Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science 324 (2016) 91–106

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

Defining Effectiveness Using Finite Sets A Study on Computability

Hugo D. Macedo [a,b,1]

[a] *DTU Compute*
Technical University of Denmark
2800 Kongens Lyngby
DENMARK

Edward H. Haeusler [2]

[b] *Departamento de Informática – PUC-Rio*
Rua Marquês de São Vicente 225
22451-900 Gávea
Rio de Janeiro
BRAZIL

Alex Garcia [3]

Instituto Militar de Engenharia – IME
Praça General Tibúrcio 80
22290-270 Urca
Rio de Janeiro
BRAZIL

Abstract

This paper studies effectiveness in the domain of computability. In the context of model-theoretical approaches to effectiveness, where a function is considered effective if there is a model containing a representation of such function, our definition relies on a model provided by functions between finite sets and uses category theory as its mathematical foundations. The model relies on the fact that every function between finite sets is computable, and that the finite composition of such functions is also computable. Our approach is an alternative to the traditional model-theoretical based works which rely on (**ZFC**) set theory as a mathematical foundation, and our approach is also novel when compared to the already existing works using category theory to approach computability results. Moreover, we show how to encode Turing machine computations in the model, thus concluding the model expresses at least the desired computational behavior. We also provide details on what instances of the model would indeed be computable by a Turing machine.

Keywords: Effectiveness, Finite sets, Models of computation.

[1] Email: hmacedo@inf.puc-rio.br
[2] Email: hermann@inf.puc-rio.br
[3] Email: garcia@ime.eb.br

http://dx.doi.org/10.1016/j.entcs.2016.09.009

1 Introduction

Together with the arrival of advanced computing machinery, the first half of the 20th century presented us with the concept of computability. Nowadays, a standard course on the subject should discuss models and results on computation and in particular the Church-Turing thesis, which equates what can be effectively computed, and what can be computed by a machine running an algorithm. In what follows, the term algorithm is to be taken according to its current meaning. Effectiveness, on the other hand, will be defined building upon its intuitive meaning of what "is computable".

Despite its apogee in the 20th century, effectiveness was already present in many fields of the human knowledge in Ancient times. In our daily lives there are routines we perform due to cultural heritage, while others are pervasive and encountered in every culture. Such pervasive routines may be considered as universal or natural aspects of our lives. Arithmetic is an example of such an universal aspect, it is present in all known civilizations [4], and (historically) a great part of our cognitive life (was and) is dedicated to learn and develop algorithms on the basic operations used to calculate with natural numbers and other number systems.

Besides arithmetic, the field of geometry has its realm of algorithms too. For example, proposition I.1, in Book I of *Euclid's Elements*, showing that given a line segment, an equilateral triangle exists that includes the segment as one of its sides, is obtained by describing an algorithm that constructs an equilateral triangle on a (given) straight line, together with its own correctness proof. Many propositions in *Euclid's Elements* are proved in this way, i.e., the description of an algorithm and its correctness companion argument.

Classical problems as *squaring the circle* or *trisecting an angle* using a finite number of straightedge and compass operations can be understood as the search of a method to solve a problem by means of a specialized algorithm. Both problems were proved impossible to solve and it seems that only in mathematics such impossibility proofs appear in a definite form. Posterior investigations have shown that constructions with straightedge and compass together with the usual algorithms on the four basic arithmetical operations are not sufficient as the representatives of effectiveness in such problems.

Archimedes' method of exhaustion provided a way to escape from these impossibility proofs sacrificing perfection, namely exact results, in favor of feasibility (see the quadrature of the parabolic segment [1]). Thus, since the ancient classical period, it is known that the nature of the objects involved in the chosen method to attack a problem determines the effectiveness of the respective method itself. Nowadays, such rule-of-thumb finds its place in the manual of best-practices of computer programmers when deciding and designing the datatypes that the program manipulates.

It took a pair of millenniums for us to realize that effectiveness should be firstly considered as a kind of property on functions, and, that functions should be consid-

[4] Furthermore, also primates, lions, and dolphins have arithmetical capabilities [2]

ered as a relationship between inputs and outputs. The last century was successful in characterizing which functions from a set of inputs A into a set of outputs B are effective. A pair of decades after its beginning, partial functions instead of total ones were admitted in the realm of effective functions. Furthermore, realizing that the set of Natural Numbers, presented as a numeral system, should be considered as the universe for the subsets A and B mentioned previously, was also a recent development.

Historically, investigations on effectiveness followed mainly two non-exclusive tracks. In the model-theoretical approach, a "model" $\mathcal{M}$ is presented and any (partial) function $f : \mathbb{N} \to \mathbb{N}$ is called $\mathcal{M}$-effective if and only if there is an instance of the model $\mathcal{M}$, written as X_f, that represents f. The meaning of the "instance $\mathcal{M}_f$ represents f" is given informally by stating that whenever an input i submitted to $f_{\mathcal{M}}$ produces an output o then $f(i) = o$ must be true. The meaning of "submitting" and of "producing" is also, at least informally, defined when introducing the "model" $\mathcal{M}$.

