have *arbitrary* joins for any set of compatible parallel maps (i.e. not just for finite sets). However, this is *not* the case: the join of an infinite collection will in general not be well-defined as finding uniform bounds is only possible for finite families.

the coproduct [8,9]. The iteration combinator has the following form:

$$\frac{h : A \to A + B}{h^\dagger : A \to B}\ \text{Iteration}$$

However, in an extensive restriction category any map $h : A \to A+B$ can be broken down into two disjoint maps $h = f \sqcup g : A \to A+B$ where $f : A \to A$ and $g : A \to B$ allowing the iteration to be re-expressed as a "Kleene wand":

$$\frac{f : A \to A \quad g : A \to B \quad f \perp g}{f \dagger g : A \to B}\ \text{Kleene Wand}$$

where $f \perp g$ means $\overline{f}\,\overline{g} = \emptyset$. Intuitively f is iterated until the guard g is encountered at which point an output is produced. In sets and partial functions the canonical Kleene wand may be expressed as $f \dagger g = \bigsqcup_{i=0}^{\infty} f^i g$. Of course, as both f and g are partial $f \dagger g$ will certainly be partial. However, note that there is an added source of partiality as the iteration of f may never hit the guard.

The Kleene wand, in order to correspond to a well-behaved trace, must satisfy some basic equations:

[W.1] When $f \perp h$ then $(fg)\dagger h = h \sqcup f((gf)\dagger(gh))$;

[W.2] When $f \perp g$, $g \perp h$, and $f \perp h$ then $(f \sqcup g)\dagger h = (f \dagger g)\dagger(f \dagger h)$;

[W.3] When $f \perp g$ then $(f \dagger g)h = f \dagger (gh)$;

[W.4] When $f \perp g$ then $1_A \times (f \dagger g) = (1_A \times f)\dagger(1_A \times g)$;

[W.5] When $f \leq f'$, $g \leq g'$, and $f' \perp g'$ then $f \dagger g \leq f' \dagger g'$.

The first identity allows finite unwindings of the iteration:

$$f \dagger g = g \sqcup (f \dagger fg) = g \sqcup fg \sqcup (f \dagger ffg) = ...$$

Note that the third identity tells us that $f \dagger g = (f \dagger \overline{g})g$ where $f \dagger \overline{g}$ provides a more primitive form in which $\overline{g}$ may be regarded as implementing a predicate guard. The fourth identity allows one to trace in a context A.

Sets and partial maps satisfy all these identities and we shall use this to establish that $\mathsf{TSet}(M)/\mathcal{C}$ also has iteration which is defined in the obvious manner: $f \dagger g(x) := g(f^n(x))$ when this is defined for some n (at most one n will work) and $|x|_{f \dagger g} := (\sum_{i=0}^{n-1} |f^i(x)|_f) + |f^n(x)|_g$. The only technical problem is then to show that iteration, as we have defined it, is actually well-defined with respect to the equivalence on maps. For this it is necessary to demand more of the complexity order $\mathcal{C}$.

We say that a complexity order is **lax** if it generated by a class of functions which are lax in the sense that $P(m)+P(n) \leq P(m+n)$ (we always have $0 \leq P(0)$). Note that sums and composites of lax functions are lax.

Proposition 4.5 *When $\mathcal{C}$ is additive and lax $\mathsf{TSet}(\mathbb{N})/\mathcal{C}$ has iteration.*

Proof. The equations all follow easily from the fact that sets and partial maps satisfy the equations. The only difficulty is to prove that if $f \leq_{\mathcal{C}} f'$ and $g \leq_{\mathcal{C}} g'$ then $f \dagger g \leq_{\mathcal{C}} f' \dagger g'$. Because of these inequalities we have:

$$\begin{aligned} &f'(x) \downarrow \Rightarrow |x|_f \leq P(|x|_{f'}) \\ &g'(x) \downarrow \Rightarrow |x|_g \leq Q(|x|_{g'}) \end{aligned}$$

