A Functorial Approach to Reductions among Decision Problems

Damiano Mazza[1]

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

1 Understanding the structure of decision problems

Characterizing complexity classes is not enough. Structural complexity theorists are looking for *lower bounds*. They want to be able to prove results of the form: in the model of computation X, problem L needs at least $R(n)$ computational resources to be solved, where n is the size of the problem instance. Notoriously, as soon as X has a minimum of expressiveness and L is not artificially constructed, the question becomes so hard that it is generally considered hopeless.

Logical approaches to computational complexity usually focus on capturing complexity classes: the classic result in these fields is that a language belongs to a complexity class C if, and only if, it may be decided by a program of a given form or expressed by a formula of a certain sort. In this way, the algorithms for solving the problems of C are "formatted" to a shape whose nature is purely logical, as opposed to algorithmic, bringing the hope that a new arsenal of tools becomes available for studying C.

Unfortunately, no use of any such tool has been found so far. If the question is showing that $H \notin \mathsf{C}$ for some purportedly hard problem H, we would like to be able to say why H does not admit any algorithm of the shape given by our logical characterization of C. However, these "shapes" are usually so complex that we are left just as clueless as with the original definition. One may say that logical characterizations (such as those given by implicit computational complexity) are good at giving us information about the structure of the *programs* corresponding to a certain complexity, but they say strictly nothing about the structure of the *decision problems* that such programs are supposed to decide, and it is this latter information that we are missing in the question of proving lower bounds.

Reductions and completeness. Summing up, if we are given a "big enough" complexity class C and a problem H, it is usually extremely difficult to show an unconditional statement of the form $H \notin \mathsf{C}$ (*i.e.*, a lower bound), unless H is artificially constructed for such a purpose (for instance by diagonalizing over C itself). We even have deep negative results telling us that presently available techniques are bound to fail [2, 13]. However, we do have a very good way of showing *conditional* lower bounds. This is achieved through the notion of *hardness*, which in turn uses the notion of *reduction*: a problem L reduces to L' if solving L' implies solving L, with the same resource bounds. Given a complexity class D, a problem H is D-*hard* if every problem in D reduces to H. In that case, if C is closed under reductions, then we do have $H \notin \mathsf{C}$, albeit conditionally to $\mathsf{C} \subsetneq \mathsf{D}$. If we also have $H \in \mathsf{D}$, then H is said to be D-*complete*.

Decades after the introduction of the notions of hardness and completeness [4, 9, 7], we have hundreds of "natural" complete problems for every "natural" complexity class which is known to have complete problems at all. Moreover, there is a remarkable empirical phenomenon: every such problem is actually complete under extremely weak forms of reduction, such as quantifier-free projections [6] or NC^0 reductions [1]. Such reductions are almost void of computational content and usually are based on what are informally known as "gadgets": L reduces to L' because each "building block" forming the instances of L may be encoded by an assembly of "building blocks" (a "gadget") of L', and the reduction then pieces together these gadgets in a completely local way, *i.e.*, without needing to perform any non-trivial computation. We are tempted to say that this phenomenon reveals the true nature of reductions: it is not an algorithmic notion but an *algebraic* notion.

Our hope is that reformulating reductions with this viewpoint in mind will expose the algebraic structure of decision problems, yielding a completely new perspective which will possibly shed more light on the question of lower bounds. In stark contrast with the logical routes followed so far (descriptive complexity, implicit computational complexity), this approach does not try to translate the definition of a complexity class but the definition of decision problem itself, *i.e.*, it is at a fundamentally deeper level. This note briefly summarizes a possible way to take some first steps in this direction.

2 Functorial reductions

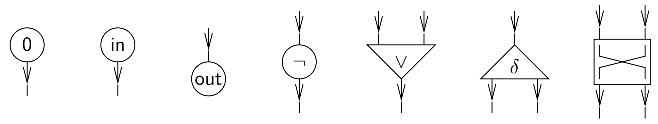
An example. A typical example of gadget reduction is the standard reduction r from CIRCUIT SAT to 3SAT, defined as follows. Let C be a Boolean circuit. If C is a circuit with inputs $x_1, \dots, x_n$, whose gates are $g_1, \dots, g_m$, our 3CNF formula $r(C)$ will be over the atoms $x_1, \dots, x_n, g_1, \dots, g_m$. We assign to each gate g a "gadget" $r(g)$, which will be a set of clauses made of at most three literals, depending on the type of g:

- g is an input gate associated with the variable x: $r(g) := \{(\neg g \vee x), (g \vee \neg x)\}$ (corresponding to $g \Leftrightarrow x$);
- g is a constant 0 gate: $r(g) := \{(\neg g)\}$ (corresponding to $g \Leftrightarrow 0$);
- g is a not gate whose input is the output of the gate h: $r(g) := \{(\neg g \vee \neg h), (g \vee h)\}$ (corresponding to $g \Leftrightarrow \neg h$);
- g is an or gate whose inputs are the outputs of the gates h and h': $r(g) := \{(\neg h \vee g), (\neg h' \vee g), (h \vee h' \vee \neg g)\}$ (corresponding to $g \Leftrightarrow (h \vee h')$).