In the proof-theoretical approach, a logical theory $\mathcal{T}$ is defined and f, a function from $A \subseteq \mathbb{N}$ into $B \subseteq \mathbb{B}$, is said to be $\mathcal{T}$-effective if and only if $f \in \mathcal{T}$. Of course $\mathcal{T}$ is presented by a set of axioms and inference rules for deriving propositions on membership to the theory $\mathcal{T}$ and on equivalences between functions f and g, i.e., of the form $g \in \mathcal{T}$ and $f \equiv g$ respectively.

A typical example of a model-theoretic approach is the Turing machine based effectiveness definition, whereas Gödel partial recursive functions exemplifies the proof-theoretical one. The approaches for defining effectiveness are not exclusively model-theoretical or proof-theoretical. One can consider lambda-calculus as a purely proof-theoretical example if one ignores the underlying evaluation model provided by the identity relationship. On the other hand, we can consider the lambda calculus as a model-theoretical approach if one focus on the λ-terms evaluation model. Roger's theorem on the abstract axiomatization for the proof-theoretical approach on effectiveness provides stronger [5] evidence for Church-Turing thesis (see [12] and mainly [10] for a very appraisal discussion). Roger axiomatization is a proof-theoretical attempt to precisely express the models for effectiveness in a truly abstract way. Programs are numbers and models are families of Natural Number valued functions indexed by the formers.

To the best of our knowledge, beyond its similarities it is worth mentioning that almost all the model-theoretical approaches for effectiveness lie inside **ZFC** [6]. Category Theory (**CT**), although an alternative that is not completely dissociated from the **ZFC** and other set theoretical approaches to the foundation of mathematics, provides, nevertheless, an alternative ontology [7]. In **CT**, classes of objects and morphisms form a category. Morphisms are typed by domain and co-domain. For example let A and B be objects in a category $\mathcal{C}$ with $f : A \to B$ a morphism in $\mathcal{C}$, f has domain A and co-domain B.

[5] Stronger than evidence provided by some concrete models, as those raised since Turing's work

[6] Zermelo Fraenkel set theory with the axiom of Choice

[7] Terminology, in philosophical sense

However, the meta-theory **CT**, apart from the parcel of **ZFC** that it uses, does not provide meaning to propositions of the form $A = B$ in $\mathcal{C}$. Such theory only provides meaning to assertions of the type $f = g$ whenever f and g are morphisms. Given that, one concludes that in **SETS**, the archetypal category of the class of all sets and all functions between them, only the identity on functions have a precise semantics [8]. Moreover, about the objects of **SETS**, i.e., the sets themselves, it cannot be stated that any two sets are equal or not equal. The most that can be said is that they are isomorphic or not [9]. This change of perspective is quite interesting since it provides more ways to compare models of certain concepts than if the concepts were formalized on top of **ZFC**.

This article follows the model-theoretical approach for defining effectiveness by providing an alternative way to present effective functions using category theory. Our proposal is different from the effective topos ([7]) and from the work presented in a series of articles by prof. Robert Walters (see [16] for a brief and easy introduction on the subject). In the next section we discuss the main motivation of our alternative to the study of effectiveness.

2 Effectiveness and their categorical models

Higher-order logic (**HOL**) has been generally used to express and formalize concepts in computer science ([9,14,15]). As a typed language, it is well suited to the ubiquitous typing discipline in formal specification and validation. Contrary to **ZFC** specifications that are usually untyped and use first-order logic, this typing discipline has deeper consequences than the simple fact of avoiding paradoxes. Because of lack of space, we cannot discuss this here. The reader can see [5,8] for a comprehensive presentation. From the model-theoretic counterpart, higher-order logic theories have special categories, called *Toposes*, that serve as abstract models for these theories, the same way Heyting algebras serve as algebraic models for Intuitionistic Propositional Logic.

Although having a simple definition, a topos describes an entire mathematical universe of discourse for **HOL** theories. Any provable **HOL** formula is true in every topos, and consequently in **SETS**. On the other hand, any topos has an internal logic, such that, if a **HOL** formula is true in this topos, the formula is provable in intuitionistic logic. The category **SETS** is a topos that has Classical **HOL** as internal language, since every valid classical **HOL** formula is true in **SETS**. On the other hand, the category of functors from the pre-ordered category $\omega = \{0, 1, 2, n, \ldots\}$ into **SETS** validates only the intuitionistically valid **HOL** formulas. In this article the sub-category of finitely valued functors from ω into **SETS** is central. This category is denoted by $\mathbf{FinSet}^{\omega}$.

What is the appropriate universe of discourse for effectiveness? Technically speaking, we would like to know what is the adequate topos for studying effectiveness. Our motivation for studying $\mathbf{FinSet}^{\omega}$ comes from concluding that there is no

[8] Equalizing functions is a known issue

[9] A is isomorphic to B, iff, there are $f : A \to B$ and $g : B \to A$, such that, $f \circ g = Id_B$ and $g \circ f = Id_A$

topos that satisfies the law of the excluded middle, has only computable functions as morphisms, and has a Natural Numbers object. Thus, the universe of discourse for effectiveness has to dropout at least one of these three properties, but let us first examine such result.