Suppose now that $f' \dagger g'(x)$ is defined then clearly $f \dagger g(x)$ will be defined and in fact their evaluations on this element will be the same. That is there is a unique n so that $g(f^n(x)) = g'(f'^n(x))$ and they are both defined. Consider the cost of these:

$$\begin{aligned} |x|_{f \dagger g} &= |x|_{f^n g} \\ &= |x|_f + |f(x)|_f + ... + |f^n(x)|_g \\ &\leq P(|x|_{f'}) + P(|f'(x)|_{f'}) + ... + Q(|f'^n(x)|_{g'} \\ &\leq (P+Q)(|x|_{f'}) + (P+Q)(|f'(x)|_{f'}) + ... + (P+Q)(|f'^n(x)|_{g'} \\ &\leq (P+Q)(|x|_{f'} + |f'(x)|_{f'} + ... + |f'^n(x)|_{g'}) \\ &\leq (P+Q)(|x|_{f' \dagger g'}). \end{aligned}$$

□

The class $\mathcal{L}$ (linear time) is lax. Significantly the class $\mathcal{P}$ is not lax: the problem is that constant functions are not lax. However, the class $\mathcal{P}^*$ is lax as we removed the constant functions! Recall that constant functions are only important when the input sizes can be zero: we shall remove this possibility shortly.

It is worth remarking that categories of timed sets often have more than one trace. Consider, for example, $\mathsf{TSet}/\mathcal{L}$ where $\mathcal{L}$ is the linear order on the natural numbers with addition. As described above $\mathcal{L}$ is lax so that there is a trace as defined above. However, on any size monoid (such as $\mathbb{N}$) which has a maximum $\mathcal{L}$ is also an order for that monoid where addition is replaced by the maximum (e.g. for $\mathbb{N}$ regarded as size monoid with addition given by $\max(x,y)$). Furthermore, the two categories of timed sets are then actually *isomorphic*. However, significantly, the traces given by the above, with respect to addition and maximum, are not the same. This not only demonstrates that a category can have more than one trace but also is a reminder that when one wishes to talk of iteration one may have a choice of iteration!

5 Splitting Restriction Idempotents

Our next objective is to split the restriction idempotents of $\mathsf{TSet}(M)/\mathcal{C}$ to produce $\mathsf{Split}(\mathsf{TSet}(M)/\mathcal{C})$. This is a standard construction (also – for all idempotents – known as the Karoubi envelope or Cauchy completion) which turns the idempotents e and e' into objects and takes as the maps $f : e \to e'$ those maps in the original category such that $efe' = f$. The construction is remarkable as it preserves almost all the properties of the original category. In particular, if one starts with a distributive restriction category then the result will be a distributive restriction category. Furthermore, ranges, joins, and iteration are all transferred onto the splitting.

Let us consider in detail what an object of $\mathsf{Split}(\mathsf{TSet}(M)/\mathcal{C})$ looks like. Recall that a restriction idempotent is just a timed partial identity: it is easy to see that the elements which are not in the domain of the underlying partial identity will not play any role. So it suffices to consider the objects whose underlying partial identity is actually the identity: this makes the only important information the timing of the idempotent. Thus, such an object is essentially a set A with a map which assigns to each element $a \in A$ a "size" $\|a\|_e$ (these are called *sized sets* in [6]). A timed map $f : e \to e'$ between two sized sets, in this sense, is a partial map such that

$$\|x\|_e + |x|_f + \|f(x)\|_{e'} \leq P(|x|_f) \qquad (P \in \mathcal{C})$$

(recall we must have $efe' =_{\mathcal{C}} f$). We may think of this as requiring that the timing of a map cannot be "faster" than what is required to read the input and produce the output! Stated like this it seems to be a reasonable requirement, however, it does mean that even "doing nothing", in the sense of just passing on the input unchanged involves actually reading the input and writing an output.

In particular, note that a map $f : e \to e'$ is total in case $\overline{f} = e$ which means in terms of timing that

$$|x|_f \leq P(\|x\|_e) \qquad (P \in \mathcal{C})\,.$$

If this condition holds, we say that f has time complexity in $\mathcal{C}$. (The other bounding identity holds already from the requirement above.) To have time complexity in $\mathcal{C}$ means that the timing of f is $\mathcal{C}$-bounded by the size of its input: so, intuitively, f "runs" in time $P(n)$ where $P \in \mathcal{C}$ and $n \in M$ is the size of the input.

