Realizability Models and Complexity Classes

Eugenio Moggi[1]

[1]DIBRIS, Genova Univ., v. Dodecaneso 35, 16146 Genova, Italy

Abstract

Realizability models allow to define categories with very rich mathematical structure starting from untyped computational models, typically partial Combinatory Algebras (pCAs).

We propose a generalization of realizability models, in which pCAs are replaced by monoids of maps (on a set of data), which allow to consider also restricted computational models, where complexity constrains can be taken into account (e.g., the monoid of maps on bit strings computable in linear time). We give some examples of monoids (on bit strings) based on complexity classes, and state how the mathematical structure of these categories relates to properties of the corresponding monoid.

Keywords

Realizability models, Category theory, Complexity theory

1. Introduction

This communication is related to a journal submission, which proposes categories where one can interpret calculi for collection types (aka bulk types), like those in [1]. These calculi provide a framework for database query languages that go beyond traditional relational database, and have a lot in common with metalanguages for computational types [2], but there are also important differences

- Equality of collections is *decidable*, and can be added as an operation with a boolean result, while equality of programs is *undecidable*;
- Functional types and recursive definitions are available in most programming languages, but they are unacceptable in database languages, since they are incompatible with decidable equality.

Calculi for collection types are interpreted in categories with finite products, and collection types are interpreted by strong monads. In [3] Manes identifies certain *finitary* monads on the category $\mathbb{S}$ of sets, called collection monads, as an appropriate semantics for collection types. These monads have a well-behaved notion of membership, and collections have only finitely many members.

In the journal submission we propose to replace the category $\mathbb{S}$ of sets with (small) lextensive subcategories $\mathcal{C}$ of $\mathbb{S}$, whose arrows are maps computable by low complexity algorithms, e.g., working in linear or polynomial time. In this communication we report only some results, that could be of interest also for researchers focused on Complexity Theory, namely:

- The construction of categories associated to a monoid C of (total) maps on a set D (of data).
- Very weak conditions on C implying that the associated categories are lextensive and have a lextensive faithful *global section functor* into $\mathbb{S}$.
- More stringent conditions on C implying that the associated categories have a NNO (Natural Number Object) $\mathbb{N}$ and a list monad $\mathbb{L}$, i.e., a strong monad mapping an object X to the free monoid over X.

Realizability Models and Complexity Theory. The way realizability models use Category Theory in relation to Computability Theory works as follows: 1) one starts from a computational model,

ICTCS 2025: Italian Conference on Theoretical Computer Science, September 10–12, 2025, Pescara, Italy
moggi@unige.it (E. Moggi)
https://person.dibris.unige.it/moggi-eugenio/ (E. Moggi)
0000-0001-8018-6543 (E. Moggi)

typically a pCA $(D, \cdot)$, where the elements of D can be interpreted as data or programs and the binary application $e \cdot d$ (when defined) returns the output produced by executing program e with input d; 2) then one defines a category $\mathcal{C}$ using the pCA; 3) and finally one studies how category-theoretic properties of $\mathcal{C}$ relate to properties of the pCA. The seminal example of this approach is in [4], where Hyland defines the Effective topos using Kleene's first applicative structure, and then studies this topos and several of its sub-categories. In Complexity Theory the main focus is on complexity classes, i.e., sets C of maps (or predicates) on a set D (e.g., the set of strings A^* on an alphabet), while computational models play an ancillary role. Therefore, we consider a variant of realizability models, where the starting point is a sub-monoid C of $D \to D$ (i.e., C contains the identity on D and is closed under composition).

2. Lextensive Categories

Lextensive categories have finite limits (in particular products) and *well-behaved* finite sums, where the definition of well-behaved is formalized by the notion of extensive category (see [5]). A lextensive functor is a functor between lextensive categories preserving finite limits and finite sums, and a lextensive sub-category is a sub-category whose inclusion functor is lextensive. We use also the terminology lex category for a category with finite limits, and similarly for lex functor and lex sub-category.

From any (locally small) category $\mathcal{C}$ with a terminal object 1 (i.e., the product of zero objects) one can define the *global section functor* $\Gamma \stackrel{\Delta}{=} \mathcal{C}(1, -) : \mathcal{C} \longrightarrow \mathbb{S}$ to the category of sets. Such functor preserves all small limits existing in $\mathcal{C}$. We are mainly interested in lextensive categories such that Γ is faithful and lextensive. The following result gives sufficient conditions on $\mathcal{C}$ to have such a property.