The argument that follows can be found in [13] and specifically in [11] using the language of **CT**. Firstly, take the *Strong Church Thesis* (**SCT**) "Every function from $\mathbb{N}$ in $\mathbb{N}$ is computable" into account. Considering that a function is computable if and only if there is a program that computes it, and, any program can be expressed by its code that by its turn can be viewed as a natural number. Thus, **STC** can be expressed by the following first-order formula:

$$\forall f\, \exists p\, \forall n\, \exists y \cdot (T(p,n,y) \wedge Out(y) = f(n)),$$

where $T(p,n,y)$ is Kleene's T predicate and $Out(y)$ is Kleene's output function. Using the theory of Peano Arithmetic we can obtain T and Out as primitive recursive predicate and function respectively. Thus, in any topos with a Natural Number object, T and Out are primitive recursive too. Thus, in the typed **HOL** language, **SCT** is of form:

$$\forall f^{\mathbb{N}^{\mathbb{N}}} \exists p^{\mathbb{N}} \forall n^{\mathbb{N}} \exists y^{\mathbb{N}} (T(p,n,y) \wedge Out(y) =_{\mathbb{N}} f(n)).$$

Considering an arbitrary topos having a Natural Numbers object, and having classical logic as internal logic, it can be shown that **SCT** is inconsistent with this situation, namely, in this topos **SCT** cannot hold. Using the fact that classical logic satisfies the law of excluded middle, the definition for g as follows:

$$g(n) = \begin{cases} m+1 \text{ if } & \exists j^{\mathbb{N}} (T(n,n,j) \wedge Out(j) = m) \\ 0 & \text{otherwise} \end{cases}$$

is provable to be defined for every n. Thus g is a total function, and hence by **SCT** has a program p. However, any program $p \in \mathbb{N}$ that implements g is such that $g(p) = (g(p)) + 1$, since there is j, such that, $T(p,p,j)$ and $Out(j) = m$ and $(g(p)) = m + 1$. This is not possible, so **SCT** is inconsistent with the law of the excluded middle in a topos having a Natural Numbers object.

Let us analyze the remaining alternatives when defining a topos. A topos may have the following properties:

(i) The internal logic of the topos is classical;

(ii) Every morphism in the topos is effective, i.e., **SCT** holds in the topos;

(iii) The topos has a Natural Numbers object.

We've just shown that ii is inconsistent with i and iii, so either we drop out classical logic or the existence of Natural Numbers object. If we define a topos with a non-classical internal logic we end up revisiting the well studied effective topos. On the other hand, if we drop out item ii, we obtain with i and iii the also well studied classical theory of recursive functions. The last alternative is to drop out the existence of a Natural Numbers object. In a naive setting, i and ii together entail finite sets and first-order finite domain logic. In the category-theoretic setting, $\mathbf{FinSet}^{\omega}$ represents this alternative which we claim to deserve more study.

3 Particularities of finite sets (FinSet)

The first step in defining the model of computation is the clarification of what is a finite set. Such endeavour avoid misunderstandings in the following development and due to its intuitive character one usually steps over formalities losing insight. Finite set is defined in terms of finite which means limited and mathematically:

Definition 3.1 The empty set is finite. A non-empty set S is finite if there exists a bijection from S to a prefix of the natural numbers, $\{1, \cdots, n\} \subset \mathbb{N}$. The number n is usually termed the cardinality of the set.

Note that by relying on natural numbers to define finite sets we do not introduce a Natural Numbers Object (NNO) in our computability model. One could easily use a definition of finiteness using other alternatives as the ones studied in [6], nevertheless the usage of $\mathbb{N}$ simplifies our characterization of finite sets.

Definition 3.2 The category with finite sets as objects and all functions between such sets, denoted as **FinSet**, is the full subcategory of **SETS** where objects are restricted to finite sets.

Our definition of **FinSet** sieves the category of all functions between sets and drops every function that has an infinite set as domain or range, the following proposition shows that the functions remaining in **FinSet** are finite.

Proposition 3.3 *A function $f : A \to B$ with a finite domain, i.e. A is a finite set, is finite, i.e. the cardinality of the set $\{(a, f(a)) \mid a \in A\}$ is finite.*

Proof. Let f be a function with type $f : A \to B$ where A is a finite set. As f is a function, for each $a \in A$ there must be only and only one element $b \in B$ such that $b = f(a)$, therefore there are at most as many elements $f(a)$ as elements in a. As the number of elements in A is finite, so are the number of elements $\{(a, f(a)) \mid a \in A\}$. □

As a corollary, every function in **FinSet** is finite. Let us now study the cardinality of the *hom*-sets in **FinSet**, of functions between finite sets.

Proposition 3.4 *Given finite sets $A = \{a_1, \cdots, a_n\}$ and $B = \{b_1, \cdots, b_m\}$ the function space B^A is a finite set with m^n elements.*

We are now in a position where we can prove the following:

Proposition 3.5 *For finite sets A and B every function $f : A \to B$ is computable.*

Proof. Suppose A and B are finite sets, thus by (3.4) the space of functions B^A is finite. Therefore let us pick a function $f : A \to B$. Let us show it is computable. Because f is a function for each $a \in A$ there exists a unique $b \in B$ s.t. $f(a) = b$. As A is finite by (3.1) it has some cardinality n and there is a bijection which we can use to assert $A = \{a_1, \cdots, a_n\}$. Now one can apply f to each element of A obtaining

```
input x;
output y;
parallel
  : if x <> r1 then t2;
  : if x <> r2 then t2;
  : if x <> r3 then t1;
end
```

Fig. 1. Program computing K

the set $\{f(a_1), \cdots, f(a_n)\}$, the image of f usually written as $f(A)$. Thus we devise a program P computing f as a simple table lookup for the input x. □

As there is a pattern matching inherent to the table lookup argument in the previous proof, one needs to study the cases where the elements pose problems to the equality decision, for instance infinite sets, or undistinguishable sets.