In case g is also the output gate, we add the clause (g) to $r(g)$. (For brevity, we omit and and constant 1 gates, which may anyway be defined from or, not and 0). It is immediate to see that C is satisfiable iff the 3CNF formed of all the clauses $r(g_1), \dots, r(g_m)$ is satisfiable. It is also obvious that r is polynomial-time computable in m (the size of C). In fact, no non-trivial computation is needed at all: the reduction just maps each gate to its corresponding "gadget" (the structure of which is constant) and assembles everything together.

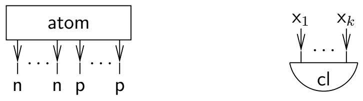
To try and formalize the intuition of "gadget reduction", it is tempting to consider problem instances as being the elements of some free structure, so that a gadget reduction becomes just a homomorphism. Let us apply it to the above reduction.

The instances of CIRCUIT SAT may be naturally presented as endomorphisms of the unit object of the free PRO (strict monoidal category whose objects are all tensor powers of a single generating object $\star$, whose 0-th power we denote by 1) generated by



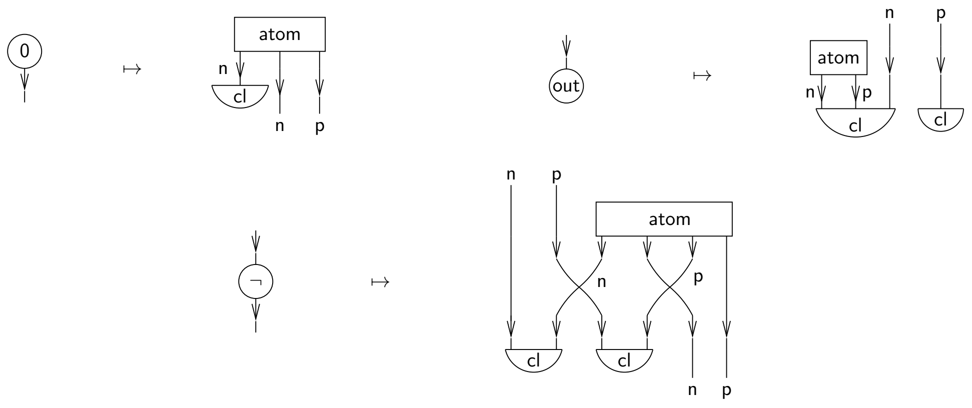
(We use string diagram notations to make the presentation more intuitive). We denote such a category by **Circ**. The generator 0 represents the Boolean constant "false"; the generators in and out represent an input and an output gate, respectively; the other generators are hopefully self-explanatory. An instance of CIRCUIT SAT is a morphism in $\mathbf{Circ}(1,1)$ (*i.e.*, a string diagram with no pending wires) containing one occurrence of out.

For 3SAT, we may see its instances as endomorphisms of the unit object of the following free strict symmetric monoidal category **3CNF**. Its objects are generated by p and n. Its morphisms are generated by



The generator atom stands for an atom, with p (resp. n) representing its positive (resp. negative) occurrences. The generators cl represent clauses with $k \leq 3$ literals, with $\mathsf{x}_1, \ldots, \mathsf{x}_k \in \{\mathsf{p}, \mathsf{n}\}$ depending on whether a literal occurs positively or negatively.

It is now fairly straightforward to implement the reduction r described above as a strict monoidal functor $r : \mathbf{Circ} \longrightarrow \mathbf{3CNF}$. On objects, we define $r(\star) := \mathsf{n} \otimes \mathsf{p}$. On morphisms, it is of course enough to define r on the generators; without being too verbose, let us give a few examples:



This hopefully gives the idea of how r is defined following the definition of the "gadget reduction" introduced above.

