Towards a Sheaf-Theoretic Definition of Decision Problems

Damiano Mazza[1]

1 **CNRS, LIPN, UMR 7030, Université Paris 13, Sorbonne Paris Cité**
damiano.mazza@lipn.univ-paris13.fr

A functorial approach to decision problems. Traditionally [18, 9, 2], a decision problem L is defined as a function

$$L : \{0,1\}^* \longrightarrow \{0,1\}$$

where $\{0,1\}^*$ is the set of binary strings (*i.e.*, sequences of zeros and ones) and $\{0,1\}$ the set containing only 0 (no) and 1 (yes). This means that underlying the definition of every decision problem there is an *encoding*, often implicit, of its instances as binary strings (except of course when the instances are binary strings themselves, as it happens sometimes). This insightful abstraction, going all the way back to Gödel and Turing, is quite convenient because it allows us to focus on just one very simple, universal set of instances rather than having to deal with arbitrarily many of them. However, one may also complain that flattening everything into $\{0,1\}^*$ may cause interesting structure to be lost. The justification "this is fine because in the end CPUs manipulate only zeros and ones" is superficial: in reality, algorithms *do* exploit the structure of their input, often in sophisticated ways and any programming language used in practice offers much more than just binary strings as data type. This observation, which is a staple of programming languages theorists, may have been overlooked by structural complexity theorists: seeing everything as a binary string is perhaps *too much* of an abstraction.

The last half century of developments in (among others) algebra [12], geometry [3], topology [5], logic [13], theoretical physics [7] and, last but not least, the semantics of programming languages [1] has given time and again evidence that adopting a category-theoretic approach to mathematics may be highly beneficial over the more traditional set-theoretic approach, providing in particular the "right" level of abstraction. In a way, categories are "structured sets" and offer a natural alternative to something like $\{0,1\}^*$ for abstracting instances of decision problems. Accordingly, we believe that decision problems should be defined as suitable *functors*. In this note we outline a possible route in the direction of such a definition.

Sites and sheaves. *Toposes* were introduced by Grothendieck [3] as a generalization of topological spaces in the context of his work in algebraic geometry. By definition, a (Grothendieck) topos is a category equivalent to the category of *sheaves* on a *site*. A site is a category equipped with a so-called *Grothendieck topology*, a collection of families of arrows sharing the same target called *covering families* and satisfying certain conditions (which we will not detail here). Intuitively, the objects of a site $\mathbf{C}$ must be seen as "decomposable" into "pieces", the pieces being themselves objects. Accordingly, an arrow $f : y \to x$ should be thought of as a specific way in which y is a piece of x. Composition of arrows is just transitivity of the "being a piece of" relation, and identities are saying that each object is a piece of itself. A covering family $\{f_i : y_i \to x\}_{i\in I}$ is saying that x may be *entirely* decomposed into the pieces y_i as specified by the f_i, where by "entirely" we mean that no piece of x is left out. An essential point here is that such a decomposition need not be disjoint: some y_i and y_j with $i \neq j$ may "overlap", *i.e.*, share a piece.

A *sheaf* on a site $\mathbf{C}$ is a functor

$$F : \mathbf{C}^{\mathrm{op}} \longrightarrow \mathbf{Set}$$

satisfying the *sheaf condition.* Intuitively, if x is an object of $\mathbf{C}$, then the set $F(x)$ should be seen as containing global information on x which may be reconstructed from the local information about the pieces of x, provided this local information is consistent. Indeed, if $\{f_i : y_i \to x\}_{i\in I}$ is a covering family for x, then any global information $a \in F(x)$ induces local information $F(f_i)(a) \in F(y_i)$ for each piece of x just by virtue of F being a (contravariant) functor. The sheaf condition says that one may go in the other direction: one may take *consistent* local information $a_i \in F(y_i)$ (for all $i \in I$) and "glue it together" into a unique global information $a \in F(x)$ satisfying $F(f_i)(a) = a_i$ for all $i \in I$. Formally, consistent means that, for all $i, j \in I$ and $g : z \to y_i$, $h : z \to y_j$ such that $f_i g = f_j h$, we have $F(g)(a_i) = F(h)(a_j)$. One may think of x as a floor made of tiles y_i: if each pair of tiles sharing a border is painted with the same color, then the whole floor will be painted with the same color.