Theorem 2.1. *If $\mathcal{C}$ is lextensive, non-trivial (i.e., $0 \ncong 1$) and has enough points (i.e., Γ is faithful), then $\Gamma : \mathcal{C} \longrightarrow \mathbb{S}$ is lextensive, moreover the action $\Gamma : \mathcal{C}(X, Y) \longrightarrow \mathbb{S}(\Gamma X, \Gamma Y)$ of Γ on each hom-set is bijective when X is a finite sum of 1s.*

The theorem above implies that a $\mathcal{C}$ satisfying the assumption is (equivalent to) a lextensive sub-category of $\mathbb{S}$, which contains the category $\mathbb{S}_f$ of finite sets as a full lextensive sub-category.

The rational for considering lextensive categories is that in them one can define

- an object of *booleans*, namely $2 \stackrel{\Delta}{=} 1 + 1$
- *decidable predicates*[1] on an object X, i.e., arrows $X \longrightarrow 2$
- when an object X has a *decidable equality*, i.e., a (necessarily unique) arrow $eq : X \times X \longrightarrow 2$ classifying the diagonal sub-object $X \rightarrowtail X \times X$.

In a lextensive category $\mathcal{C}$ some objects may not have a decidable equality, The following constructions define lextensive full sub-categories of $\mathcal{C}$, where all objects have a decidable equality.

Theorem 2.2. *If $\mathcal{C}$ is lextensive, let $\mathcal{C}_d$ be the full sub-category of the objects $X \in \mathcal{C}$ with a decidable equality, then $\mathcal{C}_d$ is a lextensive sub-category of $\mathcal{C}$.*

Theorem 2.3. *If $\mathcal{C}$ is lextensive and $X \in \mathcal{C}$ has a decidable equality and the objects 1, $X + X$ and $X \times X$ are retracts of X, let $\mathsf{P}[X]_d$ be the full sub-category of $\mathcal{C}$ consisting of the decidable sub-objects[2] of X, then $\mathsf{P}[X]_d$ is a lextensive sub-category of $\mathcal{C}_d$.*

In a lextensive category, where all objects have a decidable equality, one can interpret a typed language, where the BNF for types is $\tau ::= b \mid 0 \mid 1 \mid \tau_1 + \tau_2 \mid \tau_1 * \tau_2 \mid \{x : \tau | \phi\}$, with b base type and ϕ boolean formula (or equivalently a term of type $1 + 1$). A possible BNF for boolean formulas is $\phi ::= \top \mid e_1 = e_2 \mid \phi_1 \wedge \phi_2 \mid \neg\phi$, with e_1 and e_2 terms of the same type. Note that the last clause $\{x : \tau | \phi\}$ in the BNF for types means that the language has dependent types.

[1] This notion of decidability is not related to that in Computability Theory.

[2] Alternatively decidable predicates on X.

3. Category of Assemblies & co.

We introduce the **category of assemblies** $\mathsf{A}[C]$ for a sub-monoid C of the monoid $D \to D$ of total maps on a set D, and some of its full sub-categories. The aim is to provide realizability-like models for *low-complexity* Domain Specific Languages (DSL), where D is the set of data manipulated by *programs* in the DSL, and C are the maps *computed* by such programs. The idea (following [6]) is to replace application with composition and to abstract from programs in favor of functions, in order to achieve greater flexibility over realizability models based on pCA, like the Effective topos [4]. The definition of $\mathsf{A}[C]$, given below, is an instance of a construction in [7] based on *computability structures*.

Definition 3.1 (Assemblies). The category $\mathsf{A}[C]$ of **C-assemblies** is defined as follows

- an object X is a pair $(|X|, \vdash_X)$ with $|X|$ set and $\vdash_X \subseteq D \times |X|$ surjective, i.e., $\forall x \in |X|.\exists d \in D. d \vdash_X x$; a d s.t. $d \vdash_X x$ is called an **encoding** of x.
- an arrow $f\colon X \longrightarrow Y$ is a map $f\colon |X| \to |Y|$ s.t. $d \vdash_X x \implies f'(d) \vdash_Y f(x)$ for some $f' \in C$ called a **realizer** of f (notation $f' \vdash f$).

The faithful **forgetful functor** U from $\mathsf{A}[C]$ to $\mathbb{S}$ maps X to $|X|$.

