Preprocessing of Intractable Problems[1]

Marco Cadoli, Francesco M. Donini[2] Paolo Liberatore, and Marco Schaerf
Dipartimento di Informatica e Sistemistica
Università di Roma "La Sapienza"
Via Salaria 113, I-00198, Roma, Italy
E-mail: {cadoli,donini,liberatore,schaerf}@dis.uniroma1.it

Some computationally hard problems –e.g., deduction in logical knowledge bases– are such that part of an instance is known well before the rest of it, and remains the same for several subsequent instances of the problem. In these cases, it is useful to preprocess *off-line* this known part so as to simplify the remaining *on-line* problem. In this paper we investigate such a technique in the context of intractable, i.e., NP-hard, problems. Recent results in the literature show that not all NP-hard problems behave in the same way: for some of them preprocessing yields polynomial-time on-line simplified problems (we call them *compilable*), while for other ones their compilability imply some consequences that are considered unlikely. Our primary goal is to provide a sound methodology that can be used to either prove or disprove that a problem is compilable. To this end, we define new models of computation, complexity classes, and reductions. We find complete problems for such classes, "completeness" meaning they are "the less likely to be compilable". We also investigate preprocessing that does not yield polynomial-time on-line algorithms, but generically "decreases" complexity. This leads us to define "hierarchies of compilability", that are the analog of the polynomial hierarchy. A detailed comparison of our framework to the idea of "parameterized tractability" shows the differences between the two approaches.

1. INTRODUCTION

1.1. Background and Motivations

In this paper we analyze in a formal way the idea of processing off-line part of the data representing an instance of a problem, in order to reduce the complexity of on-line computation. We refer to the preprocessing phase as *compilation*.

[1]This paper is an extended and revised version of [10].
[2]Current address: Politecnico di Bari, Dipartimento di Elettrotecnica ed Elettronica, Via Re David 200, I-70125 Bari, Italy.

The basic idea of compilation—simplifying a problem by putting off-line part of the computation—is as old as Mathematics. For example, the Babylonians used a table of numbers in order to make multiplications between integers easier (the operation of multiplication is time-consuming, and modern algorithms still try to replace multiplications with sums and subtractions whenever possible). Namely, they had tables containing, for any integer a up to a given value, the value $\lfloor a^2/4 \rfloor$. Let x and y be two integers to be multiplied. It holds that

$$x \cdot y = \frac{(x+y)^2}{4} - \frac{(x-y)^2}{4} = \left\lfloor \frac{(x+y)^2}{4} \right\rfloor - \left\lfloor \frac{(x-y)^2}{4} \right\rfloor$$

Using the table, it is possible to speed up the computation, as evaluating the right hand side of the equation requires only one sum $x+y$, one subtraction $x-y$, looking up twice the table for $\left\lfloor \frac{(x+y)^2}{4} \right\rfloor$ and $\left\lfloor \frac{(x-y)^2}{4} \right\rfloor$, and then one subtraction.

The whole procedure is a waste of resources if a *single* multiplication is to be done, since it requires to compute the squares of two numbers. However, the table allows to compute many products $x \cdot y$ in a very efficient way. We note that few table entries (about $x + y$ many) are sufficient for the method, while prestoring all products would be even more time efficient, but uses more space. In this sense, this is a typical example of compilation.

Let us now consider a simple problem on graphs: given an undirected graph and two of its nodes, determine whether they are connected or not. Off-line processing gives computational advantages also in this case. The simplest algorithm to solve this problem visits the whole graph, and its cost is $O(n + m)$ (n being the number of nodes and m the number of edges). Consider instead the following algorithm: determine the connected components of the graph, number them, and then write down a table with n entries which contains, for each node, the number of the connected component it belongs to. To determine whether two nodes are connected, it's enough to find out which connected components they belong to: if they are the same, the two nodes are connected, otherwise they are not.

Of course, determining the connected components is not any easier than searching the graph for a single path. However, if the graph is known in advance, then the table can be computed once and for all, and then verifying the existence of a path only takes $O(1)$ time. This preprocessing makes sense if two conditions hold:

1. Part of the instance (the graph) is known *in advance*, that is, some time before the pair of nodes is known and the result is needed.
2. It is necessary to perform several connectedness checks on the same graph.

If the former condition does not hold, preprocessing cannot be done. If the latter does not hold, preprocessing could still give some computational gain, although its benefits would be limited to a single instance. With respect to the first condition, note that, if the graph is not known in advance, processing it could be useful for the following instances.

Another, more realistic, example of preprocessing can be found in database technology, with the so-called *data-warehousing*. The results (usually called *views*) of specific queries to very large databases are stored separately from the database itself, thus allowing fast answers. Maintaining the data-warehouse, e.g., with respect

to updates, is worthwhile if there are many queries. The cost of queries is reduced by posing them to the data-warehouse rather than to the whole database.

From the formal point of view, the notion of *amortized* computational complexity [39] is quite useful in analyzing the benefits of off-line processing of data. In fact, amortized computational complexity is an adequate tool when several instances of the same problem must be solved. However, amortized complexity has been applied to polynomial-time solvable problems.

The focus of this paper is on problems that are (probably) not polynomial-time solvable: we want to provide tools that help classifying problems, telling under which conditions, after preprocessing, it is possible to solve them in polynomial time. If this is possible, we say that the problem is *compilable to* the class P of problems solvable in polynomial time, otherwise we say that it is *non-compilable to* P (the formal definitions are given in Section 2). Artificial Intelligence (AI) and Knowledge Representation offer a very simple paradigmatic problem that fits in the model described above: to decide whether a propositional formula γ is implied by a propositional knowledge base K, i.e., whether $K \models \gamma$, where $\models$ denotes classical logical consequence. It is well known that this problem is coNP-complete, when both K and γ are part of the input. In many AI domains however, a single knowledge base K must be queried many times, and it makes sense to preprocess it, putting it into a form that allows the solution of $K \models \gamma$ in polynomial time. Unfortunately, if the query γ can be any clause the problem is not compilable to P. In fact, we have shown in [13], slightly generalizing a result of Kautz and Selman in [25], that the existence of a data structure whose size is polynomial with respect to the size of the knowledge base, and from which one can correctly answer all queries in polynomial time, is equivalent to NP $\subseteq$ P/poly (which implies that Σ_2^p =PH, cf. [24], i.e., that the polynomial hierarchy collapses at the second level). However, the problem is compilable to P if we constrain γ to be a single literal: the compilation can be done by caching all answers –which are at most $O(|K|)$– in a boolean array. In Section 4.1 we show another restriction of γ which is sufficient to make the problem compilable to P.

We consider any decision problem whose instances can be partitioned into two parts, one which we call *fixed*, and one which we call *varying*. Compilation is a preprocessing of the fixed part, and delivers a *compiled structure*. In our model we do not impose any limit on the time of compilation, but only on the size of the compiled structure: in order to be feasible, the compiled structure must have size polynomial with respect to the size of the fixed part.

While keeping the constraint on the size of the compiled structure, it is possible to give up polynomiality of the on-line part of the computation, as long the complexity is decreased. As an example, starting from a PSPACE-complete problem, it could make sense to have "just" an on-line NP-complete problem to deal with.

1.2. State of the Art

During the last few years many researchers in AI have introduced various forms of *knowledge compilation* [36, 37] or *off-line reasoning* [32]. Specifically, they mainly focused on the problem of deciding whether a set of propositional clauses logically entails a clause. As an example, in [32] Moses and Tennenholtz show that, under

some non-trivial restrictions on the query language, it is possible to modify off-line the knowledge base so that clause inference can be decided on-line in polynomial time. In general, proposals for representation of all the theorems of a propositional knowledge base can be classified in three main methods: 1) use of prime implicants or prime implicates; 2) adding to the knowledge base only those prime implicates that make any deduction possible by unit resolution; 3) use of prime implicates with respect to a tractable theory. An early discussion on the usefulness of knowledge compilation in the context of AI can be found in [26, 5]. A more recent survey is [9]. Benefits offered by the technique have been analyzed from the experimental point of view (cf., e.g., [35]).

Feasibility/unfeasibility of off-line processing is strictly related to possibility/impossibility of representing each formula of logic $\mathcal{L}_1$ as a formula of polynomial size in logic $\mathcal{L}_2$. "Translations" of this kind have recently been investigated in the AI literature, specifically in the context of logics for common-sense and non-monotonic reasoning. As an example, several researchers investigated the feasibility of representing propositional circumscription [20] or propositional default theories [3] as purely propositional formulae. In some sense this is also a form of knowledge compilation, because reasoning in such non-monotonic formalisms is typically a problem complete for the second level of the polynomial hierarchy. Let T and γ be propositional formulae, and let $CIRC(T)$ (i.e., the circumscription of T) be the theory whose models are the minimal models of T (cf. Section 3.3 for the definition of minimal model). Determining whether $CIRC(T) \models \gamma$ or not is Π_2^p-complete [18]. If for each propositional formula T there is a formula T' of size polynomial in the size of T such that for all formulae δ, $CIRC(T) \models \delta$ if and only if $T' \models \delta$, then the inference problem after compilation becomes "just" coNP-complete. Computational properties of such translations have been investigated in several papers, here briefly listed:

- [12, 27] for formulae which are the outcome of the revision or update of a propositional formula,
- [14] for circumscription, and
- [21, 13] for default logic.

In each of the above listed papers the authors show cases in which this form of compilation is possible, and cases in which it does not seem to be possible. As an example, in [14] it is shown that, if circumscription can be translated in the way specified before, then NP $\subseteq$ coNP/poly, which implies that Σ_3^p =PH [40], i.e., the polynomial hierarchy collapses at the third level.

Another line of research bearing similarities with the idea of off-line processing is "fixed parameter tractability" [15, 17]. There are problems such that the input can be naturally split into two parts, e.g., whether a graph G has a clique of size k or not. In this case k is a parameter that can be fixed. Roughly speaking, a problem is fixed parameter tractable if it can be solved in time $f(k) \cdot n^c$, where n is the size of the input, c is a constant, and f is a function independent of the input. Although in both approaches the input to a problem is split into two parts, there are some major differences between the fixed parameter tractability of a problem and the feasibility of its off-line processing. In a nutshell, in our setting we assume

that there are several instances of the same problem, all sharing a fixed part, and we are not interested in finding an expression of the complexity in terms of both parts of the input. An extensive analysis of the similarities and differences of the two approaches is done in Section 5.

To sum up, the state of the art shows several non-compilability results already appeared in the literature. Nevertheless there is no clear understanding of the properties that make some problems compilable and others non-compilable. Moreover, there is no general methodology to prove non-compilability.

1.3. Goal and results

The goal of this paper is to formally investigate and characterize the notion of compilability of an intractable problem. We show that proving non-compilability in the absolute sense amounts to solve some long-standing problems in the theory of computational complexity, such as the non-collapse of the polynomial hierarchy. Therefore we devise tools for proving a problem to be "probably" non-compilable —in the same way as the notions of NP class and polynomial many-one reduction are tools for proving that a problem is "probably" intractable.

We also investigate off-line processing that decreases complexity, without necessarily leading to a polynomial-time on-line problem, e.g., a Σ_2^p-complete problem can be processed to obtain an on-line problem which is in NP. For this reason, we introduce a "compilability hierarchy". Since the polynomial hierarchy [38] has both theoretical and methodological importance (as an example, AI research has recently proved it to be useful in the design of algorithms for theorem proving in non-standard logics, cf. [22]), we believe in the importance of devising an analogous notion for compilability.

In particular, we define two hierarchies of problems, that are to compilation what the polynomial hierarchy is to efficiency (without compilation). For doing that, we introduce suitable models of computation, that allow an instance of a problem to be split into two different (fixed and varying) parts.

The first hierarchy is mainly used to define the properties of compilability of problems. We have for example a class containing all the problems that are *compilable to* P, that is, all the problems that can be solved in polynomial time if a preprocessing is allowed. Another class contains problems which are in NP after the compilation, and so on. A concept of reduction for this hierarchy is defined, along with the definition of complete problems for each class of the hierarchy.

The second hierarchy is the non-uniform version of the first one. In this sense, it generalizes both the first one, and the classical non-uniform hierarchy [24, 40]. A suitable concept of reduction is defined for this hierarchy. Some prototypical AI problems are complete for some classes of the non-uniform hierarchy, but they do not seem to be complete for any class of the uniform hierarchy. Thus, the non-uniform hierarchy is better suited for assessing the non-compilability of problems.

Provided that the polynomial hierarchy does not collapse, we prove that both our hierarchies are proper, that is, all the classes in them are distinct. The proposed framework allows for much simpler compilability and non-compilability proofs than those found in the literature (cf., e.g., [14, 12, 21]), using complete problems and reductions. We demonstrate the benefits of our proposal by providing several examples of compilable and non-compilable problems, taken both from AI and from

graph theory. These examples show that the two notions of complexity and compilability of a problem are distinct.

1.4. Outline

The structure of the paper is as follows. Section 2 contains definitions for the models of computation, reductions, complexity classes and hierarchies. It also contains basic properties of reductions and classes. Section 3 lists some problems which do not appear to be compilable, while Section 4 lists some which do. In Section 5 a detailed comparison of our approach with fixed parameter tractability is performed. Section 6 concludes the work with a summary and a discussion of possible developments. Almost all the proofs appear in the Appendix.

2. DEFINITIONS AND PROPERTIES

2.1. Languages and poly-size functions

We assume that languages are built over a finite alphabet Σ s.t. $|\Sigma| \geq 2$ that contains a special symbol # denoting blanks. Throughout the paper, we assume that any sequence of suffixed blanks has no influence, that is, given a language A, for every string $x \in \Sigma^*$, $x \in A$ if and only if $x\# \in A$. This assumption is needed in some of the subsequent proofs. However, we adopt the standard notion of size of the input, i.e., each blank counts as one character.

Any language $A \subseteq \Sigma^*$ implicitly defines the decision problem of checking whether a given string $x \in \Sigma^*$ belongs to A or not. In what follows we interchangeably use the term *language* A or *problem* A, with this meaning.

The *length* of a string $x \in \Sigma^*$ is denoted by $|x|$. The cardinality of a set S is denoted as $||S||$. Numbers will be represented in unary notation, that is the number k will be represented by a sequence of k symbols. A function $f : \Sigma^* \rightarrow \Sigma^*$ is called *poly-size* if there exists a polynomial p such that for all $x \in \Sigma^*$ it holds that $|f(x)| \leq p(|x|)$.

In this paper we are concerned with problems whose input is composed of two parts, where one part is fixed (the part that is preprocessed) and the second one is varying (only accessible on-line). Therefore, the problems we are interested in can be formally defined as sets of pairs of strings. A *language of pairs* S is a subset of $\Sigma^* \times \Sigma^*$.

2.2. Logical formalisms

We introduce some terminology for the logical problems we consider in the paper. We use a finite alphabet of propositional symbols, or letters, $L = \{a, b, c, \ldots\}$, possibly with subscripts. We admit the usual boolean connectives ($\wedge$, $\vee$, and $\neg$) for constructing well-formed formulae. Symbol $\neq$ denotes exclusive-or, and $\rightarrow$ denotes implication. A *literal* is a propositional symbol or its negation. A *clause* is a disjunction of literals. An *interpretation* for L is a mapping from L to $\{\mathsf{true}, \mathsf{false}\}$. Interpretations can be extended to boolean formulae in the usual recursive way. A *model* of a formula T is an interpretation M that maps T to true (written $M \models T$). A formula F is a logical consequence of another formula T (denoted as $T \models F$) if and only if $\forall M : M \models T$ implies $M \models F$.