A more general picture. The above example inspires the following general framework. First, we take a leap of abstraction and define a decision problem as a particular type system, in the sense of Melliès and Zeilberger [11]. More precisely, we consider *strict monoidal Niefield fibrations* $p : \mathbf{E} \to \mathbf{B}$, which are certain functors introduced in [10], between strict compact closed categories $\mathbf{E}$ and $\mathbf{B}$. Such a fibration p is the generalization of a search problem, in the following sense: $\mathbf{B}$ is a category of "building blocks" of the instances of the problem, the instances themselves being endomorphisms of the unit 1; $\mathbf{E}$ is a category of "local solutions", so that, if $x \in \mathbf{B}(1,1)$ is an instance, $p^{-1}(x)$ is the set of all possible solutions of x. So, when we are interested in the decision version of the problem, what we want is to know whether, given $x \in \mathbf{B}(1,1)$, there exists $w \in \mathbf{E}(1,1)$ such that $p(w) = x$ (w is a "witness" of the fact that x is a "yes" instance). In type-theoretic terms, an instance x is a closed expression made of building blocks; it is a "yes" instance iff it is typable, with the typing rules given by the local solutions, and a type derivation w for x represents a solution for x.

The interest of having defined problems as Niefield fibrations comes from the Grothendieck construction: a Niefield fibration p as above is the same thing as a strong compact-closed functor

$$L : \mathbf{B} \longrightarrow \mathbf{Rel},$$

where **Rel** is the category of sets and relations with the usual compact-closed structure. This makes the decision version of the problem explicit: the unit of **Rel** (the singleton $\{*\}$) has only two endomorphisms, the identity and the empty relation, so L classifies instances of **B** as "yes" or "no", according to whether they are mapped to the former or the latter.[1] So, for example, circuits and 3CNFs may be presented by freely-generated strict compact-closed categories **Circ** and **3CNF** similar to the ones defined above, and CIRCUIT SAT and 3SAT may be seen as strong compact-closed functors **Circ** $\to$ **Rel** and **3CNF** $\to$ **Rel**, respectively. The former maps $\star$ to $\{0,1\}$ and the generators to the functions (seen as relations) corresponding to the logical gates (the out gate is mapped to $\{(1,*)\}$, *i.e.*, it may only take the truth value 1 as input, and has no output); the latter maps both p and n to $\{0,1\}$ and

$$\begin{aligned}\mathsf{atom} &\mapsto \{(*,(0,\dots,0,1,\dots,1)),(*,(1,\dots,1,0,\dots,0))\},\\ \mathsf{cl} &\mapsto \{((b_1,\dots,b_k),*) \mid b_1,\dots,b_k \in \{0,1\},\ b_1 \vee \cdots \vee b_k = 1\}.\end{aligned}$$

A *functorial reduction* is given by a strict compact-closed functor respecting the witness structure. In type-theoretic terms, it is a homomorphic encoding of one untyped language into another which preserves typability in both directions. For instance, for the reduction above, we have $r : \mathbf{Circ} \to \mathbf{3CNF}$ and there is a monoidal natural transformation θ

$$\begin{tikzcd} \mathbf{Circ} \arrow[rr, bend left, "\textsc{circuit sat}"] \arrow[dr, "r"'] & \Downarrow \theta & \mathbf{Rel} \\ & \mathbf{3CNF} \arrow[ur, "3\textsc{sat}"'] & \end{tikzcd}$$

such that $\theta_\star = \{(0,(1,0)),(1,(0,1))\}$, that is, the truth value b is mapped to $(\neg b, b)$. We invite the reader to check, on the three examples of the definition of r given above, that every possible way of decorating the left hand side with truth values consistently with the logical gate (*i.e.*, every admissible typing) is matched on the right hand side modulo the above mapping, and vice versa. Thanks to this, we have, for every circuit C, CIRCUIT SAT$(C) = $ 3SAT$(r(C))$. The existence of a monoidal natural transformation is actually a particularly well behaved situation; the general definition is a bit more complex. Nevertheless, a remarkable number of standard reductions, including all those presented in the chapter on NP-completeness of Papadimitriou's book [12], fit this description. It is quite amusing that, unknowingly, complexity theorists are defining monoidal functors all the time!

What now? We see this as step zero in a research program based on reformulating complexity theory in type-theoretic language, via category theory. The first goal of the program would be to reformulate class separations in terms of functorial reductions. For instance, if $\mathbf{Circ}_0$ is the subcategory of **Circ** of input-free circuits, then, one would want that

▶ Goal 1. P = NP iff there is a functorial reduction $r : \mathbf{Circ} \to \mathbf{Circ}_0$ from CIRCUIT SAT to CIRCUIT VALUE.

[1] For the reader acquainted with proof nets and their relational semantics, it is worth observing that a strong monoidal functor $L : \mathbf{B} \to \mathbf{Rel}$ with **B** free is just a rule for labelling the inputs and outputs of the generators of **B**. This induces labellings of the morphisms of **B** which, in the context of proof nets, would be called *experiments*. In particular, a morphism of **B** is typable iff there is an experiment for it, *i.e.*, a labelling respecting the rules given by L. The standard relational semantics is just a particular L in which **B** is the category of proof nets. It is known that such an L represents the decision problem telling whether a given proof net normalizes [5]. Depending on the fragment of proof nets, this problem may be undecidable, showing in particular that our approach is not limited to problems belonging to some fixed complexity class.