Given a site $\mathbf{C}$, we define $\mathrm{Sh}(\mathbf{C})$ to be the category of sheaves on $\mathbf{C}$ and natural transformations between them. This is called the *sheaf topos* on $\mathbf{C}$. By definition, every Grothendieck topos is equivalent to a sheaf topos on some site. It is a fundamental theorem of topos theory [10] that if $\mathbf{E}$ is a topos and x an object of $\mathbf{E}$, then the slice category $\mathbf{E}/x$ is also a topos, called the *slice topos* over x.

Decision problems as sheaves. Although we do not yet have a fully developed theory, it seems clear that not every site is computationally meaningful, *i.e.*, functorial decision problems will be sheaves on sites with a particular structure. The following is not necessarily the most general definition of such structure but it gives the idea for our purposes.

A *vocabulary* $\mathbf{B}$ is a site with the following properties:

- it is *directed*: there are no non-identity endomorphisms and every arrow is monic;
- its objects are partitioned in three sets: a finite set of *sorts*, a finite set of *morphemes* and a countable set of *terms*;
- arrows may only go in the direction sort $\to$ morpheme $\to$ term $\to$ term; in particular, the subcategories on sorts alone and on morphemes alone are discrete;
- a sort or morpheme x has only the trivial covering family $\{\mathrm{id}_x\}$;[1]
- every term x admits a finite covering family composed only of arrows $f : a \to x$ with a a morpheme.

The covering family of the last point may be shown to be unique and is called the *morpheme decomposition* of x. Let x be a term and let $\{f_i : a_i \to x\}_{i\in I}$ be its morpheme decomposition. We say that x is *closed* if, for each $i \in I$ and non-identity arrow $g : s \to a_i$ (so s is necessarily a sort), there exist $j \in I$, $j \neq i$ and $h : s \to a_j$ such that $f_i g = f_j h$. The closed terms are also called *instances.*

As an example, there is a vocabulary $\mathbf{Str}$ such that:

- it has one sort $\bullet$;
- it has four morphemes denoted by $\cdot 0 \cdot$, $\cdot 1 \cdot$, $\rhd\cdot$ and $\cdot\lhd$;
- it has terms of the form x (open string), $\rhd y$ (right-open string), $y\lhd$ (left-open string) and $\rhd y \lhd$ (closed string) for all $x, y \in \{0,1\}^*$, x non-empty;
- for each morpheme a there are as many distinct arrows $\bullet \to a$ as occurrences of $\cdot$ in a;

[1] Here we are using the definition of site using *coverages* [10] instead of Grothendieck topologies.

- for each term x and morpheme a, there are as many distinct arrows $a \to x$ as there are occurrences of a in x (disregarding the $\cdot$); composition of arrows $\bullet \to a$ with $a \to x$ is defined so that there are exactly $n+1$ distinct arrows $\bullet \to x$, one for each "junction" between two letters of x, plus its borders if x is open; for instance, if $f : \bullet \to \cdot 0 \cdot$ corresponds to the right dot and $g : \bullet \to \cdot 1 \cdot$ corresponds to the left dot, and if $h : \cdot 0 \cdot \to 01$ and $k : \cdot 1 \cdot \to 01$ correspond to the obvious substring inclusions, then $hf = kg$, corresponding to the junction between 0 and 1;
- given two terms x, y, there are as many distinct arrows $x \to y$ as there are occurrences of x as a substring of y; composition of these is defined in the obvious way;
- a family $\mathcal{F}$ is covering for a term x precisely when, for every arrow $g : z \to x$ with z a sort or morpheme, there exists an arrow $f : y \to x$ in $\mathcal{F}$ such that g factors through f; with this definition, the set of all arrows of the form $a \to x$ with a a morpheme is a covering family and it is indeed the morpheme decomposition of x.