2.3. Non-uniform complexity classes

We assume the reader is familiar with (uniform) classes of the polynomial hierarchy (PH), i.e., P, NP, Σ_2^p, ... and their complements, and with polynomial many-one reductions, denoted $\leq_m^p$-reductions in the rest of the paper. We now briefly introduce non-uniform classes, following Johnson [23].

DEFINITION 2.1. An *advice-taking Turing machine* is a Turing machine that has associated with it a special "advice oracle" A, which can be any function (not necessarily a recursive one). On input s, a special "advice tape" is automatically loaded with $A(|s|)$ and from then on the computation proceeds as normal, based on the two inputs, x and $A(|s|)$.

Note that the advice is only function of the *size* of the input, not of the input itself.

DEFINITION 2.2. An advice-taking Turing machine uses *polynomial advice* if its advice oracle A satisfies $|A(n)| \leq p(n)$ for some fixed polynomial p and all nonnegative integers n.

DEFINITION 2.3. If $\mathcal{C}$ is a class of languages defined in terms of resource-bounded Turing machines, then $\mathcal{C}$/poly is the class of languages defined by Turing machines with the same resource bounds but augmented by polynomial advice.

Any class $\mathcal{C}$/poly is also known as *non-uniform* $\mathcal{C}$, where non-uniformity is due to the presence of the advice. Non-uniform and uniform complexity classes are related in [24, 40]. In particular, Karp and Lipton proved in [24] that if NP $\subseteq$ P/poly then $\Pi_2^p = \Sigma_2^p =$ PH, i.e., the polynomial hierarchy collapses at the second level, while Yap in [40, p. 292 and Th. 2] generalized their results showing that if NP $\subseteq$ coNP/poly then $\Pi_3^p = \Sigma_3^p =$ PH, i.e., the polynomial hierarchy collapses at the third level. Such collapses are considered very unlikely by most researchers in structural complexity.

2.4. Compilability

According to our intuitive notion of compilability, a problem S is compilable if, given an instance $\langle x, y\rangle$ of it, the fixed part (x) can be preprocessed (thus obtaining $f(x)$) so that solving the problem on-line is simpler. Of course, this is worthwhile if the cost of solving the problem, given the varying part (y) and the result of the preprocessing $(f(x))$, is much simpler than solving the problem given the (non-preprocessed) fixed part and the varying part. Consider for example the problem 3CNF CLAUSE INFERENCE (CI). This is the problem of deciding, given a set x of clauses (each clause being composed of three literals) and a query y (a clause), whether x logically implies y or not. In terms of languages:

$$\text{CI} = \{\langle x, y\rangle \mid x \text{ is a 3CNF formula, } y \text{ is a clause, and } x \models y\}$$

As already pointed out, this is a prototypical problem in logic, AI, and computational complexity.

With the aim of making on-line reasoning polynomial, in the preprocessing phase we can generate and order all the clauses composed of variables of x that are implied

by x. The number of such clauses is exponential, but finite. Now, to decide whether $x \models y$, we have only to check whether there exists a clause implied by x which is a subset of y.

Let us analyze the computational cost of these operations: the preprocessing requires exponential time, and the result might require an exponential amount of memory to store the set of clauses. However, given the query y, deciding if it is implied by x is now only linear in the size of the initial set of clauses, since binary search can be applied to the ordered set of implied clauses.

This compilation is useful only if we can afford enough room to store the compiled structure. Since in most cases this is not realistic, we impose that the result of the compilation must have polynomial size. This is why we introduce poly-size functions. We use a poly-size function f in our definitions to formalize the fact that, given a string x, this string is processed obtaining a polynomial-size data structure $f(x)$.

Let us now formally define the concept of compilation. In the above example we considered a coNP-complete problem, and the informal definition of compilability was: the problem is compilable if, after preprocessing the fixed part, the solution can be determined in polynomial time. This concept is generalized in two directions in forthcoming Definition 2.4. First of all, problems in higher classes of the polynomial hierarchy are considered. Moreover a simpler-but-not-polynomial solving on-line algorithm is allowed: for example, we can have a Σ_2^p-complete problem, and it makes sense to compile the fixed part in such a way that the complexity of the resulting problem decreases to NP.

In the rest of the paper, we focus our attention on complexity classes conforming to the following assumption.

ASSUMPTION 2.1. *When referring to a complexity class* C*, we always assume that* C *is closed under* $\leq_m^p$*-reductions, and that there exist complete problems for* C*.*

Observe that if a complexity class C is conforming to the above assumption, then C contains the class P.

Given a complexity class C that conforms to Assumption 2.1, we introduce the class of problems compilable to C, that we denote as $\rightsquigarrow$C (pronounced "compilable to C"). The complement of this class is denoted as co-($\rightsquigarrow$C).

DEFINITION 2.4. [$\rightsquigarrow$C] Let C be a complexity class that conforms to Assumption 2.1. A language of pairs $S \subseteq \Sigma^* \times \Sigma^*$ belongs to $\rightsquigarrow$C if and only if there exist a poly-size function f and a language of pairs S' such that for all $\langle x, y\rangle \in \Sigma^* \times \Sigma^*$ it holds that:

1. $\langle f(x), y\rangle \in S'$ if and only if $\langle x, y\rangle \in S$;
2. $S' \in$ C.

This definition can be represented pictorially as in Figure 1. This schema captures our intuitive notion of compilability into C of a problem S with fixed and varying parts. The function f represents the compilation of the fixed part. In order to decide whether $\langle x, y\rangle \in S$, we process off-line the fixed part x, thus obtaining

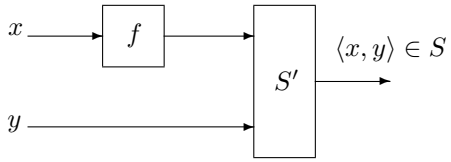

FIG. 1. Pictorial representation of the $\rightsquigarrow$C class

$f(x)$, and then we decide whether $\langle f(x), y\rangle \in S'$. The whole process is convenient if deciding $\langle f(x), y\rangle \in S'$ is easier than deciding $\langle x, y\rangle \in S$.

Note that no restriction is imposed on the *time* needed to compute the function f, but only on the *size* of the result, i.e., f is a poly-size function. The framework we present could be specialized by imposing restrictions on the computational resources used during compilation (e.g., we could require that the compilation phase is accomplished using polynomial space). We discuss in Section 6 the implications of such limitations.

Since most of the problems we discuss in this paper belong to classes of the polynomial hierarchy, it is suitable to define the *polynomial compilability hierarchy* as the set of classes $\rightsquigarrow$C such that C belongs to PH.

In particular, the class $\rightsquigarrow$P contains all the problems that can be solved in polynomial time on-line after an off-line processing of the fixed part. Many problems belong to this class, including problems where for every given fixed part x the number of distinct varying parts is bounded by a polynomial in $|x|$. This is shown by the following theorem (see the Appendix for the proof).

THEOREM 2.1. *Let S be a language of pairs and $W(S,x) = \{y \mid \langle x, y\rangle \in S\}$. If there exist two polynomials p_1, p_2 such that for every x it holds that $||W(S,x)|| \leq p_1(|x|)$ and for every $\langle x, y\rangle \in S$ it holds that $|y| \leq p_2(|x|)$, then $S \in \rightsquigarrow$P.*

Note that this is just a sufficient condition. In Section 4 we show problems in $\rightsquigarrow$P which do not satisfy the conditions of the above theorem. Note also that, given a complexity class C, the class $\rightsquigarrow$C contains problems that, when both the fixed and the varying part are given on-line, are undecidable. This proves that $\rightsquigarrow$C is an extension of the class C, in that more computational power is allowed in the preprocessing phase. In order to prove the non-membership of a problem to a class, we introduce suitable definitions of reduction and hardness.

We now recall the basic properties that must be satisfied by all reductions. We define these properties for languages composed of one input, but they can be readily applied to languages composed of pairs. First of all, if $A, B \in \Sigma^*$ are two languages, then $f : \Sigma^* \to \Sigma^*$ is a reduction between A and B if for any $x \in \Sigma^*$ it holds that $x \in A$ if and only if $f(x) \in B$. Let $\leq$ be a set of reductions (that is, a set of functions from strings to strings). We say that the problem A is $\leq$-reducible to B (written $A \leq B$) if and only if $\leq$ contains a reduction f from A to B.

Following [23] we introduce three additional properties that our reductions should satisfy.

PROPERTY 2.1 (Transitivity). *The reductions $\leq$ are transitive, i.e., $A \leq B$ and $B \leq C$ implies $A \leq C$.*

PROPERTY 2.2 (Compatibility). *The reductions $\leq$ are compatible with the class $L \subseteq \Sigma^*$, i.e., for any pair of problems A and B, if $B \in L$ and $A \leq B$, then $A \in L$.*

PROPERTY 2.3 (Existence of hard problems). *There exists a problem $B \in L$ such that, for any $A \in L$ it holds that $A \leq B$.*

The above properties easily generalize to reductions between languages of pairs. In order to have a class of reductions that are compatible with respect to the classes $\rightsquigarrow$C, and are powerful enough to allow the definition of complete problems, we introduce the notion of $\leq_{comp}$ *reduction*.

DEFINITION 2.5. [$\leq_{comp}$ reduction] A $\leq_{comp}$ *reduction* between two languages of pairs A and B is a triple $\langle f_1, f_2, g\rangle$, where f_1 and f_2 are poly-size unary functions and g is a binary polynomial-time function, such that for any pair of strings $\langle x, y\rangle$ it holds that:

$$\langle x, y\rangle \in A \quad \text{if and only if} \quad \langle f_1(x), g(f_2(x), y)\rangle \in B$$

Given two problems A and B, we say that A is *comp-reducible* to B if and only if there exists a $\leq_{comp}$ reduction between them, that is, $A \leq_{comp} B$. The $\leq_{comp}$ reductions can be represented as in Figure 2.

Intuitively, $A \leq_{comp} B$ if: 1) the fixed part of B can be obtained from the fixed part of A using a poly-size function (f_1), and 2) the varying part of B can be constructed using both a poly-size function (f_2) applied to the fixed part of A, and a polynomial-time function (g) applied to f_2 and to the varying part of A. The existence of a connection between the fixed part of the first problem and the varying one of the second (f_2) is essential for fulfilling Properties 2.1-2.3. However, in Section 3 we show various reductions that do not need to use the full power of the $\leq_{comp}$ reductions, i.e., they do not use the function f_2.

We remark here that the only bound on the functions f_1 and f_2 is the space needed to store their result: there is no bound on the time or space needed to calculate them. It would make sense to investigate what happens if f_1 and f_2 are assumed to be bounded in the time needed for the calculation of the result (e.g., exponential time) or in the space (e.g., polynomial space) as we already argued for the function f in the definition of $\rightsquigarrow$C. Some considerations on this issue can be found in [28], where a more detailed explanation of our choice of not bounding the running time/space of these functions is given.

The $\leq_{comp}$ reductions satisfy some basic properties of a reduction, as shown by the following result, that holds (as for the other results of this paper) if the considered class C conforms to Assumption 2.1.

THEOREM 2.2. *Let* C *be a complexity class that conforms to Assumption 2.1. The $\leq_{comp}$ reductions satisfy transitivity and compatibility with respect to $\rightsquigarrow$*C.

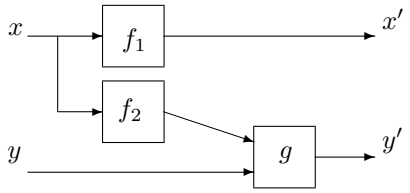

FIG. 2. $\leq_{comp}$ reductions

We now define the notion of *hardness* and *completeness* for $\rightsquigarrow$C using $\leq_{comp}$ reductions.

DEFINITION 2.6. [Hardness and Completeness] Let C be a complexity class that conforms to Assumption 2.1. A language of pairs B is $\rightsquigarrow$C-*hard* if and only if for any $A \in \rightsquigarrow$C it holds that $A \leq_{comp} B$. Moreover, B is $\rightsquigarrow$C-*complete* if and only if B is in $\rightsquigarrow$C and is $\rightsquigarrow$C-hard.

Using the above definition we can find complete problems for the class $\rightsquigarrow$C, provided that complete problems for the class C exist. Given a problem S with one input, we call ϵS the problem with two inputs defined as:

$$\epsilon S = \{\epsilon\} \times S = \{\langle x, y\rangle \mid x \text{ is the empty string and } y \in S\}$$

The following theorem shows how we can find complete problems (with respect to $\leq_{comp}$ reductions) out of complete problems under polynomial-time many-one reductions.

THEOREM 2.3. *For every complexity class* C *if* S *is* C*-complete (under polynomial time many-one reductions) then* ϵS *is complete (under* $\leq_{comp}$ *reductions) for the corresponding compilability class* $\rightsquigarrow$C*.*

The above result is not surprising since the fixed part is empty, and therefore, there is no possibility of taking advantage of preprocessing. As an example, starting from the NP-complete problem 3SAT, we obtain the $\rightsquigarrow$NP-complete problem ϵ3SAT. Of course, problems of this kind are not usual in practice. However, they will be used throughout the paper, since it is easy to prove the hardness of a problem by reducing an ϵS problem to it. For several important problems we can prove that they probably do not belong to $\rightsquigarrow$P by proving their $\rightsquigarrow$C-completeness for some C above P (cf. Theorem 2.11), and this is usually done by reducing another $\rightsquigarrow$C-complete problem to it (and often this $\rightsquigarrow$C-complete problem is an ϵS problem).

For example, it can be easily shown that the problem FORMULA INFERENCE (FI) defined as

$$\text{FI} = \{\langle x, y\rangle \mid x \text{ and } y \text{ are propositional formulae and } x \models y\}$$

is $\rightsquigarrow$coNP-complete.

THEOREM 2.4. *The problem* FI *is* $\rightsquigarrow$coNP-complete.

Classes of the form $\rightsquigarrow$C characterize our notion of compilability. However, they suffer from a severe technical problem. In fact, while we can prove that FI is non-compilable by showing that FI is $\rightsquigarrow$coNP-complete, the non-compilability proofs which appeared in the literature for many natural problems [25, 13, 14, 12, 21, 27] cannot be rephrased as proofs of $\rightsquigarrow$C-completeness. As a consequence, our classes fail to include previous results about non compilable-problems.

Consider, for instance, the problem 3CNF CLAUSE INFERENCE (CI) defined before. The result presented in Section 1.1 stated more formally, is that such a problem belongs to $\rightsquigarrow$P if and only if NP is included in P/poly [13]. This problem belongs to $\rightsquigarrow$coNP (just take f as the identity function) but it is probably not $\rightsquigarrow$coNP-complete, as proven by the following result.

THEOREM 2.5. *If* CI *is* $\rightsquigarrow$coNP*-complete then* P = NP.

Besides CI, this fact holds for many other problems in $\rightsquigarrow$coNP. For instance, with the same technique used in the proof of Theorem 2.5, one can prove that if the problem MINIMAL MODEL CHECKING (defined in Section 3.3) is $\rightsquigarrow$coNP-complete then P = NP. Similar results can be obtained also for other problems defined in Section 3.3, showing that the limitations highlighted in Theorem 2.5 are of a general nature. One choice to overcome this drawback could be to introduce a more powerful form of reduction. However, the reduction used should not violate the assumption of compatibility: otherwise, it may happen that a $\rightsquigarrow$NP-complete problem is also in $\rightsquigarrow$P. Since reductions are used to prove the non-membership of problems to a class, a too powerful reduction may be useless to this end. We are not aware of reductions that are compatible with the classes $\rightsquigarrow$C, and make CI a complete problem.