Summarizing this discussion we have:

Proposition 5.1 *When $\mathcal{C}$ is additive $\mathsf{Split}(\mathsf{TSet}(M)/\mathcal{C})$ is a discrete distributive restriction category in which the total maps are precisely the maps whose time complexity lies in $\mathcal{C}$. Furthermore, if $\mathsf{TSet}(M)/\mathcal{C}$ has joins, ranges, or iteration then so does $\mathsf{Split}(\mathsf{TSet}(M)/\mathcal{C})$.*

There is a technical move we must make at this stage to avoid elements with zero size – which are undesirable in complexity calculations. Every object in $\mathsf{Split}(\mathsf{TSet}(M)/\mathcal{C})$ has a unique total map to the terminal object $! : X \to 1; x \mapsto ()$ and $|x|_! = \|x\|$. Now there is a subobject of the object $1 = \{()\}$ given by splitting the total idempotent $\star : 1 \to 1$ where $|()|_\star = 1$. By slicing the category over the object $\star$, $\mathsf{Split}(\mathsf{TSet}(M)/\mathcal{C})/\star$ (note here the objects are *total* maps $\hbar_X : X \to e$ with maps $f : X \to Y$ with $f\hbar_Y \leq \hbar_X$) we make $\star$ the restriction final object.

We shall say that $\mathcal{C}$ is a **pointed** complexity order if $P(0) = 0$ for all $P \in \mathcal{C}$. Note that $\mathcal{L}$ and $\mathcal{P}^*$ are pointed complexity orders. We observe:

Lemma 5.2 *If $\mathcal{C}$ is a pointed and additive complexity order then an object $Y \in \mathsf{Split}(\mathsf{TSet}(M)/\mathcal{C})$ has a total map to $\star$ (it must be unique) if and only if each element of Y has a non-zero size.*

From now on we shall avoid zero size elements by working in $\mathsf{Split}(\mathsf{TSet}(M)/\mathcal{C})/\star$ for a pointed complexity order $\mathcal{C}$.

6 Computability in Timed Sets

The total maps in $\mathsf{PTSet} := \mathsf{Split}(\mathsf{TSet}(\mathbb{N})/\mathcal{P}^*)/\star$ are by no means the standard PTIME maps of complexity theory as there was no requirement that these maps be realizable by a computation. We have only managed, by construction, to arrange that the "timings" – which are quite arbitrarily given – of all total maps must be polynomially bounded on their input sizes. To obtain, for example, the standard notion of PTIME we must restrict to those timed maps which can be realized by a Turing machine (or some variant thereof – possibly with input and output tapes) in a number of steps which is $\mathcal{P}$-equivalent to the given timing.

Our objective, however, is not merely to cut out the PTIME maps from PTSet but to show how they can be seen to sit in a model of computability in which the total maps are the PTIME maps. More precisely we shall show that the "computable" maps in PTSet form a Turing category in which the total maps are exactly the PTIME maps.

A Turing category may be described as a Cartesian restriction category with an special object T, called a **Turing object**, such that:

- Every object in the category is a retract of T.
- There is an **application map**, also called a **Turing morphism**, $\bullet : T \times T \to T$ such that for every (partial) map $f : T \times T \to T$ there is a *total map* $\widetilde{f} : A \to T$, called an **index** of f, such that:

$$\begin{tikzcd} T \times T \arrow[r, "\bullet"] & T \\ T \times T \arrow[u, "\widetilde{f}\times 1_T"] \arrow[ur, "f"'] & \end{tikzcd}$$

Turing categories provide a unifying formulation of abstract computability (see [2]), and when the category has joins and is discrete, one can obtain many standard results from computability theory.