Suppose that X is the set of instances of the problem we wish to define. Rather than mapping it to $\{0,1\}^*$, we will *embed* X in a vocabulary $\mathbf{B}$, that is, we will find a suitable vocabulary $\mathbf{B}$ such that its closed terms are (in one-to-one correspondence with) the elements of X. Note the radical change of perspective: rather than "squashing" X into $\{0,1\}^*$, we regard it as part of a larger world. The fundamental intuition is that the vocabulary "around" X describes the *local structure* of the elements of X, *i.e.*, how they may be decomposed into pieces. This is precisely the structure that is destroyed by mapping everything to $\{0,1\}^*$. Another, perhaps more illuminating way of looking at a vocabulary is to see it as a description of how an algorithm may "apprehend" inputs in X: how it may access them, read them, manipulate them, etc. Note that the same X may be embedded in different vocabularies: this corresponds to giving more or less power to the algorithms processing inputs in X.

The exact definition of functorial decision problem is still unclear. For the moment, let us define a *decision problem* simply as a sheaf $L : \mathbf{B}^{\mathrm{op}} \to \mathbf{Set}$ on some vocabulary $\mathbf{B}$. Intuitively, if x is an instance, $L(x)$ is the set of witnesses of the fact that x is a "yes" instance. In particular, if $L(x) = \emptyset$ then x is a "no" instance. Another possible intuition is that $L(x)$ is the set of solutions for x: for example, the set of satisfying truth assignments for a CNF x. The functoriality of L and the sheaf condition ensure that such solutions are built following the local structure of $\mathbf{B}$. This matches the idea, exemplified by the Cook-Levin theorem, that "computation is local" [2, Sect. 2.3].

From functorial to traditional decision problems, and back, via type systems. So we have replaced the "passepartout" set $\{0,1\}^*$ with a host of structured vocabularies, such that each problem may have its own. It is still possible however to recover the traditional definition. If X is our set of instances, the traditional definition starts with an encoding function $\mathsf{enc} : X \to \{0,1\}^*$; we may invert this and define a decoding function $\mathsf{dec} : \{0,1\}^* \to X$, which sends the string x to the instance it encodes, or to an arbitrary instance in case x does not encode anything. Now, if $\mathbf{B}$ is the vocabulary in which we embedded X, we may extend the decoding into a functor $\mathsf{dec} : \{0,1\}^* \to \mathbf{B}$ by considering $\{0,1\}^*$ as a discrete category. We may therefore define a functor $\mathrm{Lang}(L) : \{0,1\}^* \to \mathbf{Bool}$ as the composition

$$\begin{tikzcd}\{0,1\}^* \arrow[r, "\mathsf{dec}"] & \mathbf{B}^{\mathrm{op}} \arrow[r, "L"] & \mathbf{Set} \arrow[r, "\mathsf{cntr}"] & \mathbf{Bool}\end{tikzcd}$$

where $\mathbf{Bool}$ is the category $0 \to 1$ and cntr is the functor mapping $\emptyset$ to 0 and every non-empty set to 1. Since $\{0,1\}^*$ is discrete, $\mathrm{Lang}(L)$ is just a function from binary strings to the set $\{0,1\}$ (the objects of $\mathbf{Bool}$): it is a decision problem in the traditional sense.

So each functorial decision problem induces a traditional decision problem. Is the converse true? The answer is trivially yes: for every subset $S \subseteq \{0,1\}^*$, there is a sheaf $L_S : \mathbf{Str}^{\mathrm{op}} \to \mathbf{Set}$ such that $\mathrm{Lang}(L_S) = S$, because we may use an infinite set to encode S. This is why the current, unrestricted definition of functorial decision problem is unsatisfactory. The indiscriminate use of infinite sets is a "trick" to bypass the structure of instances, which is precisely what we wanted to avoid in the first place. While waiting to find the right definition, we already know that we may recover all *recursively enumerable*[2] languages in terms of a non-trivial sheaf, which uses infinite sets in a much more "constructive" way. What is even more interesting is that this is possible thanks to *type systems*, thus drawing an unexpected and enticing connection with the theory of programming languages.