This suggests that the notion of $\rightsquigarrow$C-hardness is not a completely adequate tool for our purposes. Looking at proofs of non-compilability appearing in the literature gives another, orthogonal, perspective on the inadequacies of the formal tools introduced so far. In fact, non-compilability of CI in [25, 13] is proven by showing that if the problem is compilable, then NP $\subseteq$ P/poly. In short, the proof goes as follows: start from the NP-complete problem 3SAT, and reduce it to CI in such a way that the first input (i.e., x) of CI depends only on the *size* of the instance of 3SAT. This is "almost" a $\leq_{comp}$ reduction, since it can be viewed as a reduction from ϵ3SAT to CI. However, the poly-size functions of a $\leq_{comp}$ reduction are not allowed to use the size of the varying part (in this case, the size of the instance of 3SAT).

Intuitively, in [25, 13] the proofs of non-compilability contain an element that is not present in $\leq_{comp}$ reductions, namely, usage of the size of the varying part in the poly-size functions.

In order to include these results in our general framework we introduce the class of problems *non-uniformly compilable* to a class C, denoted as $\|\!\rightsquigarrow$C. This class generalizes both $\rightsquigarrow$C and C/poly and, in the following, is mainly used to prove non-compilability results.

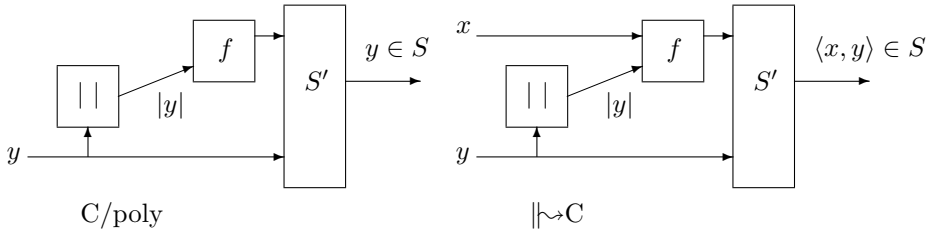

FIG. 3. Comparing C/poly and ‖⇝C

DEFINITION 2.7. [‖⇝C] Let C be a complexity class that conforms to Assumption 2.1. A language of pairs $S \subseteq \Sigma^* \times \Sigma^*$ belongs to ‖⇝C if and only if there exists a binary poly-size function f and a language of pairs $S' \subseteq \Sigma^* \times \Sigma^*$, such that for all $\langle x, y\rangle \in \Sigma^* \times \Sigma^*$ it holds that:

1. $\langle f(x, |y|), y\rangle \in S'$ if and only if $\langle x, y\rangle \in S$;
2. $S' \in$ C.

Notice that now the poly-size function f takes as input both x and the size of y. We remark that the size of y is a number, which we assume to be in unary notation, otherwise, its representation would need just a logarithmic amount of space. These classes are called "non-uniform" because, given the fixed part x of the input, we may obtain two different strings as the result of compilation for two varying parts y_1, y_2 with different sizes. This is similar to the Definition 2.3 of C/poly, in which the oracle has access to the size of the input. An alternative name for these classes could be "generalized advice classes", since the advice given by the function f depends not only on x but also on $|y|$. However, we prefer the name "non-uniform classes", because it makes clear that the way in which the advice is generalized depends on the size of the input. In Figure 3 we compare the diagrams corresponding to C/poly and ‖⇝C.

The class ‖⇝C directly extends C/poly by allowing for a fixed part x. It also generalizes ⇝C by allowing the preprocessing phase to use the size of the varying part y. This aspect makes a difference whenever both x and $|y|$ are known in advance. As we now show, allowing the compilation phase to know only an *upper bound* (k in the following theorem) on the size of the varying part makes no difference.

THEOREM 2.6. *Let C be a complexity class that conforms to Assumption 2.1, and let $S \subseteq \Sigma^* \times \Sigma^*$. A problem S belongs to ‖⇝C if and only if there exists a poly-size function f and a language of pairs S', such that for all $\langle x, y\rangle \in \Sigma^* \times \Sigma^*$ it holds that:*

1. *for all k such that $|y| \leq k$, $\langle f(x, k), y\rangle \in S'$ if and only if $\langle x, y\rangle \in S$;*
2. *$S' \in$ C.*

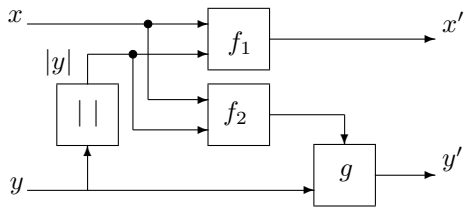

FIG. 4. $\|\!\rightharpoonup$ reductions

Similarly to the compilability classes, we define the *non-uniform polynomial compilability hierarchy* as the set of classes $\|\!\rightharpoonup$C such that C belongs to PH.

In what follows, we prove that each non-uniform class $\|\!\rightharpoonup$C is larger than the corresponding uniform class $\rightharpoonup$C. In order to define useful complete problems for the non-uniform classes, we need a new kind of reduction. Indeed, we can prove that the $\leq_{comp}$ reductions are not useful to this extent. Let us recall the motivation for defining complete problems. We have a problem, which is proved to be in a class, for instance $\|\!\rightharpoonup$NP, such that we can prove neither that is in $\|\!\rightharpoonup$P, nor that it is not. Then, we use the concept of complete problem to prove that the problem is *probably* not in $\|\!\rightharpoonup$P. For instance, CI is in $\|\!\rightharpoonup$coNP, and we did not find a proof that it is in $\|\!\rightharpoonup$P. As a result, we would like to prove that CI is a complete problem for the class $\|\!\rightharpoonup$coNP. However, as the following theorem shows, no problem in coNP can be $\|\!\rightharpoonup$coNP-complete under $\leq_{comp}$ reductions.

THEOREM 2.7. *For no problem S in* coNP *ϵS is $\|\!\rightharpoonup$*coNP*-complete under $\leq_{comp}$ reductions.*

One of the results we want to achieve is indeed to give a framework for showing that, e.g., a problem in coNP cannot be compiled so that the on-line problem belongs to P. However, if no problem in coNP can be $\|\!\rightharpoonup$coNP complete, there is no way to prove the (relative) non-compilability of a problem. Moreover, the above theorem could be generalized to any class of the polynomial hierarchy. As a result, we also need a new kind of reduction in order to define meaningful complete problems for the non-uniform classes of compilability.

DEFINITION 2.8. [$\|\!\rightharpoonup$ reduction] A $\|\!\rightharpoonup$ *reduction* between two languages of pairs A and B is a triple $\langle f_1, f_2, g\rangle$, where f_1 and f_2 are poly-size binary functions and g is a binary polynomial function, such that for any pair of strings $\langle x, y\rangle$ it holds that

$$\langle x, y\rangle \in A \quad \text{if and only if} \quad \langle f_1(x, |y|), g(f_2(x, |y|), y)\rangle \in B$$

The $\|\!\rightharpoonup$ reductions can be represented as in Figure 4. Given two problems A and B, we say that A is *non-uniformly-comp-reducible* to B if and only if there exists a $\|\!\rightharpoonup$ reduction between them, that is, $A\|\!\rightharpoonup B$.

These reductions satisfy Properties 2.1-2.2 listed above.

THEOREM 2.8. *Let C be a complexity class that conforms to Assumption 2.1. The reductions $\|\rightsquigarrow$ satisfy transitivity and compatibility with respect to the class $\|\rightsquigarrow$C.*

We can now define a notion of *hardness* and *completeness* for $\|\rightsquigarrow$C using the $\|\rightsquigarrow$ reductions.

DEFINITION 2.9. [Hardness and Completeness] Let C be a complexity class that conforms to Assumption 2.1. Let S be a language of pairs. S is $\|\rightsquigarrow$C-*hard* if and only if for all problems $A \in \|\rightsquigarrow$C it holds that $A\|\rightsquigarrow S$. The language S is $\|\rightsquigarrow$C-*complete* if it is in $\|\rightsquigarrow$C and is $\|\rightsquigarrow$C-hard.

Just like in the polynomial hierarchy and the polynomial compilability hierarchy, a problem is usually proved to be hard for a class in the non-uniform polynomial compilability hierarchy by showing that another problem, already known to be hard, can be reduced to it. As a result, for each complexity class C we need some $\|\rightsquigarrow$C-hard problem. The following theorem shows that $\rightsquigarrow$C and $\|\rightsquigarrow$C share some complete problems assuming the usual conditions on the class C.

THEOREM 2.9. *For every complexity class* C *that conforms to Assumption 2.1, if S is* C*-complete (under polynomial many-one reductions) then ϵS is complete (under $\|\rightsquigarrow$ reductions) for the corresponding non-uniform compilability class $\|\rightsquigarrow$C.*

Note, however, that complete problems for the two classes are unlikely to coincide, as shown by the following result, which shows that there is an adequate complexity class characterizing the problem CI (cf. Theorem 2.5).

THEOREM 2.10. CI *is $\|\rightsquigarrow$*coNP*-complete.*

The behavior of our classes with respect to complementation is quite interesting. In fact, for each complexity class C that conforms to Assumption 2.1, it holds that co-($\rightsquigarrow$C) = $\rightsquigarrow$(co-C) and co-($\|\rightsquigarrow$C) = nu$\rightsquigarrow$(co-C). Some more properties of the (non-uniform) compilability classes follow.

THEOREM 2.11. *Let* C *and* C′ *be complexity classes that conform to Assumption 2.1. $\rightsquigarrow$C $\subseteq$ $\rightsquigarrow$C′ if and only if* C $\subseteq$ C′*.*

THEOREM 2.12. *Let* C *and* C′ *be complexity classes that conform to Assumption 2.1. $\|\rightsquigarrow$C $\subseteq$ $\|\rightsquigarrow$C′ if and only if* C/poly $\subseteq$ C′/poly*.*

Theorem 2.12 implies (cf. [40]) that if $\|\rightsquigarrow\Sigma_i^p = \|\rightsquigarrow\Sigma_{i+1}^p$ then $\Sigma_{i+2}^p = \Sigma_{i+3}^p$. For example, since CI is $\|\rightsquigarrow$coNP-complete, if it belongs to $\rightsquigarrow$P, then it follows that $\|\rightsquigarrow$coNP= $\|\rightsquigarrow$P. Since $\|\rightsquigarrow$P is closed under complementation, this implies that $\|\rightsquigarrow$P = $\|\rightsquigarrow$NP, and thus $\Sigma_3^p = \Sigma_4^p$ = PH.

THEOREM 2.13. *Let* C *be a uniform and decidable complexity class that conforms to Assumption 2.1. The following relations between classes hold (where* $\subset$ *denotes strict containment):*

- $\bullet$C $\subset \rightsquigarrow$C
- $\bullet$C $\subset$ C/poly
- $\bullet\rightsquigarrow$C $\subset \|\!\rightsquigarrow$C

A slightly weaker separation result is obtained for the classes C/poly and $\|\!\rightsquigarrow$C.

THEOREM 2.14. *Let* C *be a complexity class that conforms to Assumption 2.1. If there exists at least one language* $L \notin$ C/poly*, then* C/poly $\subset \|\!\rightsquigarrow$C *holds.*

As a corollary of the above theorem, since by Muller's theorem [4, Th. 2.4] there exists a language L which is not in P/poly, it holds that P/poly $\subset \|\!\rightsquigarrow$P. Another corollary of Theorem 2.14 is the following.

THEOREM 2.15. *If the polynomial hierarchy is proper, then for each class* C *in the polynomial hierarchy it holds that* C/poly $\subset \|\!\rightsquigarrow$C*.*

The relations given by Theorems 2.11, 2.12, 2.13, 2.14, and 2.15 are summarized in Figure 5. As for the relationship between the classes of the non-uniform polynomial hierarchy and the classes of the compilability hierarchy, that is, the classes C/poly and $\rightsquigarrow$C, we can prove the following theorem.

THEOREM 2.16. *Let* C *be a class in the polynomial hierarchy. It holds that* C/poly $\not\subseteq \rightsquigarrow$C*. Moreover, if there exist problems that are not in* C/poly*, it also holds that* $\rightsquigarrow$C $\not\subseteq$ C/poly*.*

We close the section by giving a summary of the complexity classes that we defined. $\rightsquigarrow$P captures the idea of "compilable problem". In general, a problem is in $\rightsquigarrow$C if, after adequate preprocessing of its fixed part, solving it on-line is a problem in C. $\|\!\rightsquigarrow$NP-hard problems (under $\|\!\rightsquigarrow$ reductions) are what we call "non-compilable", as from Theorem 2.12 we know that if there exists a preprocessing of their fixed part that makes them on-line solvable in polynomial time, then NP/poly is included in P/poly. The same holds for $\|\!\rightsquigarrow$coNP-hard problems. In general, a problem which is $\|\!\rightsquigarrow$C-complete for a class C containing P can be regarded as the "toughest" problem in $\|\!\rightsquigarrow$C, even after arbitrary preprocessing of the fixed part. As for $\rightsquigarrow$NP- and $\rightsquigarrow$coNP-complete problems (under $\leq_{comp}$ reductions), they are also suggestive of "non-compilability", but, as we saw in Theorem 2.5, the notion of $\rightsquigarrow$C-completeness is not powerful enough to capture prototypically non-compilabile problems such as CI.