Pattern matching finite sets with undistinguishable elements.

Let us start with possibly undistinguishable elements. When defining a function of a finite set A with two elements, we assume the elements are distinguishable. Such argument can be derived from the fact that isomorphic objects are considered equal, thus as every finite set A with n elements can be put in bijection with the set $\{1, \cdots, n\}$. In practice, we abstract away the nature of the elements and work with the natural number labels. In the pattern matching process elements e and e' that are not distinguishable outside **FinSet** will not be distinguishable inside as well. In any case, given the fact that sets have no repeated elements, that is if $a \in A$ then $\{a\} \cup A = A$, the set built with two indistinguishable elements $\{e, e'\}$ cannot be a two element set, as it would demand a bijection $f : \{e\} \to \{0, 1\}$, but the cardinalities differ.

Pattern matching finite sets containing infinite elements.

Proposition 3.5 holds even if the elements of A and B belong to a recursive enumerable set that is not recursive. Let's illustrate such case, we know that there is no recursive identity relationship in $\mathbb{R}$. We also known that the difference in $\mathbb{R}$ is recursively enumerable. Let $A = \{r_1, r_2, r_3\} \subseteq \mathbb{R}$, $B = \{t_1, t_2\} \subseteq \mathbb{R}$ and $K : A \to B$, such that, $K(r_1) = K(r_2) = t_1$ and $K(r_3) = t_2$. We can assume that real numbers are represented by the processes/program that compute them. For example π can be represented by any program that given a precision p provides π expanded until this precision. Equivalently, we can consider the real numbers as programs that list the decimal expansion of them. In this last case, π is any program that prints 3.141593..... and never stops. Using, this last representation of (computable) real numbers, the function K is computed by the program in Figure 1. Where the command $parallel :< cmd_1 >; \ldots :< cmd_n >; end$ runs all $< cmd_i >$, $i = 1, n$, in parallel, and stops whenever some of $< cmd_i >$ stops. The output of the parallel command is the output of the $< cmd_i >$ that firstly stops. Programs like these, entail the fact that any function between finite sets is computable, even in the case

that the elements of the domain belong to a recursively enumerable data-type that is not recursive.

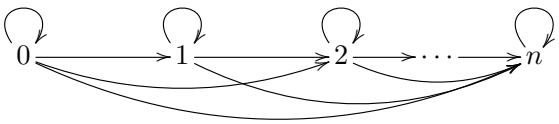
Now that we solved the issues arising from the usage of pattern matching we can proceed to prove how to build complex behaviours using finite functions as atomic and computable steps.

Proposition 3.6 *Let A, B, and C be finite sets and f and g be finite functions with types $f : A \to B$ and $g : B \to C$. The composition $g \cdot f$ is a finite and computable function.*

Proof. The composition $g \cdot f$ is a function, and has type $A \to C$. As A and C are finite sets, proposition 3.4 holds, meaning that $g \cdot f$ is member of the finite set of functions C^A, as a function it must map each element of A to an element of C, as there are finite elements in A, $g \cdot f$ maps at most finite elements, thus it is a finite function, thus by proposition 3.5 the composition $g \cdot f$ is computable. □

Thus, extending the previous argument we can prove that every morphism in **FinSet** is computable, thus every finite composition of finite functions is computable. To make such statement precise let us define finite composition of functions.

Definition 3.7 Let $n \in \mathbb{N}$ be a fixed number. From n we obtain the pre-ordered category with objects the numbers 1 to n, and all morphisms $a \to b$ when $a \leq b$, denoted as $\downarrow n$, and depicted as:



The category $\downarrow n$ is a finite linear pre-order .

Any functor from $\downarrow n$ into **FinSet** can is as a representation of finite compositions of finite functions.

Thus, by Proposition 3.6 and fixing n, the functorial category $\mathbf{FinSet}^{\downarrow n}$ is a category of computable functions. Despite such fact, the expressiveness of $\mathbf{FinSet}^{\downarrow n}$ is not enough to capture all computations, for instance the computation of π. To capture infinite behaviours we will study the case where $\downarrow n$ is substituted by ω, thus studying $\mathbf{FinSet}^{\omega}$ as model for computations. The limit just mentioned and the reason why ω allows the capture of infinite behaviors is made clear in the following section.

4 Expressing computations in $\mathbf{FinSet}^{\omega}$

In this section we define our base category $\mathbf{FinSet}^{\omega}$, we show that such category is at least as expressive as the universal Turing machine model of computation, and we observe that $\mathbf{FinSet}^{\omega}$ also includes non-computable behaviours. Such final observation leads to conjectures on what restriction should be applied to $\mathbf{FinSet}^{\omega}$ to restrict its expressive power to computable functions.