We show how to obtain a Turing category in two stages: first, we shall view Turing machines (in the standard sense) as providing a "program object" which acts on lists of bits. Second, we shall describe when a program object makes its values a Turing object so that the programmable maps form a Turing category. Thus, our discussion starts by introducing how the set of Turing machines may be viewed as a "program object", P. Programs can be evaluated on values (inputs), A, to produce new values (outputs), thus, associated with a program object is an evaluation (partial) map $\mathsf{ev} : P \times A \to A$: we then refer to P as being an A-program object. We say a map is "P-programmable" if it is of the form $f : A \to A; x \mapsto \mathsf{ev}(\lceil f \rceil, x)$, where $\lceil f \rceil : 1 \to P$ is an element of P, that is a program. Clearly we would like the programmable maps to form a Cartesian restriction category: to achieve this, however, requires more structure.

In order, for an A-program object to be able to describe maps from $A^n \to A^m$ it suffices to describe programmable maps $A^n \to A$ (and when $m = 0$ the total map $A^n \to 1$ must be programmable). To describe programmable maps $A^n \to A$ it suffices to be able to encode A^2 in A in a programmable way and this leads us to demanding that the values themselves have a special property.

In a Cartesian restriction category $\mathbb{X}$ an object A is a **powerful** in case there are total maps $s_\times : A \times A \to A$, $s_1 : 1 \to A$ and partial maps $P_0, P_1 : A \to A$ such that $s_\times \langle P_0, P_1 \rangle = 1_{A\times A}$ (and $s_1 !_A = 1_1$). The terminal object 1 is always a trivial example of a powerful object. A more interesting example, which we shall use below, is $\mathsf{List}(\mathsf{Bool})$ (lists of Booleans) in LTSet, with size given by $\|x\| = 2 \cdot (1 + \mathsf{len}(x))$ (i.e., twice the length of the list plus one). The map $s_1 : 1 \to \mathsf{List}(\mathsf{Bool})$ picks out the empty list. There are then *linear time* maps $s_\times$, P_0, and P_1 which code and decode pairs:

$$\begin{aligned}
&s_\times(b : bs, b' : bs') = 1 : b : 1 : b' : s_\times(bs, bs') && P_0(1 : b : \_ : \_ : rs) = b : P_0(rs)\\
&s_\times([], b' : bs') = 0 : 0 : 1 : b' : s_\times([], bs') && P_0(0 : 0 : \_ : \_ : rs) = []\\
&s_\times(b : bs, []) = 1 : b : 0 : 0 : s_\times(bs, []) && P_1(\_ : \_ : 1 : b' : rs) = b' : P_1(rs)\\
& && P_1(\_ : \_ : 0 : 0 : rs) = []
\end{aligned}$$

Given a powerful object A, an A-**program object** is an object P which has total operations $\mathsf{comp}, \mathsf{pair} : P \times P \to P$ together with total points $\mathsf{I}_0, \mathsf{I}_1, \mathsf{I}, \mathsf{J} : 1 \to P$ and a partial **evaluation** map $\mathsf{ev} : P \times A \to A$ such that:

$$\begin{tikzcd}
A \arrow[r, "\langle \mathsf{I},1\rangle"] \arrow[dr, equal] & P \times A \arrow[d, "\mathsf{ev}"] \\
& A
\end{tikzcd}
\qquad
\begin{tikzcd}
P \times P \times A \arrow[r, "\mathsf{comp}\times 1"] \arrow[d, "1\times\mathsf{ev}"'] & P \times A \arrow[d, "\mathsf{ev}"] \\
P \times A \arrow[r, "\mathsf{ev}"'] & A
\end{tikzcd}
\qquad
\begin{tikzcd}
P \times P \times A \arrow[r, "\mathsf{pair}\times 1"] \arrow[d, "{\langle \pi_0,\pi_2,\pi_1,\pi_2\rangle}"'] & P \times A \arrow[dd, "\mathsf{ev}"] \\
P \times A \times P \times A \arrow[d, "\mathsf{ev}\times\mathsf{ev}"'] & \\
A \times A \arrow[r, "s_\times"'] & A
\end{tikzcd}$$

$$\begin{tikzcd}
A \arrow[d, "\langle \mathsf{I}_i, 1_A\rangle"'] \arrow[dr, "P_i"] & i = 0,1 \\
P \times A \arrow[r, "\mathsf{ev}"'] & A
\end{tikzcd}
\qquad
\begin{tikzcd}
A \arrow[r, "\langle \mathsf{J},1\rangle"] \arrow[d, "!"'] & P \times A \arrow[d, "\mathsf{ev}"] \\
1 \arrow[r, "\mathsf{s}_1"'] & A
\end{tikzcd}$$