3. EXAMPLES OF NON-COMPILABLE PROBLEMS

As mentioned in Section 1, we are concerned with "probably" non-compilable problems. Machinery and definitions set up so far offer a formal and simple way

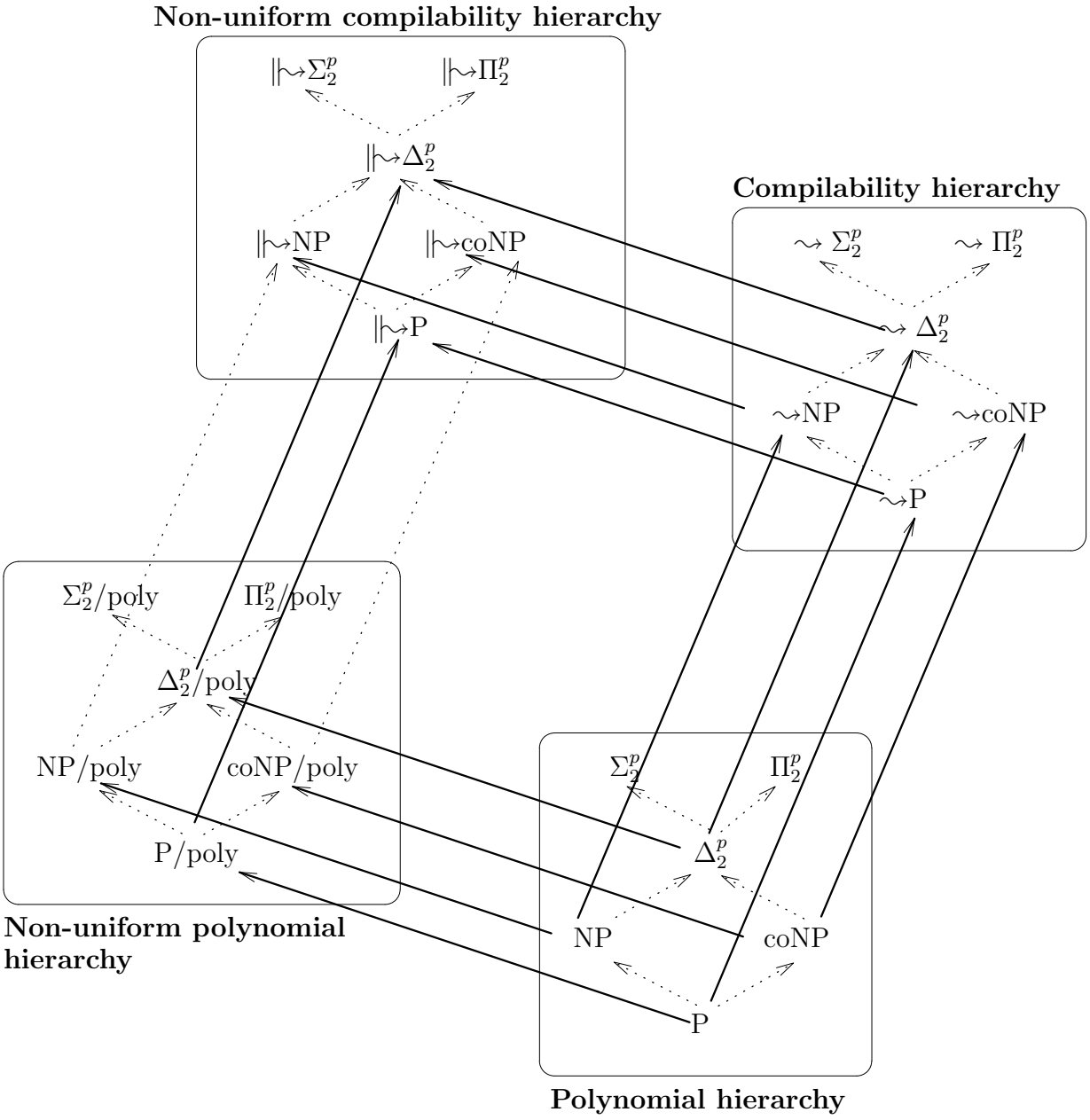

FIG. 5. Relations between complexity and compilability classes.

of proving non-compilability of a problem π: for some complexity class C above P in the polynomial hierarchy, we prove that π is $\|\!\rightsquigarrow$C-complete. Then, if π is in $\rightsquigarrow$P or $\|\!\rightsquigarrow$P, the two upper hierarchies of Figure 5 collapse, which implies that the polynomial hierarchy collapses at some level. This allows us to conclude that it is very unlikely that π is compilable. In this section we prove in this way non-compilability of four problems. Most of the proofs of theorems are included in this section, to highlight the benefits of our framework. Recall that $\|\!\rightsquigarrow$ reductions transform an instance $\langle x, y\rangle$ into an instance $\langle x', y'\rangle$ as follows (see Figure 4):

$$\begin{aligned} x' &= f_1(x, |y|) \\ y' &= g(f_2(x, |y|), y) \end{aligned}$$

Referring to such transformation, the proofs we are going to present are of increasing complexity, in the following sense:

1. In the first proof, only the function g is used, while $f_1(x, n) = x$, (i.e., f_1 just projects its first argument) and $f_2(x, n) = \epsilon$ (i.e., f_2 is constant). Hence $g(f_2(x, n), y) = g(\epsilon, y) = g(y)$ (i.e., g depends on the varying part only). That is:

$$\begin{aligned} x' &= x \\ y' &= g(y) \end{aligned}$$

2. In the second proof, $f_1(x, n) = f_1(x)$, (i.e., f_1 uses the fixed part only), while again $f_2(x, n) = \epsilon$, i.e., f_2 is constant. In formulae:

$$\begin{aligned} x' &= f_1(x) \\ y' &= g(y) \end{aligned}$$

This is still a simple transformation, in which both x and y are separately mapped into x' and y'. Intuitively, this suggests that the structure of fixed-varying part of the two problems is very similar.

3. In the third proof, all three functions are used, but none depends on the size of the varying part $|y|$.

$$\begin{aligned} x' &= f_1(x) \\ y' &= g(f_2(x), y) \end{aligned}$$

Observe that this transformation uses all of the machinery of a $\leq_{comp}$ reduction (hence it is still simpler than the general case).

4. The fourth proof is a true $\|\!\rightsquigarrow$ reduction. We start from a problem of the form ϵS. Hence, the fixed part x is the constant ϵ, and both f_1 and f_2 use only the size of the varying part:

$$\begin{aligned} x' &= f_1(|y|) \\ y' &= g(f_2(|y|), y) \end{aligned}$$

3.1. Constrained Satisfiability (C-SAT)

This problem is as SAT, but with on-line constraints. Let x be a 3CNF propositional formula and y be a partial truth assignment to variables of x. C-SAT is the problem of deciding if y can be extended to a complete truth assignment satisfying x. We consider x being given off-line, while y is given on-line.

$$\text{C-SAT} = \{\langle x, y\rangle \mid y \text{ can be extended to a truth assignment satisfying } x\}$$

PROPOSITION 3.1. C-SAT *is* $\|\!\rightharpoonup$NP*-complete.*

Proof. Membership in the class $\|\!\rightharpoonup$NP is obvious. For the hardness, we reduce to C-SAT the complement of CI, which is $\|\!\rightharpoonup$coNP-complete. We prove this by giving a simple $\leq_{comp}$ reduction (i.e., we do not use the size of the varying part). We have to prove that there exist two poly-size functions f_1 and f_2, and a polynomial-time function g such that, for any pairs $\langle x, y\rangle$, it holds that

$$\langle x, y\rangle \notin \text{CI} \text{ if and only if } \langle f_1(x), g(f_2(x), y)\rangle \in \text{C-SAT}$$

(note that here $\langle x, y\rangle$ denotes the generic instance of CI to be solved using a reduction to C-SAT).

The functions that compose the reduction are defined as follows. The function f_1 is the identity, that is, $f_1(x) = x$. The function f_2 is constant: $f_2(x) = \epsilon$. Finally g builds the partial truth assignment induced by $\neg y$, that is, $g(\epsilon, y)$ is the (partial) truth assignment that maps each letter a into true if a appears as $\neg a$ in the clause y, and false if it appears positively.

As a whole, given a formula x and a clause y, we have that $\langle f_1(x), g(f_2(x), y)\rangle$ is the pair $\langle x, z\rangle$, where z is the partial truth assignment defined from y as above. Thus, $x \not\models y$ if and only if z can be extended to form a truth assignment satisfying x.

The functions f_1, f_2 and g constitute a $\leq_{comp}$ reduction and, therefore, also a $\|\!\rightharpoonup$ reduction. Hence, C-SAT is $\|\!\rightharpoonup$NP-complete. □

The significance of Theorem 3.1 in AI is evident when x is a knowledge base, and y is some information about single literals that can change from time to time. Usually, x is much larger than y. Nevertheless, the theorem says that no preprocessing of x can speed-up the on-line check of satisfiability, given some y. Observe that C-SAT is actually the complement of CI, hence the fact that it is complete for the complemented class of CI is quite natural, and confirms our intuitions about our theory of compilability.

3.2. Constrained Vertex Cover (C-VC)

Let $G = \langle V, E\rangle$ be a graph, k be an integer and V' be a subset of V. C-VC is the problem of deciding whether there exists a vertex cover of G including V' of cardinality less than or equal to k. We assume G and k are off-line and V' is on-line. More formally,

$$\text{C-VC} = \{\langle (G, k), V'\rangle \mid \text{there is a vertex cover of } G \text{ of cardinality} \leq k \text{ including } V'\}$$

PROPOSITION 3.2. C-VC *is* $\|\!\rightsquigarrow$NP*-complete.*

Proof. The polynomial many-one reduction from 3SAT to VERTEX COVER in [19, pg.55] can be easily adapted to obtain a $\leq_{comp}$ reduction from C-SAT to C-VC. Note that we just need a $\leq_{comp}$ reduction, i.e., we do not use the size of the varying part.

Let $\langle x, y\rangle$ be an instance of C-SAT over the set of propositional letters $U = \{u_1, \ldots, u_n\}$. We denote with $U' \subseteq U$ the set of letters to which y assigns a value, i.e., $y : U' \rightarrow \{\mathsf{true}, \mathsf{false}\}$. Given x, the function f_1 constructs the graph G_x as in [19, pg.55]: for each propositional atom $u \in U$, there are in G_x two nodes $u, \bar{u}$ and an edge between them. For each clause $c_i \in x$, there is a clique of three nodes $a_1[i], a_2[i], a_3[i]$ in G_x. Then for each clause $c_i = b_1 \vee b_2 \vee b_3$ (where for $j = 1, 2, 3$ each b_j is either u_h or $\bar{u}_h$) there are three edges $(a_1[i], b_1), (a_2[i], b_2), (a_3[i], b_3)$. Moreover, f_1 counts the number of clauses m and the number of atoms n and sets $k = n + 2m$.

We define $f_2(x) = \epsilon$ for every x – that is, the function f_2 is constant.

The function $g(f_2(x), y) = g(\epsilon, y)$ just sets $V' = U'$. Continuing to adapt the proof in [19, pg.55] , it can be shown that for each $\langle x, y\rangle$, $\langle x, y\rangle \in$C-SAT if and only if the graph G_x has a vertex cover including V' of size less or equal to k.

Since a $\leq_{comp}$ reduction is also a $\|\!\rightsquigarrow$ reduction, this proves that C-VC is $\|\!\rightsquigarrow$NP-hard. To prove membership, it is sufficient to observe that – without any preprocessing – C-VC already belongs to NP. □

In the above cases we were able to show $\|\!\rightsquigarrow$NP-completeness by means of a simple usage of the $\leq_{comp}$ reduction, since the function f_2 is constant. A more complex $\leq_{comp}$ reduction – using f_2 this time – is used for the next result.

3.3. Minimal Model Checking (MMC)

This problem is also known as model checking in circumscription [30, 7]. Let x be any propositional formula. The *minimal* models of x are those truth assignments which satisfy x, and have as few positive values as possible (with respect to set containment). MMC is the problem of deciding whether a given truth assignment y is a minimal model of x. We consider the formula x as given off-line, and the truth assignment y as given on-line.

$$\text{MMC} = \{\langle x, y\rangle \mid y \text{ is a minimal model of } x\ \}$$

THEOREM 3.1. MMC *is* $\|\!\rightsquigarrow$coNP*-complete.*

Proof. Membership in the class $\|\!\rightsquigarrow$coNP is immediate, since MMC is already a coNP-complete problem [7] with no compilation at all.

Hardness is proved by showing that CI $\leq_{comp}$ MMC. Again, we do not use the size of the varying part in this reduction, so we use a $\leq_{comp}$ reduction. Let $\langle x, y\rangle$ be an instance of CI. Define the function f_1 as follows: given x, first f_1 lists all variables of x; let $a_1, \ldots, a_n$ be these variables. Then f_1 constructs a new formula x' using the set of $5n+1$ variables $\{a_1, \ldots, a_n, b_1, \ldots, b_n, c_1, \ldots, c_n, d_1, \ldots, d_n, e_1, \ldots, e_n, p\}$.

The formula x' is defined as

$$x' = (x \wedge \bigwedge_{i=1}^{n}((b_i \rightarrow a_i) \wedge (c_i \rightarrow \neg a_i)) \vee (p \wedge \bigwedge_{i=1}^{n} a_i)) \wedge \bigwedge_{i=1}^{n}((b_i \neq d_i) \wedge (c_i \neq e_i))$$

The function f_2 passes to g the list of all $5n+1$ variables of x'.

The function g takes also the clause y and outputs a truth assignment y', which we denote as v, defined as follows: $v(p) = \mathsf{true}$, and for $i \in 1..n$

- $v(a_i) = \mathsf{true}$,
- $v(b_i) = \mathsf{true}$ if and only if a_i occurs negatively in y, $v(b_i) = \mathsf{false}$ otherwise,
- $v(c_i) = \mathsf{true}$ if and only if a_i occurs positively in y, $v(c_i) = \mathsf{false}$ otherwise,
- $v(d_i) = \neg v(b_i)$, and $v(e_i) = \neg v(c_i)$.

We now prove that $\langle x, y\rangle \in$ CI if and only if $\langle x', y'\rangle \in$ MMC, that is, $x \models y$ if and only if y' is a minimal model of x'. Consider the restriction v' of v to $\{b_1, \ldots, b_n, c_1, \ldots, c_n, d_1, \ldots, d_n, e_1, \ldots, e_n\}$, and substitute values of v' in x', simplifying whenever possible. Observe that all exclusive-or's $\neq$ simplify to true, and that each implication $(b_i \rightarrow a_i) \wedge (c_i \rightarrow \neg a_i)$ simplifies to:

- a_i if a_i occurs negatively in y,
- $\neg a_i$ if a_i occurs positively in y,
- true if a_i does not occur in y.

Therefore, x' is equivalent to $(x \wedge \neg y) \vee (p \wedge \bigwedge_{i=1}^{n} a_i)$ under the partial assignment v'.

Only-if part. Let $x \models y$. Suppose v is not a minimal model of x', i.e., there exists another model u assigning false to a variable which is assigned true by v, and no variable assigned false in v is assigned true by u. Observe that u and v must agree on $\{b_1, \ldots, b_n, c_1, \ldots, c_n, d_1, \ldots, d_n, e_1, \ldots, e_n\}$, because of the exclusive-or's, hence the restriction of u to $\{b_1, \ldots, b_n, c_1, \ldots, c_n, d_1, \ldots, d_n, e_1, \ldots, e_n\}$ is the same as v', namely, v'. Then, u must assign false to a variable in $\{p, a_1, \ldots, a_n\}$. Now observe that $x \models y$ implies that $x \wedge \neg y$ is unsatisfiable, i.e., $x \wedge \neg y$ is equivalent to false. In this case, x' under v' further simplifies to $(p \wedge \bigwedge_{i=1}^{n} a_i)$. Clearly, the only way of satisfying this formula is to assign true to $\{p, a_1, \ldots, a_n\}$, as v does. Therefore, there is no such a u, and v is a minimal model of x'.

If part. Let $x \not\models y$. Then $x \wedge \neg y$ is satisfiable; let t be a model of $x \wedge \neg y$, which is a truth assignment to $\{a_1, \ldots, a_n\}$. Let u be the complete truth assignment obtained by merging t, v', and $u(p) = \mathsf{false}$. Clearly, u satisfies x'. Moreover, variable p is assigned true by v and false by u, and no variable assigned false by v is assigned true by u. Hence, v is not a minimal model of x'. □

The significance of the above theorem appears when considering how minimal models are used in AI: for instance, x can be the logical representation of an apparatus, and a minimal model y is one-one with a minimal diagnosis of a malfunctioning in Reiter's approach to diagnosis [34], or a minimal model is one-one with a minimal change of a belief base x [29]. The theorem says that there is no

"smart" preprocessing of the knowledge base x that can change the complexity class of minimal model checking.

3.4. Clause Minimal Inference (CMI)

Finally, we present the problem (CMI), which is at the second level of our hierarchy of non-compilable problems. It is defined as

$$\text{CMI} = \{\langle x, y\rangle \mid y \text{ is a clause true in all minimal models of } x\}$$

Rephrasing results from [14], we can prove the following.