4.1 Encoding computations as functors in $\mathbf{FinSet}^{\omega}$

Our standard model of computation is the Turing machine which is an automaton with a non-empty finite set of states Q, a finite set of reading/writing symbols from an alphabet Γ, a distinguished blank symbol b, a partial transition function δ describing the behaviour of the machine, a distinguished state q_o which is corresponds to the initial state of the machine and $F \subseteq Q$ a set of final states.

Thus the tuple $\mathcal{M} = (Q, \Gamma, b, \Gamma \setminus b, \delta, q_0, F)$ corresponds to a Turing Machine, the universal model of computation. To complete the model of computation the machine $\mathcal{M}$ is usually coupled with a unbounded tape containing cells with symbols from Γ, more precisely at each instant of time the machine has access to a cell and is allowed to either move to the left(-1)/right(1) cell in the tape. Such movement is ruled by δ as recorded in its type:

$$\delta : (Q \setminus F) \times \Gamma \to Q \times \Gamma \times \{-1, 1\}$$

Turing machines stop when final (F) state is reached, thus the type of δ reads as "for each defined non-final state and symbol the machine state changes, a new symbol is written in the current cell and the machine positions itself to the right or the left of the current cell.

To each machine M and tape the behaviour of the machine is well-defined and is expressed in terms of sequences of configurations C_i where i a natural number tracking the number of transitions that occurred. Thus the behaviour of a machine $\mathcal{M}$ with a given tape is equivalent to a infinite sequence of configurations $C_0 \cdot C_1 \cdot C_2 \cdots$ which encodes the stepwise execution of the machine. Such a sequence is commonly termed a computation.

Thus, let us explain how to encode computation in our model. The configuration behaviour is easily encoded in the $\mathbf{FinSet}^{\omega}$. Before any formal explanation on why that is a fact, let the following drawing do the illustration of the infinite configuration sequence. In fact it provides a geometrical perspective on the just mentioned sequence.

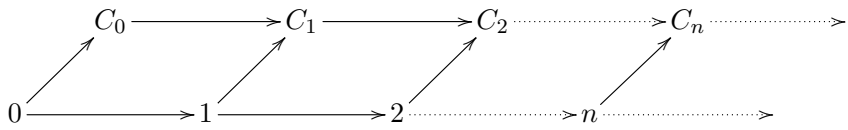


Fig. 2. Computation in $\mathbf{FinSet}^{\omega}$

Let us now make the configurations C_i concrete, providing thus a rigorous statement on the encoding using $\mathbf{FinSet}^{\omega}$. A configuration consists of a finite set containing the state $q \in Q$ in which the machine is, the current position (p) in the tape and a finite string $w \in \Sigma^*$ with the contents of the tape. Thus each configuration is a finite set indexed by a natural number, without further ado define $T \in \mathbf{FinSet}^{\omega}$

as [10]:

$$\begin{cases} T_0(i) & = C_i = (q, p, w) \\ T_1(i \to i+1) & = \lambda(q, p, w).(q', p + off, w') \\ T_1(i \to j) & = T_1((j-1) \to j) \cdot T_1(i, (j-1)) \end{cases} \tag{1}$$

where $(q', \gamma, off) = \delta(q, w_p)$ and w' is a copy of w except in position p the content is $w'_p = \gamma$

Therefore our definition of effectiveness can be used to prove one of the implications of the SCT thesis, Every Turing machine has a corresponding effective computation model. It corresponds to the unfolding of the dynamic system corresponding to the machine executions. Therefore, the interesting question is the contrapositive of the implication. What are the elements of $\mathbf{FinSet}^{\omega}$ that correspond to computations, i. e. the elements that could have been generated by a Turing machine?

Before answering that question let us highlight an interesting join point between our functorial expression of computations and the folk knowledge on computability that expresses that computable functions should necessarily satisfy the following condition: "to produce a finite amount of output only the inspection of a finite amount of the input is necessary". That statement can be observed in the contravariant functor definition of T_1 in (1). Note how it states that given a time span $(i \to j)$, to observe the outputed behaviour one needs only evaluate a finite amount (in fact $j - i + 1$ steps of applications of T_1.

4.2 Which subcategory of $\mathbf{FinSet}^{\omega}$ *corresponds to computations?*

Given that we have proved a result stating that functions between finite sets are computable, and that composition of such functions are computable as well, one would expect that given Figure 2 frames computations in terms of stepwise transitions between finite sets, given that the framework is $\mathbf{FinSet}^{\omega}$, one could conjecture that the functors of $\mathbf{FinSet}^{\omega}$ are indeed the computable entities. But that is not the case, the reason for that is that when substituting $\downarrow n$ by ω the argument of finite function composition being again a finite function is lost.

We will now present a counterexample that evidences that more structure must be present in the functor of $\mathbf{FinSet}^{\omega}$ for it to be computable. The argument is based in the following reasoning.

Assume for each i we attribute a finite set containing truth values $\top$ and $\bot$. We also assume an enumeration of Turing Machines. Then for each i attribute the finite function

$$f(i) = \begin{cases} \top \text{ if the } i^{th} \text{ machine halts} \\ \bot \text{ otherwise} \end{cases}$$

The function f is a member of $\mathbf{FinSet}^{\omega}$, but as it solves the halting problem, it

[10] We adopt the $T = (T_0, T_1)$ functor notation where T_0 is an object mapping, and T_1 a morphism mapping

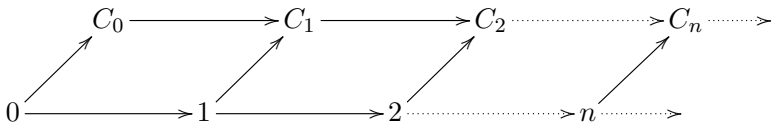
should not be computable. Therefore one should add more requirements to $\mathbf{FinSet}^{\omega}$ in order for it to correspond to a category of computations.