Remark 3.2. If C is a sub-monoid of $D \to D$, then one can define a *computability structure* C with one datatype D and the set of relations $\mathsf{C}(D, D)$ consisting of the graphs of maps in C. In this way $\mathsf{A}[C]$ coincides with the category of assemblies for the computability structure C [7]. If $(D, \cdot)$ is a (total) Combinatory Algebra (CA), let C be the set of *computed* maps on D (i.e., the f s.t. $f(d) = e \cdot d$ for some $e \in D$), then C is a sub-monoid of $D \to D$, and the category of C-assemblies coincides with the category of assemblies for the CA $(D, \cdot)$. More generally, if $(D, \cdot)$ is an applicative structure with identity and composition combinators, then the set of computed maps is a sub-monoid of $D \to D$.

We now give some examples of C (and D) that do not arise from a CA.

Example 3.3. Examples of monoids on the set $\mathbb{N}$ of natural numbers are:

1. the set TR of total recursive maps
2. the set PR of primitive recursive maps
3. the set E_n of maps in the n-th level of Grzegorczyk hierarchy.

The following inclusions hold: $\mathsf{E}_n \subset \mathsf{E}_{n+1} \subset \bigcup_n \mathsf{E}_n = \mathsf{PR} \subset \mathsf{TR}$. E_3 coincides with the set of Kalmar elementary maps, that are defined without using bounded recursion.

Before giving examples of monoids on the set B^* of bit-strings, where $B = \{0, 1\}$, we recall some notions and results from Computational Complexity (see [8])[3]. We consider three types of complexity classes: output (size), time and space.

Definition 3.4. If $f\colon B^* \to B^*$ and $g\colon \mathbb{N} \to \mathbb{N}$ (monotone and ≥ 1), then

1. $f \in OUT(g) \overset{\Delta}{\iff}$ the map $n \mapsto \max_{|u| \leq n} |f(u)|$ is in $O(g)$
2. $f \in TIME(g) \overset{\Delta}{\iff}$ there is a TM computing f and working in time $O(g)$
3. $f \in SPACE(g) \overset{\Delta}{\iff}$ there is a TM computing f and working in space $O(g)$[4].

[3] We assume that a TM M computing a partial map on B^* has one read-only input tape (where the input is written before the computation starts), one write-only output tape (where the output is written before the end of the computation), several working tapes (on which M may use symbols from a larger alphabet $A \supseteq B$).

[4] For space complexity only the cells on working tapes used in the computation count, the cells on the read-only input tape and the write-only output tape are ignored.

$$\begin{array}{ccccc}
\mathsf{LO} = OUT(n) & \supset & \mathsf{LS} = SPACE(n) \cap \mathsf{LO} & \supset & \mathsf{LT} = TIME(n) \\
\cap & & \cap & & \cap \\
\mathsf{PO} = \bigcup_{k>0} OUT(n^k) & \supset & \mathsf{PS} = \bigcup_{k>0} SPACE(n^k) \cap \mathsf{PO} & \supseteq & \mathsf{PT} = \bigcup_{k>0} TIME(n^k) \\
 & & & & \cup| \\
 & & & & \ell\mathsf{S} = SPACE(\log n)
\end{array}$$

Figure 1: Sub-monoids of $B^* \to B^*$ and their inclusion relations.

The output classes contain also non-computable maps, but they are useful in combination with the classes defined in terms of Turing Machines. For instance, the class $TIME(n^k)$ is not closed under composition when $k > 1$, while $TIME(n^k) \cap OUT(n)$ is a sub-monoid of $B^* \to B^*$.
We recall well-known inclusions between different types of complexity classes (see [8, Thm 9.4]):

$$TIME(g) \subseteq SPACE(g) \cap OUT(g) \quad \text{and} \quad SPACE(g) \subseteq \bigcup_{c>0} TIME(n * c^{g(n)})$$

Lemma 3.5. *If all maps $g_i, t_i, s_i \colon \mathbb{N} \to \mathbb{N}$ are monotone, then the following hold:*

1. *If $f_i \in OUT(g_i)$ for $i = 1, 2$, then $\exists c > 0. f_2 \circ f_1 \in OUT(g_2(c * g_1(n)))$.*
2. *If $f_i \in TIME(t_i) \cap OUT(g_i)$ for $i = 1, 2$, then $\exists c > 0. f_2 \circ f_1 \in TIME(t_1(n) + g_1(n) + t_2(c * g_1(n)))$.*
3. *If $f_i \in SPACE(s_i) \cap OUT(g_i)$ for $i = 1, 2$, then $\exists c > 0. f_2 \circ f_1 \in SPACE(s_1(n) + \log g_1(n) + s_2(c * g_1(n)))$.*

Example 3.6. By Lemma 3.5, the subsets in Figure 1 are sub-monoids of $B^* \to B^*$.