In recent work, Melliès and Zeilberger [16] advocated that functors are just a very abstract notion of type systems. In particular, if we present a programming language $\mathbf{L}$ as a category, then a type system for $\mathbf{L}$ is just a functor

$$\mathbf{D} \longrightarrow \mathbf{L}$$

where $\mathbf{D}$ is a category of type derivations, *i.e.*, witnesses of the fact that a certain program may be typed in a certain way. Now, by a standard categorical device known as the *Grothendieck construction* (or, rather, its discrete version) it turns out that a presheaf (so, in particular, a sheaf) on $\mathbf{B}$ is the same thing as a special kind of functor *into* $\mathbf{B}$, called a *discrete fibration*. But, according to Melliès and Zeilberger, a functor into $\mathbf{B}$ is a type system for $\mathbf{B}$! This seems very strange because, when $\mathbf{B}$ is a vocabulary as in our case, we hardly think of the objects of $\mathbf{B}$ as programs. And yet, this surprising perspective is precisely the one we will adopt to show that every recursively enumerable language induces a non-trivial functorial decision problem.

Let $S \subseteq \{0,1\}^*$ be recursively enumerable. By definition, there is a program M such that, for all $x \in \{0,1\}^*$, $M\underline{x}$ (the program applied to $\underline{x}$) evaluates to $\underline{1}$ iff $x \in S$, where $\underline{x}$ and $\underline{1}$ are the representations of the binary string x and the boolean 1, respectively. We define a vocabulary $\mathbf{V}_M$ which, intuitively, has as objects all syntactic expressions of the form $M\underline{x}$ as well as all their subexpressions, and such that there is an arrow $N \to P$ for each occurrence of N as subexpression of P (there may be more than one). The Grothendieck topology is defined in the natural way: a family $\{N_i \to P\}_{i \in I}$ is covering when every subexpression of P is a subexpression of some N_i, *i.e.*, the N_i account for every "piece" of P.[3] That $\mathbf{V}_M$ is a vocabulary follows from the fact that expressions of a programming language are composed of a finite number of instructions. We embed $\{0,1\}^*$ in $\mathbf{V}_M$ by mapping each string x to $M\underline{x}$.

Now, when we take our programming language to be the λ-calculus, there are type systems called *intersection type systems* verifying the following property [11]: *there is a type* `true` *such that an expression P has type* `true` *iff P evaluates to $\underline{1}$*.[4] For an expression P, let $\mathrm{TypeDer}_A(P)$ denote the set of type derivations assigning type A to P, and let $\mathrm{TypeDer}(P)$ be the set of *all* type derivations for P, regardless of the type. Given an expression P of

[2] Recall that a language $S \subseteq \{0,1\}^*$ is *recursively enumerable* if it is *accepted* (not necessarily decided!) by a Turing machine, *i.e.*, there is a Turing machine accepting x when $x \in S$ and rejecting *or not terminating* when $x \notin S$.

[3] The formal definition of $\mathbf{V}_M$ is a bit subtler, in the style of the definition of $\mathbf{Str}$ above, but the informal description here gives the idea.

[4] For the acquainted reader, assuming the standard representation of booleans, $\mathsf{true} = \alpha \to \beta \to \alpha$ with $\alpha \neq \beta$ atomic.

$\mathbf{V}_M$, we define

$$L_M(P) := \begin{cases} \mathrm{TypeDer}_{\mathsf{true}}(P) & \text{if } P = M\underline{x} \text{ for some } x \in \{0,1\}^* \\ \mathrm{TypeDer}(P) & \text{otherwise} \end{cases}$$

It turns out that L_M is a sheaf on $\mathbf{V}_M$: this is because intersection type systems assign their type *locally*, *i.e.*, to assign a type to an expression P one must only know the type assigned to the *immediate* subexpressions of P. Moreover, by the property of intersection types mentioned above, we have $\mathrm{Lang}(L_M) = S$. Indeed, for all $x \in \{0,1\}^*$, $\mathrm{Lang}(L_M)(x) = 1$ iff $L_M(M\underline{x}) \neq \emptyset$ (by definition of $\mathrm{Lang}(L_M)$) iff $M\underline{x}$ has type true (by definition of L_M) iff $M\underline{x}$ evaluates to 1 (by the property of intersection types) iff $x \in S$ (by hypothesis).