One interesting point to note is that f as described could be tabled into batches, precomputed, or partialy-evaluated. For instance, for each i we could store the results of the ith and following $i+1$th machine in a table. Furthermore, we could batch (table) enormous finite amounts of results of f, precomputing and storing the results, but despite that one cannot precompute the whole function. It is like the program running the algorithm must be infinite to be able to compute the function to the whole domain of the input. Keeping that in mind we will develop the notion of batching using natural transformations.

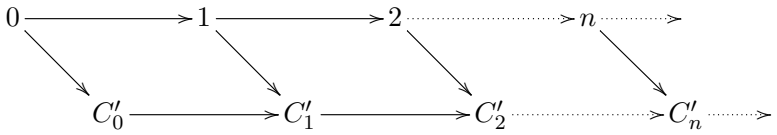
4.3 Encoding computations as natural transformations

An alternative view on the previous encoding of computations is to go higher in abstraction and use natural transformations between $FinSet^{\omega}$ functors, thus elements η typed as $\eta : \mathbf{FinSet}^{\omega} \to \mathbf{FinSet}^{\omega}$.

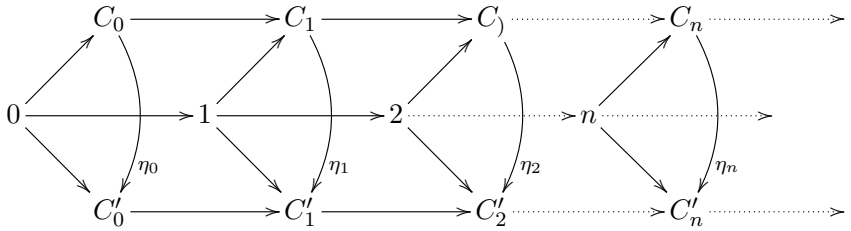
Let C and C' be functors in $\mathbf{FinSet}^{\omega}$ where C corresponds to the computations of some Turing machine and similar for the C' case. One is able to define a natural transformation $\eta : C \to C'$ such that for each $i \in \omega$ we have $\eta_i : C(i) \to C'(i)$. Take C represented graphically:



and C' represented graphically:



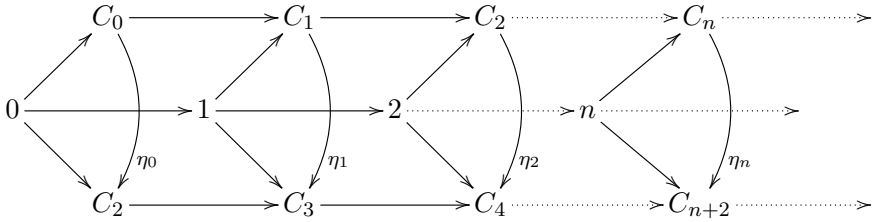
Then a natural transformation $\eta : C \to C'$ is depicted as:



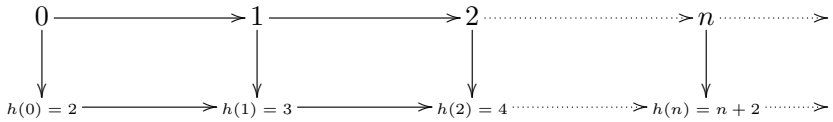
a structure transforming each element of C into and element of C' satisfying the naturallity condition.

Before delving into naturality, let us present a simple example. Choose C as the execution of some arbitrary Turing machine T and C' the execution configurations

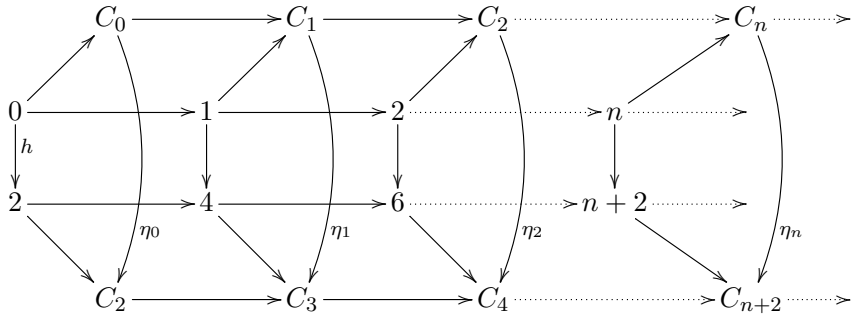
of a Turing machine that executes two steps of T at each time interval.



The fact that η is a natural transformation expresses the known fact that it is possible to group the computation in batches. To realize that just pick the morphism $h : \omega \to \omega$ in the ω category, where $h(i) = i+2$, which intuitively groups consecutive elements of ω, that is, it creates covers in the topology of the departing ω, and it is depicted as:



The property arising from naturality states that the following diagram commutes:



and encodes the property which states one is able to compute an arbitrary number of steps of computation and then look two states ahead. Or compute the same arbitrary number of steps in batches.