Program object requirements can be explained as follows. The first diagram states that there must be an identity program. The second says that one must be able to compose programs. The third says that there is an operation, pair, for combining two programs which you wish to apply to a single input in order to produce a pair of answers encoded. The last three diagrams require that there are programs for projecting from an encoded pair and to the terminal object. Clearly the codes of a Turing object include these required elements and can be combined in these ways: so *every* Turing object is automatically a program object. The converse, of course, is not true.

A map $f : A \to A$ is said to be P-**programmable** in case there is an element $\lceil f \rceil : 1 \to P$ such that

$$\begin{tikzcd}
P \times A \arrow[r, "\mathsf{ev}"] & A \\
A \arrow[u, "\langle \lceil f\rceil, 1_A\rangle"] \arrow[ur, "f"'] &
\end{tikzcd}$$

We shall indicate that X is retract of A by writing $X \triangleleft_s^r A$, where $sr = 1_X$ and

rs is an idempotent of A. A retract of A, $X \lhd^r_s A$, is a P**-programmable retract** in case the idempotent $rs : A \to A$ is P-programmable. If $X \lhd^r_s A$ and $Y \lhd^v_u A$ are P-programmable retracts of A then a map $h : X \to Y$ is P**-programmable** if the map $A \to^r X \to^h Y \to^u A$ is P-programmable.

Note that the conditions of P being an A-program object ensure that 1_A is computable and if f and g are computable fg is computable: $\langle\langle\lceil g\rceil, \lceil f\rceil\rangle\,\mathsf{comp}, 1_A\rangle\,\mathsf{ev} = \langle\lceil g\rceil, \langle\lceil f\rceil, 1_A\rangle\,\mathsf{ev}\rangle\,\mathsf{ev} \;=\; f\,\langle\lceil g\rceil, 1_A\rangle\,\mathsf{ev} \;=\; fg$ Furthermore, these conditions ensure that all powers A^n are P-programmable retracts of A (including when $n = 0$) and, thus, we may form a category, $\mathsf{Prog}_{P/A}(\mathbb{X})$, of P-programmable maps in $\mathbb{X}$ whose objects are just the powers of A. In fact, it is not hard to prove:

Proposition 6.1 *If A is a powerful object and P is an A-program object in a Cartesian restriction category then the subcategory of P-programmable maps, $\mathsf{Prog}_{P/A}(\mathbb{X})$, forms a Cartesian restriction subcategory.*

One thing which is perhaps less obvious concerns how the restriction is defined: one uses the fact that $\overline{f} = \langle 1, f\rangle\,\pi_0$: if f is programmable then $\overline{f} = \langle 1, f\rangle\,\pi_0 = \langle\langle\langle \mathsf{I}, \lceil f\rceil\rangle\,\mathsf{pair}, \mathsf{I}_0\rangle\,\mathsf{comp}, 1_A\rangle\,\mathsf{ev}$.

Finally, in order to obtain a model of computability, we wish to understand when an A-program P object turns A into a Turing object in the subcategory of P-programmable maps. The following is the main formal observation of this section:

Proposition 6.2 *If $\mathbb{X}$ is a Cartesian restriction category with a powerful object A and an A-program object P such that P is a P-programmable retract of A and* comp, pair, ev, I_0, I_1, *and* I *are all P-programmable then* $\mathsf{Prog}_{P/A}(\mathbb{X})$ *is a Turing category.*

