Parameter Compilation

Hubie Chen

University of the Basque Country (UPV/EHU), E-20018 San Sebastián, Spain *and* IKERBASQUE, Basque Foundation for Science, E-48011 Bilbao, Spain

Abstract

In resolving instances of a computational problem, if multiple instances of interest share a feature in common, it may be fruitful to compile this feature into a format that allows for more efficient resolution, even if the compilation is relatively expensive. In this article, we introduce a formal framework for classifying problems according to their compilability. The basic object in our framework is that of a parameterized problem, which here is a language along with a parameterization – a map which provides, for each instance, a so-called parameter on which compilation may be performed. Our framework is positioned within the paradigm of parameterized complexity, and our notions are relatable to established concepts in the theory of parameterized complexity. Indeed, we view our framework as playing a unifying role, integrating together parameterized complexity and compilability theory.

1998 ACM Subject Classification F.1.3 [Computation by Abstract Devices] Complexity measures and classes

Keywords and phrases compilation, parameterized complexity

Digital Object Identifier 10.4230/LIPIcs.IPEC.2015.127

1 Introduction

In resolving instances of a computational problem, if it is the case that multiple instances of interest share a feature in common, it may be fruitful to *compile* this feature into a format that allows for more efficient resolution, even if the compilation is relatively expensive. As a first, simple example, consider the problem of deciding if two nodes of an undirected graph are connected. If it is anticipated that many such connectivity queries will share the same graph G, it may be worthwhile to compile G into a format that will allow for accelerated resolution of the queries. As a second example, consider the problem of evaluating a database query on a database. If one is interested in a small set of queries that will be posed to numerous databases, it may be worthwhile to compile the queries of interest into a format that allows for the fastest evaluation. Note that a relatively expensive compilation process may be worthwhile if its results are amortized by repeated use. Indeed, one may conceive of compilation as an off-line preprocessing, whose expense is offset by its later on-line use.

In this article, we attempt to make an infrastructural contribution by introducing a formal framework for classifying problems according to their compilability. Such a framework was previously presented by Cadoli, Domini, Liberatore, and Schaerf [2], (hereafter, *CDLS*); we will discuss the relationship between our framework and theirs below.

The basic object in our framework is a *paramaterized problem*, which we define to be a language Q along with a *parameterization* κ, a polynomial-time computable mapping defined from strings to strings. (For precise details and justifications of definitions, refer to the technical sections of the article.) As usual, we refer to $\kappa(x)$ as the *parameter* of an instance x. In our framework, we wish to understand for which problems the parameters can be succinctly compiled into a form such that, post-compilation, the problem can be

10th International Symposium on Parameterized and Exact Computation (IPEC 2015).
Editors: Thore Husfeldt and Iyad Kanj; pp. 127–137

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

resolved in polynomial-time. The base class of our framework, called poly-comp-PTIME, is (essentially) defined to contain a parameterized problem (Q, κ) if there exists a polynomial-length, computable function c such that if each instance x is always presented along with $c(\kappa(x))$, then each instance can be resolved in polynomial time. The function c models the notion of compilation of the parameters. In order to give evidence of non-containment in the class poly-comp-PTIME and also to facilitate problem classification, we introduce a hierarchy of parameterized complexity classes chopped-$\mathcal{C}$, one for each classical complexity class $\mathcal{C}$; we observe (for example) that chopped-NP is not contained in poly-comp-PTIME, assuming that the polynomial hierarchy does not collapse (see Proposition 9 and Theorem 19), and hence hardness of a problem for chopped-NP can be construed as evidence of non-containment in poly-comp-PTIME. We observe a number of completeness and hardness results for chopped-NP (Section 6).[1] The class poly-comp-PTIME and the classes chopped-$\mathcal{C}$ are all subsets of the parameterized class FPT, which is considered to be the basic notion of tractability in the paradigm of parameterized complexity.[2] We believe that the introduced classes constitute a natural stratification of FPT, whose study might well lead to deeper theory.