The intuition behind the necessary condition to be able to model computable functions is that the number of steps needed to perform a computation should be finite. That is to say the inherent logic involved in the production of output is finite. That corresponds to find a limit in the batching of steps.

$$\lim_{n \to \omega} \eta_n = \eta_t$$

Such condition is again necessary but not sufficient, if understood in the sequence of configurations of the Turing machine it states that at some point in time t the δ function is completely defined and can be stored as a finite table.

5 Which elements of FinSet$^\omega$ are computable?

Let us now look at it in the reverse direction: What are the functors that indeed may be computed, or put in other words, that have a Turing machine associated?

Our conjecture is that such functors are the ones for which there exists a cover of the topology induced by F on ω. In other words, only functors F that are compact should be considered effective.

To observe such cover and analyse the topology of $\mathbf{FinSet}^{\omega}$ stepwise evolution (computation steps) we augment the model by adding another layer of ω to track batching, thus obtaining a functor $(\mathbf{FinSet}^{\omega})^{\omega} = \mathbf{FinSet}^{\omega\times\omega}$ as depicted in Fig. 3. By convention we assume the vertically growing ω encodes stepwise machine execution, and that the horizontally growing ω encodes the batching transitions. Using that encoding a transition of a Turing machine is a natural transformation between $\mathbf{FinSet}^{\omega}$ functors. Therefore, each horizontal step prescribes a morphism η between F_i and F_{i+1}.

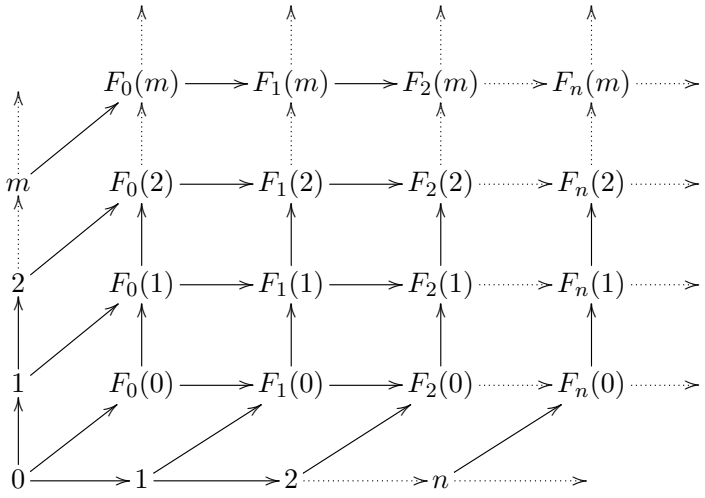


Fig. 3. The $\mathbf{FinSet}^{\omega\times\omega}$ model.

We characterize the augmented model as a manifold, the computational manifold, where each element $F_n \in \mathbf{FinSet}^{\omega}$, each $(n-1)$-step computation trace is a projection (a map in the usual manifold terminology) from the whole maifold (atlas) $\mathbf{FinSet}^{\omega\times\omega}$ satisfying the compatibility condition. Intuitively, such condition ensures that maps depicting common points in the manifold should be compatible, this is, the intersection of the maps in the neighbourhood of such point should have the same shape. 5the characterization we will use the notion of bundle, sieve, and the

Proposition 5.1 *The category* $\mathbf{FinSet}^{\omega^{\omega}}$ *can be put in correspondence with a sheaf on the site* $(\omega \times \omega, \mathbf{FinSet}^{\omega})$

Proof. Let T be a Turing machine, and the functor $F_T : FinSets^{\omega\times\omega}$ defined as above. Equip $\omega \times \omega$ with the Grothendieck topology. Then F_T satisfy the compatibility condition, thus F_T is a sheaf. □

As hinted, there is an injective mapping each Turing machine T into a member of $\mathbf{FinSet}^{\omega\times\omega}$, with such $T \mapsto F_T$ injective mapping one is able to prove the objects

of $\mathbf{FinSet}^{\omega\times\omega}$ of the form F_T for some Turing machine T form a subcategory **Tur**. Thus, the question: What about computable elements in $\mathbf{FinSet}^{\omega\times\omega}$? What are they?

Proposition 5.2 *(Conjecture/Goal:) Is* **Tur** *a reflective subcategory of* $(FinSets^{\omega})^{\omega}$*? If yes we have that every computational sheaf* C *is essentially* F_T *for some T.M.* T*.*

5.1 Effective functions as manifolds with programs as projections

In the tradition of Local/Global approaches for describing mathematical objects where a "global" (non-functional) object as a sphere is studied using "local" (functional) projections as maps of the whole atlas, we present a computational model that we term a computational manifold Such modle is a manifold with extra structure as:

- Topological manifolds are objects that are *locally* continuous, i.e., each point has a neighbor whose chart is a continuous function.
- Differential (C^k) manifolds are objects that are *locally* (C^k) differentiable, i.e., each point of the it has an open neighbour whose chart is C^k.

Our computational model is a manifold where objects are *locally* computable.