We consider two remarkable full sub-categories of $\mathsf{A}[C]$: $\mathsf{M}[C]$ of modest sets and $\mathsf{P}[C]$ of predicates.

Definition 3.7 (Special assemblies). Given an assembly $X \in \mathsf{A}[C]$ we say that

1. X is a **modest set** ($X \in \mathsf{M}[C]$) $\overset{\Delta}{\iff}$ $\vdash_X$ is the graph of a partial surjective map from D to $|X|$.
2. X is a **predicate** ($X \in \mathsf{P}[C]$) $\overset{\Delta}{\iff}$ $|X| \subseteq D$ and $\vdash_X$ is the identity relation $\Delta_{|X|}$ on $|X|$.

$\mathsf{M}[C]$ is *replete* (i.e., an assembly isomorphic in $\mathsf{A}[C]$ to a modest set is a modest set) and essentially small, because it is equivalent to the small category of PERs (Partial Equivalence Relations) on D. $\mathsf{P}[C]$ is small, but not replete. However, one can modify the definition of $\mathsf{P}[C]$ to make it replete.
The following properties of C imply that $\mathsf{A}[C]$, $\mathsf{M}[C]$ and $\mathsf{P}[C]$ satisfy the assumptions of Thm 2.1.

Definition 3.8. Given a sub-monoid $C \subseteq D \to D$ we consider the properties:
K D has at least two elements (say b_0 and b_1) and $\lambda x \colon D. d \in C$ for every $d \in D$
P there is a map $p \colon D^2 \to D$ and two maps $p_0, p_1 \in C$ s.t.
- $p_i(p(x_0, x_1)) = x_i$ for every $x_0, x_1 \in D$
- $\lambda x \colon D. p(f_0 x, f_1 x) \in C$ for every $f_0, f_1 \in C$

E there is a map $t \in C$ s.t. $t(p(p(x_0, x_1), p(x_2, x_3))) = (x_2 \text{ if } x_0 = x_1 \text{ else } x_3)$.

Property (P), i.e., encoding of pairs, is needed to express (E), i.e., equality testing; (K) and (P) implies that D is infinite. All monoids in Examples 3.3 and 3.6 satisfy properties (K,P,E).

Theorem 3.9. *If a sub-monoid $C \subseteq D \to D$ has properties (K,P,E), then:*

1. $\mathsf{A}[C]$, $\mathsf{M}[C]$ *and* $\mathsf{P}[C]$ *are lextensive categories, and* Γ *is lextensive.*
2. *In* $\mathsf{P}[C]$ *every object has a decidable equality.*
3. *The forgetful functor* $U\colon \mathsf{A}[C] \longrightarrow \mathbb{S}$ *is isomorphic to* Γ.
4. *A map* e *is epi in* $\mathsf{A}[C]$ *(*$\mathsf{M}[C]$ *or* $\mathsf{P}[C]$*)* $\iff \Gamma e$ *is epi in* $\mathbb{S}$.

When $D \neq \emptyset$, the following relations among categories hold, where $F \dashv G$ means "F left-adjoint to G"

$$\mathsf{P}[C] \hookrightarrow \mathsf{M}[C] \underset{\hookrightarrow}{\overset{\leftarrow}{\perp}} \mathsf{A}[C] \overset{U}{\underset{\leftarrow}{\overset{\rightarrow}{\perp}}} \mathbb{S} \tag{1}$$

Moreover, under the assumptions of Theorem 3.9, one has the following lextensive functors, where $\mathbb{S}_\omega$ is the category of countable sets. Moreover, $\mathsf{P}[C]$ and $\mathsf{M}[C]$ are equivalnet to $\mathbb{S}_\omega$ when $C = D \to D$ with D countable.

$$\begin{array}{ccccc} \mathbb{S}_f & \hookrightarrow & \mathbb{S}_\omega & \hookrightarrow & \mathbb{S} \\ \downarrow & & & & \uparrow U \\ \mathsf{P}[C] & \hookrightarrow & \mathsf{M}[C] & \hookrightarrow & \mathsf{A}[C] \end{array} \tag{2}$$

If $\mathcal{C}$ has finite products, then one can ask whether $\mathcal{C}$ has a NNO $1 \xrightarrow{0} N \xrightarrow{s} N$ or a *list monad* $\mathbb{L}$, given by the adjuntion between $\mathcal{C}$ and the category of monoids in $\mathcal{C}$ (in this case $\mathbb{L}1$ is a NNO).