So we took intersection types, a standard technology in the world of programming languages theory, regarded them as a discrete fibration (thanks to Melliès and Zeilberger), regarded this as a sheaf and obtained that every recursively enumerable language is a functorial decision problem in a non-trivial way. This is not just an amusing trick: as shown in [14, 15], this same methodology is at the basis of a type-theoretic reconstruction of the Cook-Levin theorem, one of the pillars of structural complexity theory.

The topos of a decision problem. Every standard complexity class is a subclass of the class of recursively enumerable languages, so the bulk of traditional complexity theory may, at least in principle, be reformulated under a (non-trivial) functorial perspective. But we do not just want to reformulate, we want to go beyond. This is where topos theory enters the scene. By definition, a functorial decision problem L on a vocabulary $\mathbf{B}$ lives in the sheaf topos $\mathrm{Sh}(\mathbf{B})$ and we may therefore associate with it the slice topos $\mathrm{Sh}(\mathbf{B})/L$, which we denote by $\mathrm{Top}(L)$. This is a sort of generalized topological space, which comes with an imposing arsenal of invariants from algebraic topology (cohomology, homotopy, etc. [10]). Indeed, remember that toposes were invented precisely to transfer methods of algebraic topology to algebraic geometry [3]. In the next section we explain how, in some interesting cases, $\mathrm{Top}(L)$ is the generalization of an algebraic object which played a central role in the proof of the dichotomy theorem for CSPs [4, 6, 19]. This is very interesting, because we now have a way of associating such objects not just to CSPs but, in principle, with *every* decision problem. The perspective of being able to apply methods of algebraic topology to structural complexity theory with such a level of generality, surpassing any previous work of this kind [8, 17], is very exciting, but it is of course too early to say whether it will develop into something fruitful.

Extending the CSP dichotomy. We saw that allowing unrestricted use of infinite sets in sheaves on vocabularies trivializes the theory. While we still do not know the "right" restriction to impose on shaves, we may certainly try the most drastic: consider only *finitary sheaves* $\mathbf{B}^{\mathrm{op}} \to \mathbf{FinSet}$, *i.e.*, restrict to *finite* sets. We call these shaves *regular decision problems* (rdps).

The name "regular" comes from the fact that the rdps on $\mathbf{Str}$ may be shown to be *exactly* the regular languages, in the sense of the subsets of $\{0,1\}^*$ recognizable by a finite state automaton. We see here an example of how the Grothendieck topology of different vocabularies describes different, more or less restrictive ways in which a program may access its input string: in the vocabulary $\mathbf{V}_M$ defined above, they may be accessed with the power of M, an arbitrary program of a Turing-complete language (*e.g.* the λ-calculus); in $\mathbf{Str}$, they may be accessed only one letter after another, as a one-directional, read-only automaton would do. When we couple this with the finite set restriction, we obtain exactly finite state automata (the image of $\bullet$ is the set of states of the automaton).

Interestingly, *constraint satisfaction problems* (CSPs) are also rdps. Indeed, given a CSP Γ, we may define a vocabulary $\mathbf{B}_\Gamma$ whose objects are conjunctive formulas on the constraints[5] and whose arrows are subformula relations. It is then possible to define an rdp L_Γ on $\mathbf{B}_\Gamma$ such that $\mathrm{Lang}(L_\Gamma) = \mathrm{CSP}(\Gamma)$ intuitively because: 1) the truth of a conjunction is given locally by the fact that each constraint is individually satisfied, which allows the sheaf condition to be met; 2) the domain of Γ is finite, which guarantees that we land in **FinSet** rather than **Set**.

So rdps are a simultaneous generalization of CSPs and regular languages. It is also not hard to see that every rdp still lies within NP. But there are many more rdps than regular languages or CSPs! Well-known examples of rdps which are *not* CSPs are: all sorts of variants of the CIRCUIT VALUE problem (and their complement), all sorts of reachability problems for directed or undirected graphs (and their complement), as well as HAMILTONIAN PATH; more generally, any P problem which is the verification task induced by an NP-complete rdp (like CIRCUIT VALUE for CIRCUIT SAT) is an rdp which is not a CSP.