In the CDLS framework, the basic object is a language consisting of pairs of strings (called a *language of pairs*), and one aims to understand when a compilation can be applied to the first entry of each pair so as to allow for efficient decision. This is a point of difference with our framework, but note that the notions from our framework can be readily applied to the languages of pairs that CDLS study by using the parameterization π_1 that returns the first entry of a pair. Another point of difference between our framework and theirs is that their analog of our compilation function c is not required to be computable; while this makes the negative results stronger, in our view there ought to be a focus on positive results, which are rendered less meaningful without the computability requirement. (Actually, we are not aware of any natural computable problem for which the presence or absence of this requirement makes a difference.) Although these differences may appear slight, by initiating our theory with our particular choice of definitions, we are able to position our framework within the language and tradition of parameterized complexity and relate our notions to existing ideas in parameterized complexity. For instance, although not difficult, we can directly relate the notion of a *polynomial kernelization* to the classes chopped-$\mathcal{C}$ (Proposition 10) and use this relationship to observe the chopped-NP-completeness of the standard parameterization of the hitting set problem for hypergraphs of bounded edge size (see Theorem 30). We also believe that the theory that results from our framework's definitions witnesses that working with parameterized problems as opposed to languages of pairs allows for greater flexibility and smoother formulation (consider, for example, the characterization of chopped-$\mathcal{C}$ using the *length parameterization* given by Proposition 14).

Our framework and that of CDLS also differ later in the respective developments. Notably, our notion of reduction (Definition 11) is readily seen to be a restricted version of the usual fpt many-one reduction in parameterized complexity, and we believe that our notion of reduction is conceptually simpler to comprehend than that of CDLS [2, Definition 2.8]. Despite these differences – and we view this as crucial – we demonstrate how classification results obtained in the CDLS framework can be formulated and obtained in our framework; this is made precise and performed in Section 5.

---

[1] These results include a hardness result on model checking existential positive sentences (Proposition 31); we remark that obtaining a broader understanding of the non-compilability results in the author's previous study of model checking [5] was in fact a motivation of the present article.

[2] Note that the containment of poly-comp-PTIME in FPT is essentially observed (in different language) in the last paragraph of Section 5 of [2].

The presentation and development of our framework may thus be viewed as playing a unifying role, integrating together parameterized complexity and compilation. Our choices of definitions and in formulation allows us to directly relate the resulting concepts to the theory of parameterized complexity. At the same time, we believe that these concepts capture in an essential way the core mathematical content and the core ideas of the CDLS framework (as borne out by our results and discussion in Section 5).

Related work. The CDLS framework was deployed after its introduction to analyze the compilability of reasoning tasks, see for example [10, 11].

In the context of compiling propositional formulas, a notion of compilation whereby a compiled version should have the same models as the original formula was studied, for example by Gogic et al. [9] and by Darwiche and Marquis [6]; see also the recent work by Bova et al. [1].

Variants of the CDLS framework that relaxed the requirement that the size of compilations be polynomial were also studied [3, 4].

Finally, we mention that Fan, Geerts, and Neven [7] also developed a framework for classifying problems according to compilability, with a focus on efficient parallel processing (modelled using the complexity class NC) following a polynomial-time compilation. We believe that it may be of interest to better understand and develop the relationship between our framework and theirs. While we leave such a study to future work, we mention that their notion of Π-*tractability* on a language Q of pairs can be described using our framework.[3]

2 Preliminaries

Throughout, π_i denotes the operator that, given a tuple, returns the ith entry of the tuple.

When T is a set, we use $\wp_{\text{fin}}(T)$ to denote the set $\{S \subseteq T \mid S \text{ is finite}\}$.

We generally use Σ to denote the alphabet over which strings are formed, and generally assume $\{0,1\} \subseteq \Sigma$. As is standard, we freely interchange between elements of Σ^* and $\Sigma^* \times \Sigma^*$. When $k \geq 0$, we use $\Sigma^{\leq k}$ to denote the set of strings in Σ^* of length less than or equal to k. For $n \in \mathbb{N}$, we use $\text{un}(n)$ to denote its unary encoding 1^n as a string.

We assume that languages under discussion are non-trivial, that is, not equal to $\emptyset$ nor Σ^*. We use PTIME to denote the set of all languages decidable in polynomial time, and fPTIME to denote the set of all functions from Σ^* to Σ^* that are computable in polynomial time.