THEOREM 3.2. CMI *is* $\|\!\sim\Pi_2^p$*-complete.*

The proof (see appendix) makes use of the size of the varying part.

Since inference in circumscription is in Π_2^p, some researchers proposed to use reasoning methods apt for NP-complete problems, with some preprocessing. Namely, Gelfond, Przymuszinsky and Przymuszinska [20] proposed to

1. compute so-called "free-for-negation" formulae, and add them to the knowledge base x;
2. use standard propositional theorem proving for the augmented x.

Instead, Nerode et al. [33] proposed to

1. compute a "compact" representation of minimal models of the knowledge base x
2. use integer programming techniques

While both methods are formally correct, our results shows that the result of the first phase in both approaches, must have size superpolynomial in $|x|$.

4. EXAMPLES OF COMPILABLE PROBLEMS

We already know by Theorem 2.1 that all problems such that (a) for each instance $\langle x, y\rangle$ the size of y is bounded by a polynomial in the size of x and, (b) for each x, there are only polynomially-many distinct y's, are in $\rightsquigarrow$P. As a consequence the problem k-CI, which is the version of CI where the size of the clause is bounded by a constant k, is in $\rightsquigarrow$P.

There are problems whose varying part is not bounded by a polynomial, but which can be easily "translated" into problems obeying to such a constraint. In the following Section 4.1 we present an example.

Of course, the notion of compilability is more interesting for problems whose varying part is not bounded by any polynomial, and such that there is no immediate translation into a problem of this kind. In Section 4.2 we show a problem that belongs to $\rightsquigarrow$P. Another example of a problem in $\rightsquigarrow$P is shown in Section 4.3.

Finally, in Section 4.4 we show a problem for which compilation can decrease the complexity, without making it tractable.

4.1. Conjunction Inference (CONJ-I)

In the problem CONJUNCTION INFERENCE, defined as

$$\text{CONJ-I} = \{\langle x, y\rangle \mid \begin{array}{l} x \text{ is a propositional formula,} \\ y \text{ is a conjunction of literals, and } x \models y\}\end{array}$$

the varying part is not bounded by a polynomial, as there are exponentially many conjunctions of literals. Nevertheless, the answer to a conjunctive query can be easily built using the answers to atomic queries, as $x \models y_1 \wedge \cdots \wedge y_k$ if and only if $x \models y_1$, and ..., and $x \models y_k$. The problem of answering single-literal queries has clearly a varying part which is bounded by a polynomial.

PROPOSITION 4.1. CONJ-I *is in* $\rightsquigarrow$P.

4.2. Cycle in a Hamiltonian Reduction (CHR)

Given a graph $G = \langle V, E\rangle$, an edge which does not occur in any of its Hamiltonian cycles is called *H-irrelevant*. The subgraph $\langle V, E'\rangle$ of G ($E' \subseteq E$) which contains no H-irrelevant edges is called the *Hamiltonian Reduction* of G, and is denoted as $HR(G)$. Given a graph G and a subset S of its nodes, deciding whether there is a cycle in $HR(G)$ that uses a subset of the nodes in S is an NP-hard problem: take $S = V$ (the whole set of nodes in G); if G has no Hamiltonian cycle then $HR(G)$ has no edges ($E' = \emptyset$, since every edge in G is H-irrelevant); otherwise, if G has at least one Hamiltonian cycle then there is a cycle in $HR(G)$ involving all nodes of S.

Nevertheless, the problem CYCLE IN A HAMILTONIAN REDUCTION, defined as:

$$\text{CHR} = \{\langle G, S\rangle \mid \text{ there is a cycle in } HR(G) \text{ that uses a subset of the nodes in } S \ \}$$

belongs to $\rightsquigarrow$P: computing $HR(G)$ can be done off-line using polynomial space, and then checking whether there is a cycle that uses a subset of the nodes in S is clearly polynomial.

While this problem is rather artificial, we believe that it can be seen as an instance of a general schema for generating compilable problems that are intractable without preprocessing.

PROPOSITION 4.2. CHR *is in* $\rightsquigarrow$P.

Note that the varying part of CHR is not bounded by a polynomial, as there are exponentially many possible subsets of nodes. Also, asking for the existence of cycles of fixed length does not help.

4.3. Generalized closure Model Checking (GMC)

Given a propositional formula T, its *generalized closure* [31] $GCWA(T)$ is defined as $T \cup \{\neg p \mid p$ is a letter which is false in all minimal models of $T\}$. Given a formula T and an interpretation M of its propositional letters, deciding whether M is a model of $GCWA(T)$ is a coNP-hard problem: given any formula F on alphabet $A = \{a_1, \ldots, a_n\}$, and another atom $u \notin A$, define $T = (F \wedge \neg u) \vee (u \wedge a_1 \wedge \cdots \wedge a_n)$. Let $M = \{u\} \cup A$. It holds that F is satisfiable if and only if $M \not\models GCWA(T)$. Note that $M \not\models GCWA(T)$ is equivalent to M not being a minimal model of T. Therefore, this reduction also shows that MMC (cf. Section 3.3), when both x and y are given on-line, is coNP-hard.

Again, the problem GENERALIZED CLOSURE MODEL CHECKING, defined as:

$$\text{GMC} = \{\langle x, y\rangle \mid y \models GCWA(x)\ \}$$

belongs to $\rightsquigarrow$P: computing $GCWA(x)$ can be done off-line using $O(|x|)$ space, and then checking whether $y \models GCWA(x)$ is a polynomial-time problem.

PROPOSITION 4.3. GMC *is in* $\rightsquigarrow$P.

We remark that the varying part of GMC is not bounded by a polynomial, as there are exponentially many truth assignments to letters. Moreover, the two problems MMC and GMC, where both inputs are given on-line, are proven to be coNP-hard by means of the same many-one polynomial-time reduction, and still GMC is compilable while MMC is not.

4.4. Generalized closure Inference (GI)

GCWA gives us the opportunity to show a $\rightsquigarrow$coNP problem. Given a formula T and a clause γ, to know whether $GCWA(T) \models \gamma$ is a Π_2^p-hard problem [18]. Nevertheless, the problem GENERALIZED CLOSURE INFERENCE, defined as:

$$\text{GI} = \{\langle T, \gamma\rangle \mid GCWA(T) \models \gamma\ \}$$

belongs to $\rightsquigarrow$coNP, as checking whether $GCWA(T) \models \gamma$ —after $GCWA(T)$ has been computed off-line— is in coNP.

PROPOSITION 4.4. GI *is in* $\rightsquigarrow$coNP.

Actually, it is easy to show that GI is $\|\!\rightsquigarrow$coNP-complete, i.e., it has the same "compilability degree" as CI. This confirms that our compilability classes account for the complexity of a problem *after* preprocessing.

5. RELATED WORK

In this section we compare our definitions of compilability with the idea of *fixed parameter tractability*. The framework for fixed parameter tractability has been introduced and analyzed in many papers [2, 1, 15, 16]. Here we repeat the definitions as introduced in [17]. We show similarities and differences between the two ideas, from both the conceptual and the technical point of view.

As in the approach proposed in this paper, in [17] the concern is on problems with two inputs. A language $L \subseteq \Sigma^* \times \Sigma^*$ is called a *parameterized language*, and if $\langle x, k\rangle \in L$, k is called a *parameter*. Usually, the parameter is a positive integer, but it might be a graph or a formula. Several decision problems can be meaningfully modeled as parameterized languages; as an example, in the parameterized version of the VERTEX COVER problem, the input is a graph G, the parameter is a positive integer k, and the question is whether G has a vertex cover of size $\leq k$. Although the VERTEX COVER problem is NP-hard when k is not fixed, it can be solved by an algorithm which runs in time $O(2^k|G|)$ for each k. The VERTEX COVER problem is therefore polynomial if the parameter k is fixed, as captured by the following definition.

DEFINITION 5.1. **(Uniform fixed parameter tractability, Definition 2.4(i) of [17])** Let A be a parameterized problem. A is *uniformly fixed parameter tractable* ($A \in$ uFPT) if there is an algorithm Φ, a constant c, and an arbitrary function $f : \mathbf{N} \longrightarrow \mathbf{N}$ such that

(a) the running time of $\Phi(\langle x, k\rangle)$ is at most $f(k)|x|^c$, and
(b) $\langle x, k\rangle \in A$ if and only if $\Phi(\langle x, k\rangle) = 1$.

VERTEX COVER is uniformly fixed parameter tractable [17, pg. 196].

The main similarities between fixed parameter tractability and compilability are the following:

• in both cases the input to a problem is split in two parts, which have different status;
• in both cases the aim is to prove that some NP-hard problems become tractable if one part of the input is treated in some special way.

The former similarity immediately raises a question: how do the inputs relate to each other in the two approaches? We envisage two possible answers, that we present considering the VERTEX COVER problem.

1. Map k (the parameter) into the fixed part (x) of the input and the graph G into the varying one (y).
2. Map k (the parameter) into the varying part (y) of the input and the graph G into the fixed one (x).

The second choice makes sense when the graph is known in advance and the size of the covering is known on-line. From now on, we follow the second choice.

The main conceptual differences between fixed parameter tractability and compilability are the following:

• in fixed parameter tractability "the main idea is to study languages that are tractable by the slice", and the focus is on the cost of an algorithm when a parameter of the input is "fixed";
• in compilability, the main idea is "to investigate the idea of processing off-line part of the input data", the only bound being that the compiled data structure does not have exponential size.

In the rest of the section, we focus on formal differences between the two approaches, showing the two properties below.

Property 1: uFPT $\not\subseteq \rightsquigarrow$P (unless the polynomial hierarchy collapses),
Property 2: $\rightsquigarrow$P $\not\subseteq$ uFPT (unconditionally).

To prove Property 1, let us consider the CI problem (3CNF CLAUSE INFERENCE, cf. Section 2), which we recall here for convenience:

$$\text{CI} = \{\langle x, y\rangle \mid x \text{ is a 3CNF formula, } y \text{ is a clause and } x \models y\}.$$

CI is probably not compilable, since it belongs to $\rightsquigarrow$P if and only if NP is included in P/poly (cf. Theorem 2.10).

Nevertheless, CI can be easily shown to belong to uFPT (we assume that the off-line part x is treated as a fixed parameter). Given x and y, it is sufficient to generate all the models of x, and for each such model, verify whether it implies y. The running time of this procedure is $O(2^{|x|} \cdot |y|)$, and is thus uniformly fixed parameter tractable. Such an algorithm is exponential only in the size of the fixed part, but it needs to access the varying part. Obviously, this procedure does not prove that CI is compilable to P.

The following remarks are in order:

- As for syntax, in the approach of [17] the parameter is the second argument of $\langle x, k\rangle$, while in our approach the fixed part, which is conceptually analogous, is the first argument.
- In our approach, the fixed part is a string, not an integer. However, we can formally overcome this problem and prove that uFPT $\subseteq \rightsquigarrow$P implies the collapse of the polynomial hierarchy also when the fixed part is an integer. The proof (omitted here) can be found in [11].

To prove Property 2 we consider the halting problem defined as follows:

$$\{\langle x, y\rangle \mid x \text{ represents an always-terminating Turing machine } \}$$

This problem is clearly compilable to P, since the decision of the termination can be done off-line. On the other hand, there is no algorithm that can decide the termination of a Turing machine, thus it is not uniformly fixed parameter tractable.

Let us discuss our choice of allowing the solution of undecidable problems in the preprocessing phase. In this way we are just strengthening non-compilability results such as $\|\!\rightsquigarrow$coNP-hardness of CI: any problem that is, e.g., $\|\!\rightsquigarrow$coNP-hard is not compilable *even if we can solve arbitrarily hard problems in the preprocessing phase*. If we were to limit such power, e.g., by giving an alternative definition of $\rightsquigarrow$P (let's call it $\rightsquigarrow$'P) as the class of problems that can be solved in polynomial time by preprocessing the fixed part with a terminating procedure, the corresponding reduction (to be compatible with the classes of this new hierarchy) must have f_1 and f_2 decidable too. As a result, each reduction of this kind is also a $\|\!\rightsquigarrow$ reduction, but not the other way around. This implies that there are problems that can be proven to be not compilable using $\|\!\rightsquigarrow$ reductions and hardness, but can not if decidability is imposed. Thus, limiting the preprocessing phase to decidable problems does not give any advantage in proving that a problem is not compilable.

As for the properties of the alternative definition of compilation, it is interesting to note that $\rightsquigarrow$'P $\subset$ uFPT: for a single instance of the problem, the total time for the preprocessing plus the time for the on-line computation is $h(|x|) + g(|x| + |y|)$, where h is a generic function, and g is a polynomial. This is a case of fixed parameter tractability, since the non-polynomial part of this function depends only on the size of the fixed part. On the other hand, the classes of compilability characterize the complexity of problems where many instances sharing the same fixed part, or the fixed part is known in advance.

6. SUMMARY AND DISCUSSION

We applied theoretical computer science techniques to analyze intractable problems where part of the input—the off-line part—can be preprocessed. The sole condition we imposed on the preprocessing is that its output must have polynomial size with respect to the preprocessed input data.

In particular, preprocessing could be even non-recursive, and this is a very strong condition. In future work we will consider a model in which preprocessing is done by a recursive function. Some preliminary work is reported in [28]. As an example, if we modify Definition 2.4 by requiring the function f to be in PSPACE, then all problems listed in Section 4 would still be compilable. Moreover, all relations of Figure 5 would still be valid, with the exception of the inclusion relations between the polynomial hierarchy and the compilability hierarchy, which would be strict unless P=PSPACE. We note that imposing such a constraint on f confines the applicability of the new definitions to problems in PSPACE. For example, if PSPACE $\neq$ EXPTIME, then EXPTIME-complete problems arising in Knowledge Representation or Databases such as reasoning in expressive description logics [6] could not benefit from preprocessing.

We defined suitable computational models with a preprocessing phase, followed by a (ordinary) computation taking as inputs both the result of preprocessing and some other (on-line) input.

We proposed two new classes of problems, namely compilable and non-uniformly compilable problems, and generalized this notions to hierarchies which are analogous (and related) to the polynomial hierarchy and its non-uniform version.

Our proposal systematizes many proofs of the impossibility of representing a formula in logic $\mathcal{L}_1$ as a formula of polynomial size in logic $\mathcal{L}_2$ (e.g., [14, 21]). These proofs can be now rephrased in terms of membership/completeness in a specific class of the hierarchies.

Moreover, our proofs of the impossibility of having a preprocessing with a polynomial-size output provide worst cases for the space needed for compiling Artificial Intelligence problems.

The formal tools we built have been used in [8] for investigating the *space efficiency* of a propositional Knowledge Representation formalism. Informally, the space efficiency of a formalism F in representing a certain piece of knowledge α, is the size of the shortest formula of F that represents α. Knowledge can be either a set of propositional interpretations or a set of formulae (theorems). Using such tools, we were able to show that space efficiency is not always related to time complexity. As an example, while theorem proving for WIDTIO and circumscription (two Knowledge Representation formalisms) have the same time complexity, circumscription is a more compact formalism than WIDTIO to represent theorems.

Finally, we compared our classes with the class of fixed parameter tractable problems, and showed that the classes do not coincide.