Theorem 3.10. *If a sub-monoid $C \subseteq D \to D$ has properties (K,P,E) and is closed under primitive recursion, then* $\mathsf{A}[C]$, $\mathsf{M}[C]$ *and* $\mathsf{P}[C]$ *have a parametric NNO and a strong list monad* $\mathbb{L}$.

Among the monoids in Example 3.3 and 3.6 only TR and PR are closed under primitive recursion, in all other cases the category $\mathsf{A}[C]$ ($\mathsf{M}[C]$ and $\mathsf{P}[C]$) fails to have a NNO (and consequently a list monad).

A lex sub-category $\mathcal{C}$ of $\mathbb{S}$ may have a NNO-structure $1 \xrightarrow{0} N \xrightarrow{s} N$, which does not have the universal property of a NNO in $\mathcal{C}$, but it does in $\mathbb{S}$. In this case we say that the structure *behaves like* a NNO. Similarly, we say that a (strong) monad $\mathbb{L}$ on $\mathcal{C}$ *behaves like* a list monad, when it is the restriction of a list monad on $\mathbb{S}$. Since structures satisfying a universal property are unique up to (unique) isomorphism, NNO-structures in $\mathcal{C}$ that behave like a NNO are isomorphic in $\mathbb{S}$, but may fail to be isomorphic in $\mathcal{C}$. However, some of these unique isomorphisms in $\mathbb{S}$ are morphisms in $\mathcal{C}$, thus these structures form a preorder. For instance, in $\mathsf{P}[C]$, where C is any of the monoids in Figure 1, the unary and binary encodings, $\mathbb{N}_1$ and $\mathbb{N}_2$, of the natural numbers behave like a NNO. However, only the isomorphism from $\mathbb{N}_1$ to $\mathbb{N}_2$ in $\mathbb{S}$ is a morphism also in $\mathsf{P}[C]$.

It is relatively easy to prove (basically an encoding exercise), that for all C in Examples 3.3 and 3.6 the category $\mathsf{A}[C]$ ($\mathsf{M}[C]$ and $\mathsf{P}[C]$) have structures that behave like a NNO and like a list monad, with the caveat that for the monoid LO and its sub-monoid LS and LT, the obvious monad $\mathbb{L}$ behaving like a list monad is not strong, i.e., it does not have a *tensorial strength* $X_1 \times \mathbb{L}(X_2) \longrightarrow \mathbb{L}(X_1 \times X_2)$.

Declaration on Generative AI

The author have not employed any Generative AI tools.

References

[1] P. Buneman, S. A. Naqvi, V. Tannen, L. Wong, Principles of Programming with Complex Objects and Collection Types, Theoretical Computer Science 149 (1995) 3–48. doi:10.1016/0304-3975(95)00024-Q.

[2] E. Moggi, Notions of Computation and Monads, Information and Computation/information and Control 93 (1991) 55–92. doi:10.1016/0890-5401(91)90052-4.

[3] E. G. Manes, Implementing Collection Classes with Monads, Mathematical Structures in Computer Science 8 (1998) 231–276. doi:10.1017/S0960129598002515.

[4] J. M. E. Hyland, The effective topos, in: Studies in Logic and the Foundations of Mathematics, volume 110, Elsevier, 1982, pp. 165–216.

[5] A. Carboni, S. Lack, R. Walters, Introduction to extensive and distributive categories, Journal of Pure and Applied Algebra 84 (1993).

[6] D. S. Scott, Relating theories of the lambda calculus, To HB Curry: Essays on combinatory logic, lambda calculus and formalism (1980) 403–450.

[7] J. Longley, Computability structures, simulations and realizability, Mathematical Structures in Computer Science 24 (2014) e240201.

[8] C. H. Papadimitriou, Computational Complexity, Addison Wesley, 1994.