Here, by a *parameterization*, we refer to a map from Σ^* to Σ^*. Relative to a parameterization $\kappa : \Sigma^* \to \Sigma^*$, it is typical to refer to $\kappa(x)$ as the *parameter* of the string x. While it is typical in the literature to define a parameterization to be a map from Σ^* to $\mathbb{N}$, in this article we want to apply compilation functions to parameters and discuss the *length* of the results, and we find that this is facilitated in many cases by permitting the parameter of a string to be a string itself. Throughout, we employ the following assumption (which is discussed below in Remark 4).

▶ **Assumption 1.** Each parameterization is polynomial-time computable, that is, in fPTIME.

We use len to denote the parameterization defined by $\text{len}(x) = \text{un}(|x|)$. A *parameterized problem* is a pair (Q, κ) consisting of a language Q and a parameterization κ.

[3] Precisely, a language Q of pairs being Π-*tractable* can be verified to be equivalent to the parameterized problem (Q, π_1) being in our class poly-comp-NC via a poly-compilable function $g(x,y) = f(c(\pi_1(x,y)),(x,y)) = f(c(x),(x,y))$ where c is polynomial-time computable.

By a *classical complexity class*, we refer to a set of computable languages. For a classical complexity class $\mathcal{C}$, we define para-$\mathcal{C}$ to be the set that contains a parameterized problem (Q, κ) if there exists a computable function $c : \Sigma^* \to \Sigma^*$, and a language $Q' \subseteq \Sigma^* \times \Sigma^*$ in $\mathcal{C}$ such that, for each string $x \in \Sigma^*$, it holds that $x \in Q \Leftrightarrow (c(\kappa(x)), x) \in Q'$. We define FPT to be para-PTIME (although this is perhaps not the usual definition of FPT, it is equivalent [8, Theorem 1.37]).

As usual, when $\mathcal{D}$ is a set of problems (that is, a set of either languages or parameterized problems), we say that a problem P' is $\mathcal{D}$-*hard* under a notion of reduction if each P in $\mathcal{D}$ reduces to P'; if in addition $P' \in \mathcal{D}$, we say that P' is $\mathcal{D}$-*complete*. We say that $\mathcal{D}$ is *closed* under a notion of reduction if, when P reduces to P' and $P' \in \mathcal{D}$, it holds that $P \in \mathcal{D}$.

3 Framework

3.1 Problem classes

In this subsection, we introduce the complexity classes of our framework. We begin by introducing two basic definitions. By a *length function*, we refer to a function from $\mathbb{N}$ to $\mathbb{N}$.

▶ **Definition 2.** Let $\mathcal{L}$ be a set of length functions.

- A function $c : \Sigma^* \to \Sigma^*$ is said to be $\mathcal{L}$-length if there exists $\ell \in \mathcal{L}$ such that for each $x \in \Sigma^*$, it holds that $|c(x)| \leq \ell(|x|)$.
- A function $g : \Sigma^* \to \Sigma^*$ is $\mathcal{L}$-*compilable* with respect to a parameterization κ if there exist $f \in$ FPTIME and a computable, $\mathcal{L}$-length function $c : \Sigma^* \to \Sigma^*$ such that (for each $x \in \Sigma^*$) $g(x) = f(c(\kappa(x)), x)$.

Put informally, a function g is $\mathcal{L}$-compilable if, when one has the result of applying c to the parameter of an instance x, the value $g(x)$ can be efficiently computed. The function c can be thought of as performing a precomputation or compilation of the parameter. Here, we do not place any restriction on the computational resources needed to compute c, other than requiring that c is computable. We view the requirement that c be computable as natural in terms of claiming positive results, as we find it hard to argue that a non-computable compilation would actually be usable. We do restrict the length of c according to $\mathcal{L}$; we will be most interested in the case where the length of c is polynomially bounded.

With this terminology in hand, we can now define our first classes of parameterized problems.

▶ **Definition 3.** Let $\mathcal{L}$ be a set of length functions, and let $\mathcal{C}$ be a classical complexity class.