ACKNOWLEDGMENT

The authors are grateful to Pierluigi Crescenzi, Georg Gottlob, and Riccardo Silvestri for interesting discussions on the topic of the paper. They also thank Pino Italiano and Fabio Massacci for useful comments on an early draft. Mike Fellows sent useful material on fixed parameter tractability. The reviewers provided very helpful comments, that led to a substantial improvement of the paper. All remaining mistakes are of course ours. This work has been supported by ASI

(Italian Space Agency), MURST (Italian Ministry for University and Scientific and Technological Research) and CNR (Italian Research Council).

REFERENCES

1. K. A. Abrahamson, R. G. Downey, and M. F. Fellows. Fixed-parameter tractability and completeness IV: On completeness for W[P] and PSPACE analogues. *Annals of Pure and Applied Logics*, 73:235–276, 1995.
2. K. R. Abrahamson, J. A. Ellis, M. R. Fellows, and M. E. Mata. On the complexity of fixed parameter problems. In *Proceedings of the Thirtieth Annual Symposium on the Foundations of Computer Science (FOCS'89)*, pages 210–215, 1989.
3. R. Ben-Eliyahu and R. Dechter. Default reasoning using classical logic. *Artificial Intelligence*, 84(1–2):113–150, 1996.
4. R. Boppana and M. Sipser. The complexity of finite functions. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume A, chapter 14, pages 757–804. Elsevier Science Publishers (North-Holland), Amsterdam, 1990.
5. A. Borgida, R. J. Brachman, D. W. Etherington, and H. A. Kautz. Vivid knowledge and tractable reasoning: Preliminary report. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence (IJCAI'89)*, pages 1146–1152, 1989.
6. Martin Buchheit, Francesco M. Donini, and Andrea Schaerf. Decidable reasoning in terminological knowledge representation systems. *Journal of Artificial Intelligence Research*, 1:109–138, 1993.
7. M. Cadoli. The complexity of model checking for circumscriptive formulae. *Information Processing Letters*, 44:113–118, 1992.
8. M. Cadoli, F. Donini, P. Liberatore, and M. Schaerf. Comparing space efficiency of propositional knowledge representation formalisms. In *Proceedings of the Fifth International Conference on the Principles of Knowledge Representation and Reasoning (KR'96)*, pages 364–373, 1996.
9. M. Cadoli and F. M. Donini. A survey on knowledge compilation. *AI Communications—The European Journal on Artificial Intelligence*, 10:137–150, 1997.
10. M. Cadoli, F. M. Donini, P. Liberatore, and M. Schaerf. Feasibility and unfeasibility of off-line processing. In *Proceedings of the Fourth Israeli Symposium on Theory of Computing and Systems (ISTCS'96)*, pages 100–109. IEEE Computer Society Press, 1996. URL = `ftp://ftp.dis.uniroma1.it/PUB/AI/papers/cado-etal-96.ps.gz`.
11. M. Cadoli, F. M. Donini, P. Liberatore, and M. Schaerf. Preprocessing of intractable problems. Technical Report DIS 24-97, Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza", November 1997. URL = `http://ftp.dis.uniroma1.it/PUB/AI/papers/cado-etal-97-d-REVISED.ps.gz`.
12. M. Cadoli, F. M. Donini, P. Liberatore, and M. Schaerf. The size of a revised knowledge base. *Artificial Intelligence*, 115(1):25–64, 1999.
13. M. Cadoli, F. M. Donini, and M. Schaerf. Is intractability of non-monotonic reasoning a real drawback? *Artificial Intelligence*, 88(1–2):215–251, 1996.
14. M. Cadoli, F. M. Donini, M. Schaerf, and R. Silvestri. On compact representations of propositional circumscription. *Theoretical Computer Science*, 182:183–202, 1997.
15. R. G. Downey and M. F. Fellows. Fixed-parameter tractability and completeness I: Basic results. *SIAM Journal on Computing*, 24(4):873–921, 1995.
16. R. G. Downey and M. F. Fellows. Fixed-parameter tractability and completeness II: On completeness for W[1]. *Theoretical Computer Science*, 141:109–131, 1995.
17. R. G. Downey and M. F. Fellows. *Parameterized Complexity*. Springer-Verlag, 1999.
18. T. Eiter and G. Gottlob. Propositional circumscription and extended closed world reasoning are Π_2^p-complete. *Theoretical Computer Science*, 114:231–245, 1993.
19. M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, San Francisco, Ca, 1979.
20. M. Gelfond, H. Przymusinska, and T. Przymusinsky. On the relationship between circumscription and negation as failure. *Artificial Intelligence*, 38:49–73, 1989.

21. G. Gogic, H. A. Kautz, C. Papadimitriou, and B. Selman. The comparative linguistics of knowledge representation. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI'95)*, pages 862–869, 1995.

22. G. Gottlob. Complexity and expressive power of KR formalisms. In *Proceedings of the Fifth International Conference on the Principles of Knowledge Representation and Reasoning (KR'96)*, pages 647–649, 1996.

23. D. S. Johnson. A catalog of complexity classes. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume A, chapter 2, pages 67–161. Elsevier Science Publishers (North-Holland), Amsterdam, 1990.

24. R. M. Karp and R. J. Lipton. Some connections between non-uniform and uniform complexity classes. In *Proceedings of the Twelfth ACM Symposium on Theory of Computing (STOC'80)*, pages 302–309, 1980.

25. H. A. Kautz and B. Selman. Forming concepts for fast inference. In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI'92)*, pages 786–793, 1992.

26. H. J. Levesque. Making believers out of computers. *Artificial Intelligence*, 30:81–108, 1986.

27. P. Liberatore. Compact representation of revision of Horn clauses. In Xin Yao, editor, *Proceedings of the Eighth Australian Joint Artificial Intelligence Conference (AI'95)*, pages 347–354. World Scientific, 1995.

28. P. Liberatore. *Compilation of intractable problems and its application to Artificial Intelligence*. PhD thesis, Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza", 1998. URL = `ftp://ftp.dis.uniroma1.it/pub/AI/papers/libe-98-c.ps.gz`.

29. P. Liberatore and M. Schaerf. Relating belief revision and circumscription. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI'95)*, pages 1557–1563, 1995.

30. J. McCarthy. Circumscription - A form of non-monotonic reasoning. *Artificial Intelligence*, 13:27–39, 1980.

31. J. Minker. On indefinite databases and the closed world assumption. In *Proceedings of the Sixth International Conference on Automated Deduction (CADE'82)*, pages 292–308, 1982.

32. Y. Moses and M. Tennenholtz. Off-line reasoning for on-line efficiency: knowledge bases. *Artificial Intelligence*, 83:229–239, 1996.

33. A. Nerode, R. T. Ng, and V. S. Subrahmanian. Computing circumscriptive databases. I: Theory and algorithms. *Information and Computation*, 116:58–80, 1995.

34. R. Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32:57–95, 1987.

35. R. Schrag and J. Crawford. Implicates and prime implicates in random 3SAT. *Artificial Intelligence*, 81:199–222, 1996.

36. B. Selman and H. A. Kautz. Knowledge compilation using Horn approximations. In *Proceedings of the Ninth National Conference on Artificial Intelligence (AAAI'91)*, pages 904–909, 1991.

37. B. Selman and H. A. Kautz. Knowledge compilation and theory approximation. *Journal of the ACM*, 43:193–224, 1996.

38. L. J. Stockmeyer. The polynomial-time hierarchy. *Theoretical Computer Science*, 3:1–22, 1976.

39. R. E. Tarjan. Amortized computational complexity. *SIAM Journal on Algebraic and Discrete Methods*, 6:306–318, 1985.

40. C. K. Yap. Some consequences of non-uniform conditions on uniform classes. *Theoretical Computer Science*, 26:287–300, 1983.

APPENDIX: PROOFS

Theorem 2.1 *Let S be a language of pairs and $W(S,x) = \{y \mid \langle x,y\rangle \in S\}$. If there exist two polynomials p_1, p_2 such that for every x it holds that $||W(S,x)|| \leq p_1(|x|)$ and for every $\langle x,y\rangle \in S$ it holds that $|y| \leq p_2(|x|)$, then $S \in \rightsquigarrow$P.*

Proof. Given off-line x, we simply build a table, in which we record the solution to the decision problem $\langle x,y\rangle$ for each y. Thus on-line we read y and access such a

table. This access is obviously feasible in polynomial time with respect to the size of the table. Since $W(S,x)$ has size polynomial in $|x|$, this table has polynomial size too.

Formally, given x let $W(S,x) = \{y_1, \ldots, y_m\}$, with m bounded by a polynomial $p_1(|x|)$. Let @ be a new symbol, and

$$\begin{aligned} f(x) &= y_1@\cdots@y_m \\ S' &= \{\langle a_1@\cdots@a_m, y\rangle \mid y = a_i \text{ for some } i\} \end{aligned}$$

Observe that $|f(x)| \leq m \cdot p_2(|x|) \leq p_1(|x|) \cdot p_2(|x|)$, hence f is poly-size. Then, we have

$$\begin{aligned} \langle x,y\rangle \in S &\Leftrightarrow y \in W(S,x) \\ &\Leftrightarrow y = \text{ some element of } f(x) \\ &\Leftrightarrow \langle f(x), y\rangle \in S' \end{aligned}$$

■

Theorem 2.2 *Let* C *be a complexity class that conforms to Assumption 2.1. The* $\leq_{comp}$ *reductions satisfy transitivity and compatibility with respect to* $\rightsquigarrow$C*.*

Proof. We have to prove that the relation $\leq_{comp}$ is transitive and is compatible with the compilable uniform hierarchy, that is, if $A \leq_{comp} B$ and $B \in \rightsquigarrow$C then $A \in \rightsquigarrow$C.

Transitivity Suppose $A \leq_{comp} B$ and $B \leq_{comp} C$. By definition there exist four poly-size functions f_1, f_2, f_1' and f_2' and two polynomial reductions g and g' such that

$$\begin{aligned} \langle x,y\rangle \in A &\Leftrightarrow \langle f_1(x), g(f_2(x), y)\rangle \in B \\ \langle x',y'\rangle \in B &\Leftrightarrow \langle f_1'(x'), g'(f_2'(x'), y')\rangle \in C \end{aligned}$$

Now let

$$\begin{aligned} f_1''(x) &= f_1'(f_1(x)) \\ f_2''(x) &= f_2'(f_1(x))@f_2(x) \\ g''(x,y) &= \begin{cases} g'(r, g(s,y)) & \text{if @ appears exactly once in } x \text{ and } x = r@s \\ \epsilon & \text{otherwise} \end{cases} \end{aligned}$$

where @ is a new symbol. Note that the functions f_1'' and f_2'' are poly-size, and g'' is a polynomial reduction. Then, we have

$$\begin{aligned} \langle x,y\rangle \in A &\Leftrightarrow \langle f_1(x), g(f_2(x), y)\rangle \in B \\ &\Leftrightarrow \langle f_1'(f_1(x)), g'(f_2'(f_1(x)), g(f_2(x), y))\rangle \in C \\ &\Leftrightarrow \langle f_1''(x), g''(f_2'(f_1(x))@f_2(x), y\rangle \in C \\ &\Leftrightarrow \langle f_1''(x), g''(f_2''(x), y)\rangle \in C \end{aligned}$$

Compatibility Let B be a problem in $\rightsquigarrow$C, and $A \leq_{comp} B$. We prove that A is a problem in $\rightsquigarrow$C. By definition, there are three poly-size functions f_1, f_2 and f', a polynomial function g and a problem S' in C such that

$$\begin{aligned} \langle x, y\rangle \in A &\Leftrightarrow \langle f_1(x), g(f_2(x), y)\rangle \in B \\ \langle x', y'\rangle \in B &\Leftrightarrow \langle f'(x'), y'\rangle \in S \end{aligned}$$

Now, let $f''(x) = f'(f_1(x))@f_2(x)$ and S'' be the problem in C characterized by

$$S'' = \{\langle a@b, c\rangle | \langle a, g(b, c)\rangle \in S'\}$$

We have that

$$\begin{aligned} \langle x, y\rangle \in A &\Leftrightarrow \langle f_1(x), g(f_2(x), y)\rangle \in B \\ &\Leftrightarrow \langle f'(f_1(x)), g(f_2(x), y)\rangle \in S \\ &\Leftrightarrow \langle f''(x), y\rangle \in S'' \end{aligned}$$

so A belongs to $\rightsquigarrow$C. □

Theorem 2.3 *For every complexity class* C *if* S *is* C*-complete (under polynomial time many-one reductions) then* ϵS *is complete (under* $\leq_{comp}$ *reductions) for the corresponding compilability class* $\rightsquigarrow$C*.*

Proof. Suppose that $A \in \rightsquigarrow$C. Then there exists a poly-size function f and a problem S' in C such that

$$\langle x, y\rangle \in A \Leftrightarrow \langle f(x), y\rangle \in S'$$

Now, S is a C-complete problem, so given $S' \in$ C, there exists a polynomial function g such that

$$a \in S' \Leftrightarrow g(a) \in S$$

Consider the following functions:

$$\begin{aligned} f_1(x) &= \epsilon \\ f_2(x) &= f(x) \\ g'(a, b) &= g(\langle a, b\rangle) \end{aligned}$$

Using these poly-size functions, A can be reduced to ϵS since

$$\begin{aligned} \langle x, y\rangle \in A &\Leftrightarrow \langle f(x), y\rangle \in S' \\ &\Leftrightarrow g(\langle f(x), y\rangle) \in S \\ &\Leftrightarrow \langle \epsilon, g(\langle f(x), y\rangle)\rangle \in \epsilon S \\ &\Leftrightarrow \langle f_1(x), g'(f_2(x), y)\rangle \in \epsilon S \end{aligned}$$

so $A \leq_{comp} \epsilon S$. □

Theorem 2.4 *The problem* FI *is* $\rightsquigarrow$coNP-complete.

Proof. Since the decision problem FI where both the knowledge base and the query are given on-line is in coNP, then FI is also in $\rightsquigarrow$coNP. By Theorems 2.2 and 2.3 it suffices to show that $\epsilon 3\text{UNSAT} \leq_{comp}$ FI. Given an instance $\langle x, y\rangle$ of $\epsilon 3\text{UNSAT}$, we define two poly-size functions f_1, f_2 and a polynomial function g such that

$$\langle x, y\rangle \in \epsilon 3\text{UNSAT} \quad \Leftrightarrow \quad \langle f_1(x), g(f_2(x), y)\rangle \in \text{FI}$$

Let

$$\begin{aligned} f_1(x) &= \mathsf{true} \\ f_2(x) &= x \\ g(x, y) &= \begin{cases} \mathsf{false} & \text{if } x \neq \epsilon \\ \neg y & \text{otherwise} \end{cases} \end{aligned}$$

We have two cases, either $x = \epsilon$ or not. In the first case, we need to show that $\langle \epsilon, y\rangle \in \epsilon 3\text{UNSAT}$ if and only if $\langle \mathsf{true}, \neg y\rangle \in$ FI. This is equivalent to say that y is unsatisfiable iff $\mathsf{true} \models \neg y$, that trivially holds. In the second case ($x \neq \epsilon$), we show that $\langle x, y\rangle \in \epsilon 3\text{UNSAT}$ if and only if $\langle \mathsf{true}, \mathsf{false}\rangle \in$ FI. Notice that $\langle x, y\rangle \notin \epsilon 3\text{UNSAT}$ when $x \neq \epsilon$ and, at the same time, it is not the case that $\mathsf{true} \models \mathsf{false}$. □

Theorem 2.5 *If* CI *is* $\rightsquigarrow$coNP*-complete then* P = NP.