It is quite easy to achieve this goal with the non-uniform equality $\mathsf{P/poly} = \mathsf{NP/poly}$ instead. However, we believe that uniformity should also be understood in the above terms and we are currently working on Goal 1 as formulated above. This, of course, is where things become interesting and we have not gone far enough to tell more at present. The upshot, however, is that separation of classes becomes algebraic, opening the way for the second step of the program, which would be to develop a suitable notion of homology of monoidal categories (a notion currently also studied by Guiraud and Malbos) and harness it to build a wholly new approach to the question of lower bounds.

Another interesting phenomenon, which we could not address in this short note, is that in order to achieve Goal 1, even in the non-uniform case, functorial reductions need to be defined on compact-closed categories endowed with a certain sequentiality structure, *i.e.*, instances must come with an ordering of some sort. This is related to the fact that the standard definition of search problems is in terms of (binary) strings, which are linearly-ordered structures. Such a structure may be exploited by reductions and must be accounted for in the functorial formulation (the reduction r above does *not* use sequentiality structure, but it is the exception, not the rule). This is reminiscent of the question of *order-independent queries*, a subject of intense study (and deep conjectures) in descriptive complexity [6].

Ours is not the first attempt at formalizing the notion of "gadget" in reductions. Building on previous work by Bellare, Goldreich and Sudan [3], Trevisan, Sorkin, Sudan and Williamson [14] gave a formal definition of gadget and used it to prove non-approximability results on several well-known optimization problems. Their definition, however, is limited to constraint satisfaction problems, which is certainly a rich and interesting class of problems (3SAT is an example) but has a quite specific formulation and it is difficult to see how one may go beyond it. It turns out that our definition not only encompasses Trevisan et al.'s as a special case, but extends to *all* search problems. In fact, our functorial approach allows a unified view of several variants of decision problems: indeed, we observe that **Rel** is just the free bicompletion of the Boolean semiring [8]; by picking a different semiring $\mathcal{S}$, one gets different notions of problems as functors into $\mathbf{Mat}(\mathcal{S})$, the free bicompletion of $\mathcal{S}$. For instance, if $\mathcal{S}$ is the $(\max, +)$ (resp. $(\min, +)$) semiring, then one gets a generalization of optimization problems (maximizing and minimizing, respectively); if $\mathcal{S}$ is the natural numbers, one gets counting problems. It is an interesting question whether the techniques of Trevisan et al. carry over to this more general setting.

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

References

1 Manindra Agrawal, Eric Allender, Russell Impagliazzo, Toniann Pitassi, and Steven Rudich. Reducing the complexity of reductions. *Computational Complexity*, 10(2):117–138, 2001.

2 T. Baker, J. Gill, and R. Solovay. Relativizations of the p = np question. *SIAM Journal on Computing*, 4(4):431–442, 1975.

3 Mihir Bellare, Oded Goldreich, and Madhu Sudan. Free bits, pcps and nonapproximability—towards tight results. *SIAM J. Comput.*, 27(3):804–915, 1998.

4 Stephen Cook. The complexity of theorem proving procedures. In *Proceedings of STOC*, pages 151–158, 1971.

5 Daniel de Carvalho and Lorenzo Tortora de Falco. A semantic account of strong normalization in linear logic. *Inf. Comput.*, 248:104–129, 2016.

6 Neil Immerman. *Descriptive complexity*. Graduate texts in computer science. Springer, 1999.

7 Richard M. Karp. Reducibility among combinatorial problems. In *Proceedings of a symposium on the Complexity of Computer Computations*, pages 85–103, 1972.

8 Jim Laird, Giulio Manzonetto, Guy McCusker, and Michele Pagani. Weighted relational models of typed lambda-calculi. In *Proceedings of LICS*, pages 301–310, 2013.

9 Leonid Levin. Universal search problems. *Problems of Information Transmission*, 9(3):115–116, 1973.

10 Damiano Mazza, Luc Pellissier, and Pierre Vial. Polyadic approximations, fibrations and intersection types. *Proceedings of the ACM on Programming Languages*, 2(POPL:6), 2018.

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

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

13 Alexander A. Razborov and Steven Rudich. Natural proofs. *J. Comput. Syst. Sci*, 55(1):24–35, 1997.

14 Luca Trevisan, Gregory B. Sorkin, Madhu Sudan, and David P. Williamson. Gadgets, approximation, and linear programming. *SIAM J. Comput.*, 29(6):2074–2097, 2000.