- We say that a parameterized problem (Q, κ) is $\mathcal{L}$-*compilable to* $\mathcal{C}$ if there exists a function $g : \Sigma^* \to \Sigma^*$ that is $\mathcal{L}$-compilable (with respect to κ) and a language $Q' \in \mathcal{C}$ such that (for each $x \in \Sigma^*$) $x \in Q \Leftrightarrow g(x) \in Q'$.
- We define $\mathcal{L}$-comp-$\mathcal{C}$ to be the set that contains each parameterized problem that is $\mathcal{L}$-compilable to $\mathcal{C}$.

When $\mathcal{L}$ is the set of all polynomials on $\mathbb{N}$, we define poly-comp-$\mathcal{C}$ as $\mathcal{L}$-comp-$\mathcal{C}$ and speak, for instance, of *poly-compilability*; similarly, when $\mathcal{L}$ is the set $\cup\{O(2^p) \mid p \text{ is a polynomial}\}$ of *exponential functions*, we define exp-comp-$\mathcal{C}$ as $\mathcal{L}$-comp-$\mathcal{C}$ and speak, for instance, of *exp-compilability*.

▶ **Remark 4.** In this paper, the smallest class that we will consider is poly-comp-PTIME, and we will regard an inclusion result in this class as the most positive result demonstrable on a parameterized problem. Suppose that a parameterized problem (Q, κ) is in poly-comp-PTIME via $g(x) = f(c(\kappa(x)), x)$ and Q'. One way to intuitively interpret this inclusion is as follows.

Suppose that the value $c(k)$ is known for parameter values k in a limited range. Then, for each instance $x \in \Sigma^*$ having parameter value $\kappa(x)$ in that limited range, whether or not $x \in Q$ can be determined efficiently, by applying the efficiently computable function f to $(c(\kappa(x)), x)$ and then by invoking an efficient decision procedure for Q'. Indeed, our intention here is to model the notion of efficient decidability modulo knowledge of c; this is why we put into effect Assumption 1.

We observe the following upper bound on each class $\mathcal{L}$-comp-$\mathcal{C}$, which in particular indicates that poly-comp-PTIME $\subseteq$ FPT.

▶ **Proposition 5.** *Let $\mathcal{L}$ be a set of length functions, and let $\mathcal{C}$ be a classical complexity class that is closed under many-one polynomial-time reduction. It holds that $\mathcal{L}$-comp-$\mathcal{C} \subseteq$ para-$\mathcal{C}$.*

Proof. Suppose that (Q, κ) is $\mathcal{L}$-compilable to $\mathcal{C}$ via $g(x) = f(c(\kappa(x)), x)$ and $Q'' \in \mathcal{C}$, so that $x \in Q \Leftrightarrow g(x) \in Q''$. Define $Q' = \{(a, b) \mid f(a, b) \in Q''\}$. The language Q' is many-one polynomial-time reducible to Q'' via f, so $Q' \in \mathcal{C}$. We have $x \in Q \Leftrightarrow (c(\kappa(x)), x) \in Q'$, implying that $Q \in$ para-$\mathcal{C}$. ◀

We now define a family of complexity classes which will be used to classify parameterized problems in FPT according to their compilability, and in particular to give evidence of non-inclusion in poly-comp-PTIME, via hardness results.

▶ **Definition 6.** For each classical complexity class $\mathcal{C}$, we define chopped-$\mathcal{C}$ to be the set that contains each parameterized problem (Q, κ) that is in poly-comp-$\mathcal{C}$ via a function g for which there exists a polynomial $p : \mathbb{N} \to \mathbb{N}$ such that (for each $x \in \Sigma^*$) $|g(x)| \leq p(|\kappa(x)|)$.

For the sake of understanding this definition, let us call the restriction of a language Q' to $Q' \cap \Sigma^{\leq k}$ the *chop having magnitude k* of Q'. Then, intuitively speaking, a problem is in chopped-$\mathcal{C}$ if it is in poly-comp-$\mathcal{C}$ via g and Q' where $g(x)$ accesses only a chop (of Q') having magnitude restricted by a polynomial in the parameter of x. The following proposition is clear from the definition of chopped-$\mathcal{C}$.