We conjecture that there is no NP-intermediate rdp, *i.e.*, that rdps enjoy the same dichotomy as CSPs. This is a considerable strengthening of the dichotomy for CSPs and is therefore all the more interesting. We already know that the proofs [6, 19] of the CSP dichotomy theorem do not immediately extend to rdps. Indeed, these crucially rely on associating with a CSP L an algebraic object $\mathrm{Pol}(L)$ (its *clone of polymorphisms* [4]) such that if L and K are CSPs having different complexity, then $\mathrm{Pol}(L) \neq \mathrm{Pol}(K)$. Unfortunately, there is a CSP L and an rdp K with different complexity ($K \in \mathsf{P}$ while L is NP-complete) such that $\mathrm{Pol}(L) = \mathrm{Pol}(K)$. But not all is lost, and in fact this is where things get interesting: it turns out that the polymorphisms (of arity n) of K exactly correspond to natural transformations $K^n \Rightarrow K$; recall that the objects of $\mathrm{Top}(K)$ (introduced above) are natural transformations $F \Rightarrow K$ for *arbitrary* sheaves F, so $\mathrm{Top}(K)$ *strictly extends* $\mathrm{Pol}(K)$ and might therefore provide just what we need to separate L and K and "reboot" the CSP technology for rdps.

Acknowledgements. Partially supported by ANR project ELICA (ANR-14-CE25-0005).

References

1 Roberto Amadio and Pierre-Louis Curien. *Domains and Lambda-calculi*. Cambridge University Press, 1998.

2 Sanjeev Arora and Boaz Barak. *Computational Complexity - A Modern Approach*. Cambridge University Press, 2009.

3 Michael Artin, Alexandre Grothendieck, and Jean-Louis Verder, editors. *Séminaire de Géométrie Algébrique du Bois Marie - 1963-64 - Théorie des topos et cohomologie étale des schémas - (SGA 4) - vol. 1–3*, volume 269, 270, 305 of *Lecture Notes in Mathematics*. Springer, 1972.

4 Libor Barto. Constraint satisfaction problem and universal algebra. *SIGLOG News*, 1(2):14–24, 2014.

5 Ronald Brown. *Topology and Grupoids*. BookSurge Publishing, 2006.

6 Andrei A. Bulatov. A dichotomy theorem for nonuniform csps. In *Proceedings of FOCS*, pages 319–330, 2017.

7 Bob Coecke, editor. *New Structures for Physics*, volume 813 of *Lecture Notes in Physics*. Springer, 2011.

[5] As for the vocabulary $\mathbf{V}_M$, the exact definition is a bit subtler but this one conveys the intuition.

8 Joel Friedman. Cohomology in grothendieck topologies and lower bounds in boolean complexity. *CoRR*, abs/cs/0512008, 2005.

9 Oded Goldreich. *Computational complexity - a conceptual perspective.* Cambridge University Press, 2008.

10 Peter T. Johnstone. *Sketches of an Elephant: A Topos Theory Compendium.* Clarendon Press, 2003.

11 Jean-Louis Krivine. *Lambda Calculus, Types and Models.* Ellis Horwood, 1993.

12 Serge Lang. *Algebra.* Springer, 2002.

13 Saunders MacLane and Ieke Moerdeijk. *Sheaves in Geometry and Logic.* Springer, 1994.

14 Damiano Mazza. Church meets Cook and Levin. In *Proceedings of LICS*, pages 827–836, 2016.

15 Damiano Mazza. *Polyadic Approximations in Logic and Computation.* Habilitation thesis, Université Paris 13, 2017.

16 Paul-André Melliès and Noam Zeilberger. Functors are type refinement systems. In *Proceedings of POPL*, pages 3–16, 2015.

17 Ketan Mulmuley and Milind A. Sohoni. Geometric complexity theory: Introduction. *CoRR*, abs/0709.0746, 2007.

18 Christos H. Papadimitriou. *Computational complexity.* Addison-Wesley, 1994.

19 Dmitriy Zhuk. A proof of CSP dichotomy conjecture. In *Proceedings of FOCS*, pages 331–342, 2017.