6 Conclusion

The status of our current approach to a model of effectiveness based on finite sets is still ongoing, but as an outcome we already established some important steps. We show how to encode Turing machine behaviors as a functor in $\mathbf{FinSet}^{\omega}$. We related the functoriality to the necessary condition that to produce a finite amount of input a Turing machine must consume only a finite amount of input. We express n-steps (batches) of computation as a natural transformation, and use the notion of batches to devise a restriction on the elements of $\mathbf{FinSet}^{\omega}$ that are indeed computable by a Turing machine.

Related work

We built a model realizing a model-theoretical approach for effectiveness using **CT** that is different from the Effective Topos **Eff** and from [16]. It is different from **Eff**, the topos emerging from a category of sets equipped with a symmetric and transitive relation and equivalence classes of functional relations between such sets, since it does use a simpler intuitionist topos without Natural Numbers Object (NNO), while **Eff** uses a non-classical one and has a NNO. It is also different from the approach described by [16] since it uses NNO too. Our framework has no infinite object inside the category. We think that this can be seen as an advantage over [16] that strongly depends on free objects, such as monoids for input and output data. These input/output data are not essential in our approach.

In [4], a categorical presentation of recursiveness is provided using **CT**. It axiomatizes categories able to define primitive recursive morphisms in a completely abstract way. Using the internal language of the category it is possible to precisely define any primitive recursive function. This work is very interesting, since, it joins in a quite harmonious way a model-theoretic definition with a proof-theoretic one. The identity present in the meta-theory provides meaning for a theory of equality between intentionally distinct ways of defining the primitive recursive functions. Besides that no mention on a concrete numerical system of even richer definition of natural number is needed, but the one need to define primitive recursiveness.

Recently there were attempts to change the status of the Church-Turing from a unprovable thesis to a formal proof [3]. Our goal is not a proof such a huge result, but first steps into the definition of effectiveness.

Future work

The model of computations presented does not model environmental input, the input tape content is fixed at the beginning of computation. Even though the tape content is passible of growing by action of the machine execution one cannot accept infinite input that is being generated by another machine, for instance one machine is producing an infinite decimal expansion of π while other machine is processing such output as input. For that reason future work could address that issue either by enriching the model or just reinterpreting $\mathbf{FinSet}^{\omega\times\omega}$. We also envision a better and formalized proof of the claim equating turing machines with locally computable sheaves in $\mathbf{FinSet}^{\omega\times\omega}$.

Another path for further study is the comparison and modeling of of other paradigms of computability. For instance recursive functions on sequences of naturals. And how our topological structure relates to the topologies arising from domain theoretical studies, e.g. the Scott topology.

Acknowledgement

The present work was supported by a grant from the CNPq, Conselho Nacional de Desenvolvimento Científico e Tecnológico - Brasil

References

[1] Archimedes and T. L. Heath, "The Works of Archimedes with the Method of Archimedes," Dover Publications, New York, 1953.

[2] Dehaene, S., N. Molko, L. Cohen and A. J. Wilson, *Arithmetic and the brain*, Current opinion in neurobiology **14** (2004), pp. 218–224.

[3] Dershowitz, N. and Y. Gurevich, *A natural axiomatization of computability and proof of church's thesis*, Bulletin of Symbolic Logic **14** (2008), pp. 299–350.

[4] Eilenberg, S. and C. C. Elgot, "Recursiveness," Academic Press, 1970.

[5] Goldblatt, R., "Topoi: the categorial analysis of logic," Dover, 2006.

[6] Haeusler, E. H., *Finiteness and computation in toposes*, Preliminary Proceedings of the 11th International Workshop on Developments in Computational Models (DCM 2015) http://dcm-workshop.org. uk/2015 (2015), p. 27.

[7] Hyland, J. M. E., *The effective topos*, Studies in Logic and the Foundations of Mathematics **110** (1982), pp. 165–216.

[8] Johnstone, P. T., *Sketches of an elephant: A topos theory compendium-volume 1*, Sketches of an Elephant: A Topos Theory Compendium-Volume 1, by Peter T Johnstone, pp. 562. Foreword by Peter T Johnstone. Oxford University Press, Nov 2002. ISBN-10: 0198534256. ISBN-13: 9780198534259 **1** (2002).

[9] Lambek, J. and P. Scott, "Introduction to Higher Order Categorical Logic," Number 7 in Cambridge Studies in Advanced Mathematics, Cambridge University Press, 1986.

[10] Machtey, M. and P. Young, "An introduction to the general theory of algorithms," North-Holland, 1978, vii, 264 p. ; pp.

[11] McLarty, C., "Elementary categories, elementary toposes," Oxford University Press, 1992.

[12] Rogers, H., "Theory of Recursive Functions and Effective Computability," MIT Press, 1987, 141–146 pp.

[13] Shapiro, S., *Varieties of pluralism and relativism for logic*, A companion to relativism (2011), pp. 526–552.

[14] Smith, D. R., *Mechanizing the development of software*, NATO ASI SERIES F COMPUTER AND SYSTEMS SCIENCES **173** (1999), pp. 251–292.

[15] Srinivas, Y. V. and R. Jüllig, *Specware: Formal support for composing software*, in: *Mathematics of Program Construction*, Springer, 1995, pp. 399–422.

[16] Walters, R. F. C., "Categories and computer science," Cambridge University Press, 1991.