Proof. Define the program $Q := \lceil\langle P_0, P_1\rangle\, f\rceil$ then

$$\begin{tikzcd}
A \times A \arrow[r, "\langle P_0,P_1\rangle\times 1"] & A \times A \times A \arrow[dl, no head, double] \arrow[d, "1\times s_\times"] \arrow[dr, "r_P\times s_\times"] & & \\
A \times A \times A \arrow[u, "s_\times\times 1"] \arrow[r, "1\times s_\times"'] & A \times A \arrow[r, "r_p\times 1"] & P \times A \arrow[dr, "\mathsf{ev}"] & \\
 & P \times A \arrow[u, "s_P\times 1"] \arrow[ur, no head, double] \arrow[rr, "\mathsf{ev}"] & & A \\
A \times A \arrow[uu, "(Qs_P)\times 1\times 1"] \arrow[r, "s_\times"'] & A \arrow[u, "Q\times 1"] \arrow[r, "\langle P_0,P_1\rangle"'] & A \times A \arrow[ur, "f"'] &
\end{tikzcd}$$

where $(Q \times 1)s_P s_\times$ is the required total map and the Turing morphism is $\bullet := (\langle P_0, P_1\rangle \times 1)(r_P \times s_\times)\mathsf{ev}$. $\square$

Notice that the proof uses all the structure of a programming object and the powerfulness of A. In particular, the definitions of the Turing morphism and of the index use pairing and composition non-trivially.

Our aim is to apply this to the set of Turing machines regarded as $\mathsf{List}(\mathsf{Bool})$-program object in PTSet with the usual timing of evaluation. First note that regarding the program object, P, as the set of specifications of Turing machines certainly means that P may be viewed a programmable retract of $A = \mathsf{List}(\mathsf{Bool})$. Some bit strings may not be legal specifications of a Turing machine but recognizing the legal specifications can certainly be achieved in polynomial time – indeed in linear time as the legal specification can be given by a regular language. The composition and pairing function must take in two specifications of Turing machines and modify them to produce, respectively, the composite or the interleaved pairing. Composition is clearly programmable map on pairs of programs: it basically involves identifying the starting state of the second machine with the final state of the first. Thus, it can be done in linear time. Somewhat trickier is to see that pairing is programmable: the trick is to view pairing as a composite of duplicating the input (onto "odd" and "even" positions on the tape) modifying the first program to act on only "even" bits and composing it with the second program modified to only act on "odd" bits. This can be done in linear time and is certainly programmable and total.

The only remaining difficulty is to show that the map ev is programmable. Recall that the way we defined evaluation was to run the specified Turing machine on the input: now we are asking that there be a universal Turing machine which can interpret any Turing machine specification in a manner so as to make the two maps equivalent with respect to the complexity order (here $\mathcal{P}^*$). Fortunately, that any Turing machine can be simulated by a universal Turing machine with only a loss of an $O(\log n)$ factor in performance is well-known (see [1] Theorem 1.9 or [12] in the proof of the "Time Hierarchy Theorem" 9.10). This tells us that evaluation is programmable.

This discussion shows that the conditions of Proposition 6.2 are satisfied and yields:

Corollary 6.3 *The maps which are programmable by a Turing machine in polynomial time in* PTSet *form a Turing category,* $\mathbb{T}_{\mathsf{ptime}}$*, whose total maps are precisely the PTIME maps.*

As the best known simulation of a Turing machine by a universal Turing machine has an order $\log n$ overhead, one cannot quite make the argument (as presented) work for linear time computations. However, for any additive complexity order which contains $n \log n$ the argument works verbatim. Hence, this provides a number of examples of Turing categories whose total maps have low time complexity.