Proof. Let UNSAT be the complementary problem of SAT. Assume that CI is a $\rightsquigarrow$coNP-complete problem. Then the problem ϵUNSAT is reducible to it, that is, there are two poly-size functions f_1, f_2 and a polynomial function g such that

$$\langle x, y\rangle \in \epsilon\text{UNSAT} \;\Leftrightarrow\; \langle f_1(x), g(f_2(x), y)\rangle \in \text{CI}$$

where $f_1(x)$ is a 3-CNF formula and $g(f_2(x), y)$ is a clause. Now, let F be a propositional formula. We have

$$\begin{aligned} F \text{ unsatisfiable} &\Leftrightarrow \langle \epsilon, F\rangle \in \epsilon\text{UNSAT} \\ &\Leftrightarrow \langle f_1(\epsilon), g(f_2(\epsilon), F)\rangle \in \text{CI} \end{aligned}$$

The condition $\langle f_1(\epsilon), g(f_2(\epsilon), F)\rangle \in$ CI is equivalent to $f_1(\epsilon) \models g(f_2(\epsilon), F)$. Now, $f_1(\epsilon)$ is a constant formula and $g(f_2(\epsilon), F)$ is a clause. Deciding if a clause is implied by a constant formula is a polynomial-time problem. But this solves also the coNP-complete problem of deciding whether F is unsatisfiable or not. Hence, if CI

is a $\rightsquigarrow$coNP-complete problem P=coNP. Since P is closed under complementation P=NP. □

Theorem 2.6 *Let C be a complexity class that conforms to Assumption 2.1, and let $S \subseteq \Sigma^* \times \Sigma^*$. A problem S belongs to $\|\!\rightsquigarrow$C if and only if there exists a poly-size function f and a language of pairs S', such that for all $\langle x, y\rangle \in \Sigma^* \times \Sigma^*$ it holds that:*

1. *for all k such that $|y| \leq k$, $\langle f(x,k), y\rangle \in S'$ if and only if $\langle x, y\rangle \in S$;*
2. *$S' \in$ C.*

Proof.

The "if" direction immediately follows. In fact, by hypothesis, we have that for all k such that $|y| \leq k$, $\langle x, y\rangle \in S$ if and only if $\langle f(x,k), y\rangle \in S'$ which implies (choosing $k = |y|$) that $\langle x, y\rangle \in S$ if and only if $\langle f(x, |y|), y\rangle \in S'$.

Let us now consider the "only if" direction of the proof. Assume that S is a $\|\!\rightsquigarrow$C problem: we prove that there exist f and S' that satisfy the conditions stated above. By definition, since S is in $\|\!\rightsquigarrow$C problem, there exists a poly-size function f' and a C problem S'' such that

$$\langle x, y\rangle \in S \quad \Leftrightarrow \quad \langle f'(x, |y|), y\rangle \in S''$$

Let @ be a new symbol. We define f and S' as follows.

$$\begin{aligned} f(x,k) &= f'(x,0)@\cdots@f'(x,k) \\ S' &= \{\langle a_0@\cdots@a_m, b\rangle \mid \langle a_{|b|}, b\rangle \in S''\} \end{aligned}$$

Determining whether a string is in S' is clearly a problem in C, as it amounts to select a substring of $a_0@\cdots@a_m$ and then determining membership of it in S''. The function f is poly-size: note that being poly-size means that the size of the result should be of polynomial size w.r.t. the size of the first argument x and the size of the second argument when represented in unary notation, and this is satisfied by the function f above, since f' is poly-size.

We prove now that, for each $k \geq |y|$, it holds that $\langle x, y\rangle \in S$ if and only if $\langle f(x,k), y\rangle \in S'$.

$$\begin{aligned} \langle f(x,k), y\rangle \in S' &\Leftrightarrow \langle f'(x,0)@\cdots@f'(x,k), y\rangle \in S' \\ &\Leftrightarrow \langle f'(x, |y|), y\rangle \in S'' \\ &\Leftrightarrow \langle x, y\rangle \in S \end{aligned}$$

This is exactly the condition over f and S' to be proved. □

Theorem 2.7 *For no problem S in* coNP *ϵS is $\|\!\sim$coNP-complete under $\leq_{comp}$ reductions.*

Proof. Let A be a coNP-complete problem, and let HALT be the following (undecidable) language

$$\text{HALT} = \{x \in \Sigma^* \mid |x| \text{ is the Gödel number of a Turing machine that always terminates } \} \tag{A.1}$$

Notice that ϵHALT is in $\|\!\sim$P, since we can compile the size of the on-line part to directly produce the answer of the problem. However, we cannot reduce HALT to A using $\leq_{comp}$ reductions. □

Theorem 2.8 *Let C be a complexity class that conforms to Assumption 2.1. The reductions $\|\!\sim$ satisfy transitivity and compatibility with respect to the class $\|\!\sim$C.*

Proof. We have to prove that the relation $\|\!\sim$ is transitive, and that if $A\|\!\sim B$ and $B \in \|\!\sim$C, then $A \in \|\!\sim$C.

Transitivity

By hypothesis there exist four poly-size functions f_1, f_2, f_1' and f_2', and two polynomial-time functions g and g', such that

$$\langle x, y\rangle \in A \;\Leftrightarrow\; \langle f_1(x, |y|), g(f_2(x, |y|), y)\rangle \in B \tag{A.2}$$

$$\langle x', y'\rangle \in B \;\Leftrightarrow\; \langle f_1'(x', |y'|), g'(f_2'(x', |y'|), y')\rangle \in C \tag{A.3}$$

Since g is a polynomial function, we can determine a polynomial p such that $|g(a, b)| \leq p(|a| + |b|)$, for any a, b. Now let @ be a new symbol, and let

$$\begin{aligned} f_1''(x, |y|) &= f_1'(f_1(x, |y|), p(|f_2(x, |y|)| + |y|) \\ f_2''(x, |y|) &= f_2'(f_1(x, |y|), p(|f_2(x, |y|)| + |y|))@f_2(x, |y|) \end{aligned}$$

Let $r(n) = \#\ldots\#$ (a sequence of n blanks) and $k = p(|a| + |b|) - |g(a, b)|$. Notice that, by the definition of p, k is non-negative. Now, let $g_r(a, b) = g(a, b)r(k)$, that is $g(a, b)$ followed by k blanks. The polynomial function g'' is defined as:

$$g''(x, y) = \begin{cases} g'(r, g_r(s, y)) & \text{if @ appears exactly once in } x \text{ and } x = r@s \\ \epsilon & \text{otherwise} \end{cases}$$

We can now prove the claim.

$$\begin{aligned} \langle x, y\rangle \in A &\Leftrightarrow \langle f_1(x, |y|), g(f_2(x, |y|), y)\rangle \in B \\ &\Leftrightarrow \langle f_1(x, |y|), g_r(f_2(x, |y|), y)\rangle \in B \end{aligned}$$

$$\begin{aligned}
&\Leftrightarrow \langle f_1'(f_1(x,|y|),|g_r(f_2(x,|y|),y)|), g'(f_2'(f_1(x,|y|),|g_r(f_2(x,|y|),y)|),\\
&\quad g_r(f_2(x,|y|),y))\rangle \in C\\
&\Leftrightarrow \langle f_1'(f_1(x,|y|),p(|f_2(x,|y|)|+|y|), g'(f_2'(f_1(x,|y|),p(|f_2(x,|y|)|+|y|)),\\
&\quad g_r(f_2(x,|y|),y))\rangle \in C\\
&\Leftrightarrow \langle f_1''(x,|y|), g''(f_2''(x,|y|),y)\rangle \in C
\end{aligned}$$

An explanation of each one of the above equivalences follows:

1. cf. equivalence (A.2),
2. cf. assumption on the blanks made in Section 2.1,
3. cf. equivalence (A.3),
4. cf. definition of k,
5. cf. definitions of f_1'' and f_2''.

■

Compatibility

Let us suppose $A\|\!\rightsquigarrow B$ and $B \in \|\!\rightsquigarrow$C. We can prove that $A \in \|\!\rightsquigarrow$C. By hypothesis there exist two poly-size functions f_1 and f_2 and a polynomial one g such that

$$\langle x,y\rangle \in A \quad \Leftrightarrow \quad \langle f_1(x,|y|), g(f_2(x,|y|),y)\rangle \in B$$

and there is a poly-size function f and a problem S' in C such that

$$\langle x',y'\rangle \in B \quad \Leftrightarrow \quad \langle f(x',|y'|),y'\rangle \in S'$$

Since g is a polynomial function, we can determine a polynomial p such that $|g(a,b)| \leq p(|a|+|b|)$, for any a, b. Now let @ be a new symbol, and let f' and S'' be

$$\begin{aligned}
f'(x,|y|) &= f(f_1(x,|y|),0)@f(f_1(x,|y|),1)@\cdots@f(f_1(x,|y|),p(|x|+|y|))@f_2(x,|y|)\\
S'' &= \{\langle a_0@\cdots@a_n@b,c\rangle \mid \langle a_i, g(b,c)\rangle \in S' \text{ where } i = |g(b,c)|\}
\end{aligned}$$

In other words, f' concatenates the results of the function f for all values from 0 up to $p(|x|+|y|)$, while the language S'' picks and concatenates some of the first elements of the language S'. Notice that deciding membership in S'' is a problem in C since the complexity of checking membership is polynomially related to the complexity of checking membership in S'.

We have

$$\begin{aligned}
\langle x,y\rangle \in A &\Leftrightarrow \langle f_1(x,|y|), g(f_2(x,|y|),y)\rangle \in B\\
&\Leftrightarrow \langle f(f_1(x,|y|),|g(f_2(x,|y|),y)|), g(f_2(x,|y|),y)\rangle \in S'\\
&\Leftrightarrow \langle f'(x,|y|),y\rangle \in S''
\end{aligned}$$

Theorem 2.9 *For every complexity class* C *that conforms to Assumption 2.1, if* S *is* C*-complete (under polynomial many-one reductions) then* ϵS *is complete (under* $\|\!\rightsquigarrow$ *reductions) for the corresponding non-uniform compilability class* $\|\!\rightsquigarrow$C.

Proof. Let A be a generic $\|\!\rightsquigarrow$C problem. By hypothesis,

$$\langle x, y\rangle \in A \quad \Leftrightarrow \quad \langle f(x, |y|), y\rangle \in S'$$

But $S' \in C$, thus it can be reduced to S using a polynomial function g such that

$$\langle x, y\rangle \in S' \quad \Leftrightarrow \quad g(x, y) \in S$$

Now, let

$$\begin{aligned} f_1(x, |y|) &= \epsilon \\ f_2(x, |y|) &= f(x, |y|) \\ g'(a, b) &= g(a, b) \end{aligned}$$

We have

$$\begin{aligned} \langle x, y\rangle \in A &\Leftrightarrow \langle f(x, |y|), y\rangle \in S' \\ &\Leftrightarrow g(f(x, |y|), y) \in S \\ &\Leftrightarrow \langle \epsilon, g(f(x, |y|), y)\rangle \in \epsilon S \\ &\Leftrightarrow \langle f_1(x, |y|), g(f_2(x, |y|), y)\rangle \in \epsilon S \end{aligned}$$

■

Theorem 2.10 CI *is* $\|\!\rightsquigarrow$coNP*-complete.*

Proof. Let us consider the decision problem obtained from CI by having on-line both the knowledge base and the query. Since this problem is in coNP, CI is in $\|\!\rightsquigarrow$coNP. By Theorems 2.8 and 2.9 it suffices to show that ϵ3UNSAT $\|\!\rightsquigarrow$ CI. Given an instance $\langle \delta, \pi\rangle$ we define two poly-size functions f_1, f_2 and a polynomial function g such that

$$\langle \delta, \pi\rangle \in \epsilon 3\textsc{unsat} \quad \Leftrightarrow \quad \langle f_1(\delta, |\pi|), g(f_2(\delta, |\pi|), \pi)\rangle \in \textsc{ci}$$

Let $n = |\pi|$ and L the alphabet of (at most) n atoms used in π. Notice that each three-literals clause over L can be given a name: we denote with C a set of atoms one-to-one with all possible three-literals clauses of L

$$C = \{c_i \mid \gamma_i \text{ is a three-literals clause of } L\} \tag{A.4}$$

Let

$$f_1(\delta, n) \;=\; \begin{cases} \mathsf{true} & \text{if } \delta \neq \epsilon \\ \bigwedge_{i=1..n} c_i \rightarrow \gamma_i & \text{otherwise} \end{cases}$$

$$f_2(\delta, n) = \delta$$
$$g(\delta, \pi) = \begin{cases} \mathsf{false} & \text{if } \delta \neq \epsilon \\ (\bigvee_{\gamma_i \notin \pi} c_i) \vee (\bigvee_{\gamma_i \in \pi} \neg c_i) & \text{otherwise} \end{cases}$$

We prove that if $T = f_1(\delta, \pi)$ and $q = g(f_2(\delta, \pi), \pi)$, then $\langle \delta, \pi \rangle \in \epsilon\text{UNSAT}$ if and only if $\langle T, q \rangle \in \text{CI}$.

Suppose $\langle \delta, \pi \rangle \in \epsilon\text{UNSAT}$. By definition, $\delta = \epsilon$, hence we have

$$\begin{aligned} T \wedge \neg q &\equiv \{c_i | \gamma_i \in \pi\} \wedge \{\neg c_i | \gamma_i \notin \pi\} \wedge \{c_i \rightarrow \gamma_i\} \\ &\equiv \bigwedge \{c_i | \gamma_i \in \pi\} \wedge \bigwedge \{\neg c_i | \gamma_i \notin \pi\} \wedge \pi \end{aligned}$$

Now, π is unsatisfiable, thus $T \wedge \neg q$ is unsatisfiable too. This implies $T \models q$.

Suppose $\langle \delta, \pi \rangle \notin \epsilon\text{UNSAT}$. This implies that either $\delta \neq \epsilon$ or π is satisfiable. In the first case, we obtain $T \equiv \mathsf{true}$ and $q \equiv \mathsf{false}$, so $\langle T, q \rangle \notin \text{CI}$. In the second case, we have

$$T \wedge \neg q \equiv \bigwedge \{c_i | \gamma_i \in \pi\} \wedge \bigwedge \{\neg c_i | \gamma_i \notin \pi\} \wedge \pi$$

Since π is satisfiable, it follows that $T \wedge \neg q$ has a model M. Since $M \not\models q$, it follows $T \not\models q$. This proves that $\epsilon\text{UNSAT} \not\leadsto \text{CI}$. □

Theorem 2.11 *Let* C *and* C′ *be complexity classes that conform to Assumption 2.1.* $\leadsto$C $\subseteq$ $\leadsto$C′ *if and only if* C $\subseteq$ C′.

Proof. Let us assume C $\subseteq$ C′, and let S be any language in C. Thus, there exist a poly-size function f and a language S' in C such that

$$\langle x, y \rangle \in S \quad \text{iff} \quad \langle f(x), y \rangle \in S'$$

Since C $\subseteq$ C′ it follows $S' \in$ C′. Hence $S \in \leadsto$C′.

Let us assume $\leadsto$C $\subseteq$ $\leadsto$C′. Let $S \subseteq \Sigma^*$ be a complete language in C. We prove that the decision problem for S belongs to C′, thus proving that C $\subseteq$ C′.