▶ **Proposition 7.** *For each classical complexity class $\mathcal{C}$, it holds that*

$$\text{chopped-}\mathcal{C} \subseteq \text{poly-comp-}\mathcal{C}.$$

We also have the following upper bound on chopped-$\mathcal{C}$, which shows that the classes chopped-$\mathcal{C}$ constitute a stratification of the class FPT.

▶ **Proposition 8.** *For each classical complexity class $\mathcal{C}$, it holds that*

$$\text{chopped-}\mathcal{C} \subseteq \text{exp-comp-PTIME},$$

and hence that chopped-$\mathcal{C} \subseteq$ FPT *(by Proposition 5).*

Proof. We prove that chopped-$\mathcal{C} \subseteq$ exp-comp-PTIME. Fix $x_N, x_Y \in \Sigma^*$ and $P \in$ PTIME such that $x_Y \in P$ and $x_N \notin P$. Assume that (Q, κ) is in chopped-$\mathcal{C}$ via $g(x) = f(c(\kappa(x)), x)$, the polynomial p, and $Q' \in \mathcal{C}$. Let $c_1^+ : \Sigma^* \to \Sigma^*$ be the function computed by the algorithm that, given $k \in \Sigma^*$, loops over each string y in $\Sigma^{\leq p(|k|)}$ and, for each such string y, outputs 1 or 0 depending on whether or not $y \in Q'$; thus, $|c_1^+(k)| = |\Sigma^{\leq p(|k|)}|$. Define $c^+(k) = (c_1^+(k), c(k), k)$. Let f^+ be a function computed by a polynomial-time algorithm that, given a string $((d_1, d, k), x)$ where d_1 is a string over $\{0, 1\}$ of length $|\Sigma^{\leq p(|k|)}|$, computes $x' = f(d, x)$, computes the bit b of d_1 corresponding to x' (whenever $|x'| \leq p(|k|)$), and outputs x_Y or x_N depending on whether or not $b = 1$ or $b = 0$. The function $g^+(x) = f^+(c^+(\kappa(x)), x)$ witnesses that (Q, κ) is exp-compilable to PTIME: We have that $x \in Q$ iff $f(c(\kappa(x)), x) \in Q'$ iff $f^+((c_1^+(\kappa(x), c(\kappa(x)), \kappa(x)), x) = x_Y$ iff $f^+(c^+(\kappa(x)), x) \in P$. ◀

We observe that our base class poly-comp-PTIME coincides with the class chopped-PTIME, which is the smallest class that we will consider from the hierarchy of classes chopped-$\mathcal{C}$.

▶ **Proposition 9.** chopped-PTIME = poly-comp-PTIME.

The classes chopped-$\mathcal{C}$ can be directly related to kernelization in the following way. Here, we say that a parameterized problem (Q, κ) has a *polynomial kernelization* if there exists a polynomial-time computable function $K : \Sigma^* \to \Sigma^*$ and a polynomial $p : \mathbb{N} \to \mathbb{N}$ such that (for each $x \in \Sigma^*$) $x \in Q \Leftrightarrow K(x) \in Q$ and $|K(x)| \leq p(|\kappa(x)|)$.

▶ **Proposition 10.** *Suppose that a parameterized problem (Q, κ) has a polynomial kernelization and $\mathcal{C}$ is a classical complexity class such that $Q \in \mathcal{C}$. Then, the problem (Q, κ) is in* chopped-$\mathcal{C}$.

Proof. We have that (Q, κ) in poly-comp-$\mathcal{C}$ via K (the function from the definition of polynomial kernelization), since $x \in Q \Leftrightarrow K(x) \in Q$. Moreover, it holds that there exists a polynomial p such that $|K(x)| \leq p(|\kappa(x)|)$ by the definition of polynomial kernelization. ◀

3.2 Reduction

We now introduce a notion of reduction for comparing the compilability of parameterized problems.

▶ **Definition 11.** We say that a parameterized problem (Q, κ) *poly-comp reduces* to another parameterized problem (Q', κ') if there exists a function $g : \Sigma^* \to \Sigma^*$ that is poly-compilable with respect to κ and a poly-length, computable function $s : \Sigma^* \to \wp_{\mathrm{fin}}(\Sigma^*)$ such that (for each $x \in \Sigma^*$) it holds that $x \in Q \Leftrightarrow g(x) \in Q'$ and that $\kappa'(g(x)) \in s(\kappa(x))$.