We have in these arguments relied on having *a priori* a grasp of how a Turing machine is timed. However, it is worth remarking that this too can be explained within this formalism using the trace. A Turing machine has the property that its evaluation map is given by iterating a *total* transition map, that is $\mathsf{ev} = \mathsf{step}^\dagger\mathsf{halt}$ where $\overline{\mathsf{step}} \vee \overline{\mathsf{halt}} = 1_{P\times A}$. The existence of a total transition map is a key ingredient of being a (deterministic) machine. By modeling the machine steps in PTSet one adds timing. Notice, however, that to arrange that each step has unit cost we are essentially forced to assume that the input has unit cost. This makes $\mathsf{step}^\dagger\mathsf{halt}$ exceedingly partial. This means these maps do not live in $\mathbb{T}_{\mathsf{ptime}}$. On the other hand, $\mathbb{T}_{\mathsf{ptime}}$ does inherit the trace of PTSet and, in fact, the evaluation map – the Turing morphism we consider above – is given by such an iteration (up to the equivalence determined by $\mathcal{P}^*$). However, now a single step must have a timing that dominates the cost of "reading" its input (i.e. the size of the tape), so this internal machine, while present, does not reflect well the usual step-counting intuition for measuring time complexity. Thus, it seems that the usual intuition is better served by using the iteration in PTSet with unit timing (as above) together with input and output functions that restore the usual sizing of the input and output. Specifically, taking $\mathsf{step}\dagger\mathsf{halt}$ to be the iteration that counts steps, and e_{in},e_{out} to be the restriction idempotents whose timing measures the size of lists of bits correctly, the evalution map we actually require in PTSet is

$$\mathsf{List}(\mathsf{Bool})^2 \xrightarrow{e_{\mathsf{in}};\mathsf{step}\dagger\mathsf{halt};e_{\mathsf{out}}} \mathsf{List}(\mathsf{Bool})\,.$$

The arguments above all rely, mimicking the approach taken in complexity theory, heavily on the machine model and the way resources are measured. Changing the machine model, of course, changes what can be computed and also the overhead of simulation. To illustrate this we briefly discuss LOGSPACE computations. Recall that a **transducer** is a Turing machine with a read only input tape (on which one can go backward and forward), a read/write working tape, and a write only output tape (on which one can only write and move right): the maximum length that the work tape attains during a computation is the space resource which is measured – this is abstractly now to be the "timing" of the map. The manner of composing transducers is crucial: this, in particular, determines how we set the "timing" of the identity on a list of bits. A transducer thinks of its input as being generated "by need": if it requires the n^{th} bit it simply runs the transducer which generates its input, throwing away all the bits it generates, until the required bit is produced. It then proceeds with its calculation having obtained the required bit. This is extremely time inefficient but it is space efficient. Minimally to do this one must be able to actually count the bits one must throw away and, for this, one needs roughly $O(\log n)$ space. Therefore, the cost of the "reading" the input (the idempotent) on a list of bits is set to be the logarithm (base 2) of the length of the list (rounded away from zero). Furthermore, somewhat surprisingly, the "timing" of composition is given by addition under the linear complexity order, $\mathcal{L}$. Thus, the category of LOGSPACE computations is carved out from $\mathsf{LTSet} := \mathsf{Split}(\mathsf{TSet}(\mathbb{N})/\mathcal{L})/\star$. Furthermore, the program object we use this time is the set of transducer specifications on lists of bits.

Again we wish to use Proposition 6.2 to show this category is a Turing category. For this we need to check the requirements. Clearly legal transducer specifications, represented as list of bits, can be recognized using a log space computation so P is a programmable retract of A. Clearly, also, A is still a powerful object with respect to the same maps as above which are all clearly log space. It is less obvious how composition and pairing of transducers can be performed in log space. Composition is implemented by the second program calling on the first program whenever it moves the input head, the first program provides the input required by running itself until the required output is generated. This is achieved by modifying the second program's read instructions to call the first program and to record a position and can be achieved in log space. Once one has composition the pairing is straightforward using the technique described above: one expresses it as a composite of duplicating the input, modifying the first program to work on "odd" bits and the second to work on "even" bits. Clearly, these modifications can all be managed in LOGSPACE. For the program evaluation map itself we recall that it is well-known that a universal Turing machine (or transducer) can simulate a specified transducer with only a constant factor of degradation in space efficiency (see exercises in [12,1]).