By Theorem 2.3, ϵS is in $\leadsto$C, thus it is also in $\leadsto$C′ by hypothesis, thus there exist a poly-size function f and a C′ language S' such that

$$\langle x, y \rangle \in \epsilon S \quad \text{iff} \quad \langle f(x), y \rangle \in S'$$

In order to prove $S \in$ C′, we reduce it to S'. Indeed,

$$y \in S_1 \quad \text{iff} \quad \langle \epsilon, y \rangle \in \epsilon S \quad \text{iff} \quad \langle f(\epsilon), y \rangle \in S'$$

Since $f(\epsilon)$ is a constant, S can be polynomially reduced to S', thus S is in C′. Since S is C-complete and C conforms to Assumption 2.1, any problem in C can be reduced to S, hence we obtain C $\subseteq$ C′. □

Theorem 2.12 *Let* C *and* C′ *be complexity classes that conform to Assumption 2.1.* $\|\!\rightharpoonup$C $\subseteq$ $\|\!\rightharpoonup$C′ *if and only if* C/poly $\subseteq$ C′/poly.

Proof. Let us assume C/poly $\subseteq$ C′/poly, and let S be a C-complete problem. This implies that ϵS is $\|\!\rightharpoonup$C-complete and that $S \in$ C′/poly. Thus, it holds that $\epsilon S \in \|\!\rightharpoonup$C′. It follows that there is a $\|\!\rightharpoonup$C-complete problem that belongs to $\|\!\rightharpoonup$C′. Thus, $\|\!\rightharpoonup$C $\subseteq$ $\|\!\rightharpoonup$C′.

Let now assume $\|\!\rightharpoonup$C $\subseteq$ $\|\!\rightharpoonup$C′. Let S be a generic C/poly problem. We prove that S is also C′/poly. By definition there exist a poly-size function f and a C language S' such that

$$y \in S \quad \text{iff} \quad \langle f(|y|), y\rangle \in S'$$

It follows that the language of pairs ϵS is in $\|\!\rightharpoonup$C. Therefore, it is also in $\|\!\rightharpoonup$C′ and there exist a poly-size function f and a C′ language S'' such that

$$\langle x, y\rangle \in \epsilon S \quad \text{iff} \quad \langle f(x, |y|), y\rangle \in S''$$

As a consequence, we have

$$y \in S \quad \text{iff} \quad \langle \epsilon, y\rangle \in \epsilon S \quad \text{iff} \quad \langle f(\epsilon, |y|), y\rangle \in S''$$

Since $f(\epsilon, |y|)$ is indeed a poly-size function of $|y|$ alone, and S'' is in C′, we conclude that S is in C′/poly. □

Theorem 2.13 *Let* C *be a uniform and decidable complexity class that conforms to Assumption 2.1. The following relations between classes hold (where* $\subset$ *denotes strict containment):*

- C $\subset \rightharpoonup$C
- C $\subset$ C/poly
- $\rightharpoonup$C $\subset \|\!\rightharpoonup$C

Proof. Non-strict containments trivially hold by definition. We must show that they are strict.

Using the language HALT (cf. (A.1) in the proof of Theorem 2.7), we define:

$$\text{HALT}\epsilon = \{\langle x, y\rangle \mid x \in \text{HALT}, y = \epsilon\}$$

Notice that HALTϵ does not belong to any decidable class C, while, for each class C, it belongs to $\rightharpoonup$C and to C/poly. Hence, we have that C $\subset \rightharpoonup$C and C $\subset$ C/poly. Moreover, ϵHALT belongs to $\|\!\rightharpoonup$C, but not to $\rightharpoonup$C, since the language $\{\langle f(\epsilon), y\rangle | y \in \text{HALT}\}$ is undecidable for any function f. Hence, $\rightharpoonup$C $\subset \|\!\rightharpoonup$C. □

Theorem 2.14 *Let* C *be a complexity class that conforms to Assumption 2.1. If there exists at least one language* $L \notin \mathrm{C/poly}$*, then* $\mathrm{C/poly} \subset \|\!\rightharpoonup\!\mathrm{C}$ *holds.*

Proof. Let L be the language that does not belong to C/poly. Consider the language

$$L' = \{\langle x, y\rangle \mid x \in L\}$$

We now prove that:

1. L' is not in C/poly,
2. $L' \in \|\!\rightharpoonup\!\mathrm{C}$

Proof of 1:. Suppose there exist a poly-size function f and a C language S such that

$$\langle x, y\rangle \in L' \quad \text{iff} \quad \langle f(|x| + |y|), \langle x, y\rangle\rangle \in S$$

Then, L would be in C/poly as well. Indeed, just consider that

$$\begin{aligned} x \in L \ \text{ iff } \ & \langle x, \epsilon\rangle \in L' \\ \text{iff } \ & \langle f(|x| + |\epsilon|)), \langle x, \epsilon\rangle\rangle \in S \\ \text{iff } \ & \langle f(|x|), \langle x, \epsilon\rangle\rangle \in S \end{aligned}$$

Now, take

$$S' = \{\langle a, b\rangle \mid \langle a, \langle b, \epsilon\rangle\rangle \in S\}$$

which is in C. As a result,

$$x \in L \quad \text{iff} \quad \langle f(|x|), x\rangle \in S'$$

where f is poly-size and S' is in C. This implies $L \in \mathrm{C/poly}$. Since this is false by hypothesis, $L' \notin \mathrm{C/poly}$ holds.

Proof of 2:. Let

$$f(x, k) = \begin{cases} 1 & \text{if } x \in L \\ 0 & \text{otherwise} \end{cases}$$

and $S = \{\langle 1, y\rangle \mid y \text{ is any string }\}$. It holds that $\langle x, y\rangle \in L'$ if and only if $\langle f(x, |y|), y\rangle \in S$, where f is poly-size. As a result, L' is in $\|\!\rightharpoonup\!\mathrm{C}$, while it is not in C/poly. □

Theorem 2.15 *If the polynomial hierarchy is proper, then for each class* C *in the polynomial hierarchy it holds that* $\mathrm{C/poly} \subset \|\!\rightharpoonup\!\mathrm{C}$.

Proof. If the polynomial hierarchy is proper, then (cf. [40]) also the non-uniform polynomial hierarchy is proper. Hence for each class C of the polynomial hierarchy there exists a language L such that $L \notin \mathrm{C/poly}$. □

Theorem 2.16 *Let* C *be a class in the polynomial hierarchy. It holds that* C/poly $\not\subseteq \rightsquigarrow$C. *Moreover, if there exist problems that are not in* C/poly*, it also holds that* $\rightsquigarrow$C $\not\subseteq$ C/poly.

Proof. As for the first claim, let us consider the language ϵHALT (cf. (A.1) in proof of Theorem 2.7 for the definition of HALT) This problem is clearly in C/poly but not in $\rightsquigarrow$C.

As for the second claim, let us assume that there is a problem L which is not in C/poly. Then, the problem

$$\epsilon L = \{\langle x, y\rangle \mid x \in L \text{ and } y = \epsilon\}$$

is not in C/poly also. On the other side, this problem is in $\rightsquigarrow$C, since we can define f as the function that determines whether x is in L or not, and S' as the language $\{\langle a, y\rangle \mid a = 1 \text{ and } y = \epsilon\}$. □

Theorem 3.2 CMI *is* $\|\!\rightsquigarrow\Pi_2^p$*-complete.*

Proof. We give a $\|\!\rightsquigarrow$ reduction from a $\|\!\rightsquigarrow\Pi_2^p$-complete problem. The prototypical Π_2^p-complete problem is deciding the truth of a (restricted form of) quantified boolean formula. More precisely, given two sets of propositional atoms $X = \{x_1, \ldots, x_n\}$, $Y = \{y_1, \ldots, y_m\}$, and a 3CNF formula E containing literals on the alphabet $X \cup Y$, we call a $\forall\exists$-QBF a quantified boolean formula F of the following form:

$$F = \forall x_1, \ldots, x_n \exists y_1, \ldots, y_m . E \tag{A.5}$$

We call E the *matrix* of F. We call $\forall\exists$-QBF also the set of formulae of the form (A.5) which evaluate to true. By Theorem 2.9, we have that the problem $\epsilon\forall\exists$-QBF is $\|\!\rightsquigarrow\Pi_2^p$-complete. Thus, we now show a reduction from $\epsilon\forall\exists$-QBF to CMI.

To simplify the definition of the three functions f_1, f_2, g needed in the $\|\!\rightsquigarrow$ reduction, we assume the following: in any $\forall\exists$-QBF F the existentially quantified variables (the set Y) have indices from 1 to (at most) $|F|$, and the universally quantified variables (the set X) have indices from $|F| + 1$ to $2|F|$. Observe that this is always possible, since a formula F cannot use more than $|F|$ variables. From now on, we assume that $|F| = ||X|| = ||Y|| = k$.

Let C be a set of new atoms, one for each three-literals clause over $X \cup Y$, i.e., $C = \{c_i \mid \gamma_i \text{ is a three-literals clause of } X \cup Y\}$. Moreover, let D be a set of new atoms in one-to-one correspondence with C, and let Z be another set of new atoms in one-to-one correspondence with X. Finally, let W be the set of atoms $C \cup D \cup \{u\}$, where u is a distinguished atom. To avoid conflicts, we assume that the indices of all these new variables are greater than $2k$. However, we keep the notation z_i for a variable in Z to highlight the correspondence between z_i and x_i, and the same for the variables in C and D.

Given the $\forall\exists$-QBF F with matrix E, we denote

$$C_E = \{c_i \in C \mid \gamma_i \text{ is a clause of } E\}$$

and similarly for D_E. Moreover, $\overline{C_E} = C - C_E$, and $\overline{D_E} = D - D_E$.

We now define a formula imposing non-equivalence between atoms in X and their correspondents in Z, and the same for C and D. We call Σ the following 2CNF formula:

$$\Sigma = \bigwedge_{i=1}^{k} (x_i \neq z_i) \wedge \bigwedge_{i=1}^{||C||} (c_i \neq d_i)$$

Now we want to code every possible 3CNF formula over $X \cup Y$, using the atoms in C as "enabling gates". We call Γ the following 4CNF formula:

$$\Gamma = \bigwedge_{i=1}^{||C||} (c_i \rightarrow \gamma_i)$$

We define $f_1(\delta, k)$ as:

$$f_1(\delta, k) \;=\; \begin{cases} \mathsf{true} & \text{if } \delta \neq \epsilon \\ T_k & \text{otherwise} \end{cases}$$

where T_k is defined as :

$$T_k = \Sigma \wedge ((u \wedge y_1 \wedge \cdots \wedge y_n) \vee (\neg u \wedge \Gamma)). \tag{A.6}$$

Note that the size k of F is needed to build T_k, because Σ, Γ and the set Y have a size that depends on k. Observe that the size $|T_k|$ is $O(k^3)$, and that T_k can be rewritten as an equivalent 5CNF formula. Moreover, T_k does not depend on a specific $\forall\exists$-QBF F, but only on its size.

The function f_2 is defined as $f_2(\delta, k) = \delta$, while the function g is defined as:

$$g(f_2(\delta, k), F) = g(\delta, F) \;=\; \begin{cases} \mathsf{false} & \text{if } \delta \neq \epsilon \\ Q_F & \text{otherwise} \end{cases}$$

where Q_F is defined as:

$$Q_F = \neg u \vee \neg y_1 \vee \cdots \vee \neg y_m \vee \Big(\bigvee_{c_i \in C_E} \neg c_i\Big) \vee \Big(\bigvee_{c_i \in \overline{C_E}} c_i\Big) \vee \Big(\bigvee_{d_i \in D_E} d_i\Big) \vee \Big(\bigvee_{d_i \in \overline{D_E}} \neg d_i\Big)$$

Observe that Q_F contains all atoms of $Y \cup W$.

We now show that $\langle \delta, F\rangle \in \epsilon\forall\exists$-QBF iff $\langle f_1(\delta, |F|), g(\delta, F)\rangle \in$ CMI. For the case $\delta \neq \epsilon$, the pair $\langle \delta, F\rangle$, which does not belong to $\epsilon\forall\exists$-QBF, is transformed into $\langle \mathsf{true}, \mathsf{false}\rangle$ which does not belong to CMI. For the case $\delta = \epsilon$, we show that $\langle \epsilon, F\rangle \in \epsilon\forall\exists$-QBF iff $\langle T_{|F|}, Q_F\rangle \in$ CMI. From now on, we omit the subscript of T and Q to improve readability.

We denote with $\mathcal{M}(T)$ the models of T, and similarly with $\mathcal{M}(\Sigma)$ the models of Σ. Given $M \in \mathcal{M}(\Sigma)$, we define its extension $\mathcal{E}(M) = M \cup Y \cup \{u\}$.

The following properties of T are easy to prove. A full proof can be found in [11].

PROPOSITION A.1. *Let T be as in (A.6). Then:*

1.*T is always satisfiable.*
2.*If $M \in \mathcal{M}(\Sigma)$ then $\mathcal{E}(M) \in \mathcal{M}(T)$.*
3.*Every model $M \in \mathcal{M}(T)$ defines a unique matrix $E = \{\gamma_i \mid c_i \in M\}$.*
4.*For every model $M \in \mathcal{M}(T)$ such that $M \cap C = C_E$, M satisfies Γ iff M satisfies E.*

We now prove that F is valid iff Q is true in all minimal models of T.

If part. Suppose that Q is true in all minimal models of T. Consider any model $M \in \mathcal{M}(\Sigma)$ such that $M \cap C = C_E$, that is, M contains exactly the atoms of C corresponding to clauses of E. Note that the model $\mathcal{E}(M) = M \cup Y \cup \{u\}$ cannot be a minimal model, because it does not satisfy Q. Hence for any such M there exists a different extension $\mathcal{E}_1(M)$, with $\mathcal{E}_1(M) \subset \mathcal{E}(M)$, that is a minimal model of T satisfying Q. Since $\mathcal{E}_1(M)$ satisfies Q, it must satisfy $\neg u$. As a consequence, $\mathcal{E}_1(M) \models \Gamma$ (cf. (A.6)). But any model satisfying Γ and whose intersection with C equals C_E satisfies also E (cf. Proposition A.1 point 4). Hence for any $M \in \mathcal{M}(\Sigma)$ such that $M \cap C = C_E$, there is an extension $\mathcal{E}_1(M)$ satisfying E. Since each $M \in \mathcal{M}(\Sigma)$ contains a truth assignment to variables in X, it follows that for each assignment to variables in X there is an assignment to variables in Y (namely, $\mathcal{E}_1(M) \cap Y$) such that E is satisfied. Therefore, F is valid.

Only if part. Assume that there exists a minimal model M of T such that $M \models \neg Q$. Observe that $M \models u \wedge y_1 \wedge \cdots \wedge y_n$, $M \cap C = C_E$, $M \cap D = \overline{D_E}$. Let $M_\Sigma = M \cap (X \cup Z \cup C \cup D)$. Obviously, M is an extension of M_Σ. The minimality of M implies that $M \not\models \Gamma$, otherwise $M - \{u\}$ would be a model of T. Since $M \cap C = C_E$ and $M \not\models \Gamma$, by Proposition A.1 (point 4) we also have that $M \not\models E$. Again from the minimality of M, it follows that no other extension M' of M_Σ satisfies T. As a consequence, there exists an assignment to the variables in X (i.e., $M_\Sigma \cap X$) for which there is no assignment to the variables in Y that makes E true. Therefore F is not valid.□