We therefore have:

Corollary 6.4 *The maps which are programmable by transducers in logarithmic time in* LTSet *form a Turing category,* $\mathbb{T}_{\mathsf{logspace}}$*, whose total maps are precisely the LOGSPACE maps.*

It is also worth remarking that there is a functor $V : \mathbb{T}_{\mathsf{logspace}} \to \mathbb{T}_{\mathsf{ptime}}$ which takes a LOGSPACE computation to a PTIME computation using the fact that a computation in space s can be performed in time $2^{O(s)}$(using [1] Theorem 4.2 for example): the only subtlety is that one must actually "slow down" space computations which do not use the full exponential time so that equivalent programs are taken to equivalent programs. Clearly, this functor preserves the meaning in sets and partial maps:

$$\begin{tikzcd}
\mathbb{T}_{\mathsf{logspace}} \arrow[rr, "V"] \arrow[dr] & & \mathbb{T}_{\mathsf{ptime}} \arrow[dl] \\
& \mathsf{Par} &
\end{tikzcd}$$

This shows how relationships between functional complexity classes give rise to functorial relationships between their Turing categories. Whether there exists an *isomorphism* between $\mathbb{T}_{\mathsf{logspace}}$ and $\mathbb{T}_{\mathsf{ptime}}$ is, of course, an intriguing question equivalent [15] to the open problem whether PTIME = LOGSPACE.

7 Conclusion

The objective of this work was to provide concrete models of computability, as embodied in Turing categories, in which the total maps belong to a specific functional complexity class (such as PTIME and LOGSPACE). The constructions we have provided do achieve this. Furthermore, they closely mimic the standard approach taken in complexity theory. This suggests that the results of complexity theory may be mapped fairly directly into categorical facts.

[15] It is immediately clear that if this functor is an isomorphism then PTIME = LOGSPACE. The converse not obvious but, nonetheless, is true. From the assumption that PTIME = LOGSPACE, one can prove that this V must be an isomorphism for *all* "computable" functions. Here is a sketch of the proof: the idea is to view an arbitrary computation, f, as a function which has a time bound provided by an additional argument, so that $f(x) = a \Leftrightarrow \exists t. f'(x,t) = a + 1$. The computation $f'(x,t)$ is total: one interprets $f'(x,t) = 0$ as meaning that the computation had not completed by time t. One can further assume that, when it has completed, so that $f'(x,t) = a+1$, then for every $t' \geq t$ one has $f'(x,t') = a+1$. In fact, f' is not only total and runs in polynomial time but, when t is represented in *unary*, it can actually be arranged to run in linear time. This means, by assumption, the program f' can be transformed into a LOGSPACE program. However one can then iteratively search over the time, t, using the program f' to show that $f(x)$ can be implemented in LOGSPACE.

References

[1] S. Aurora and B. Barak, *Computational Complexity: a modern approach.* Cambridge University Press (2009).

[2] J.R.B Cockett and P. Hofstra, *Introduction to Turing Categories.* Annals of Pure and Applied Logic, Vol. 156 (2008) 183–209.

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

[4] J.R.B. Cockett and S. Lack, *Restriction categories III: colimits, partial limits and extensivity.* Mathematical Structures in Computer Science, Vol 17- 4 (2007) 775-817.

[5] J.R.B. Cockett and E. Manes, *Boolean and Classical Restriction Categories.* Mathematical Structures in Computer Science, Vol 19-2 (2009) 357-416.

[6] J. R. B. Cockett and B. F. Redmond *A Categorical Setting for Lower Complexity.* Electronic Notes in Theoretical Computer Science, Vol. 265 (2010) 277 300.

[7] D. Gurr, *Semantic frameworks for complexity.* PhD Thesis, University of Edinburgh (1991).

[8] M. Hasegawa, *On traced monoidal closed categories.* Mathematical Structures in Computer Science, Vol 19-2 (2009):217-244.

[9] A. Joyal, R. Street, and D. Verity, *Traced monoidal categories.* Math. Proc. Cambridge Philos. Soc. Vol. 119-3 (1996) 447468.

[10] R.A. Di Paola and A. Heller, *Dominical categories: recursion theory without elements.* Journal of Symbolic Logic, Vol. 52 (1987), 595-635.

[11] E. Robinson and G. Rosolini *Categories of partial maps.* Information and Computation, vol. 79 (1988) 94–130.

[12] M. Sipser *Introduction to the theory of computation.* PWS Publishing Company (1996)