The notion of poly-comp reduction can be viewed as a restricted version of fpt many-one reduction. (Consider, for example, the definition given by Flum and Grohe [8, Definition 2.1]; the function g in Definition 11 can be seen to be computable by a fpt-algorithm, and the condition on the function s ensures that their condition (3), when reformulated for parameterizations of the type considered here, holds.)

Note that, in Definition 11, we assume that the set $s(x)$ is given according to a standard representation that lists the strings therein; hence, as a consequence of the assumption that s is poly-length, the size $|s(x)|$ of $s(x)$ is bounded above by a polynomial in $|x|$.

We have the following two basic properties of poly-comp reduction.

▶ **Theorem 12.** *For each classical complexity class $\mathcal{C}$, it holds that* poly-comp-$\mathcal{C}$ *is closed under poly-comp reduction.*

▶ **Theorem 13.** *Poly-comp reducibility is transitive.*

We now give an alternative characterization of chopped-$\mathcal{C}$ in terms of poly-comp reduction.

▶ **Proposition 14.** *Let $\mathcal{C}$ be a classical complexity class. A parameterized problem (Q, κ) is in* chopped-$\mathcal{C}$ *if and only if there exists a language $Q' \in \mathcal{C}$ such that (Q, κ) poly-comp reduces to (Q', len).*

From the just-given characterization of chopped-$\mathcal{C}$, we may infer the following two results.

▶ **Proposition 15.** *For each classical complexity class $\mathcal{C}$, the class* chopped-$\mathcal{C}$ *is closed under poly-comp reduction.*

Proof. This is a consequence of Proposition 14 and Theorem 13. ◀

When discussing a class chopped-$\mathcal{C}$*, we assume by default that* hardness *and* completeness *are with respect to poly-comp reducibility.*

▶ **Proposition 16.** *Let $\mathcal{C}$ be a classical complexity class and assume that Q^+ is $\mathcal{C}$-complete under many-one polynomial-time reduction. Then, the parameterized problem (Q^+, len) is complete for* chopped-$\mathcal{C}$*.*

4 Chopped classes and advice

In this section, we relate the classes chopped-$\mathcal{C}$ to advice-based complexity classes; this will allow us to provide evidence of separation between classes of the form chopped-$\mathcal{C}$.

We first present a known notion from computational complexity theory, the notion of an advice version of a complexity class. For each classical complexity class $\mathcal{C}$, we define $\mathcal{C}/\mathrm{poly}$ to be the set that contains a language Q if and only if there exists a poly-length map $a : \{1\}^* \to \Sigma^*$ and a language $Q' \in \mathcal{C}$ such that, for each $x \in \Sigma^*$, it holds that $x \in Q \Leftrightarrow (a(\mathrm{un}(|x|), x)) \in Q'$.

The following theorem shows that containment of one chopped class in another implies a containment in classical complexity.

▶ **Theorem 17.** *Let $\mathcal{C}$ and $\mathcal{C}'$ be classical complexity classes where $\mathcal{C}'$ is closed under many-one polynomial-time reduction. If* chopped-$\mathcal{C} \subseteq$ chopped-$\mathcal{C}'$*, then $\mathcal{C} \subseteq \mathcal{C}'/\mathrm{poly}$.*

To prove this theorem, we first establish a lemma.

▶ **Lemma 18.** *Let $\mathcal{C}'$ be a classical complexity class that is closed under many-one polynomial-time reduction. If Q is a language such that $(Q, \mathrm{len}) \in$* chopped-$\mathcal{C}'$*, then $Q \in \mathcal{C}'/\mathrm{poly}$.*

Proof of Theorem 17. Suppose that $Q \in \mathcal{C}$. By Proposition 14, it holds that (Q, len) is in chopped-$\mathcal{C}$. By hypothesis, it holds that (Q, len) is in chopped-$\mathcal{C}'$. By Lemma 18, it follows that $Q \in \mathcal{C}'/\mathrm{poly}$. ◀

We use Σ_i^{P} and Π_i^{P} (with $i \geq 0$) to denote the classes of the polynomial hierarchy (PH); recall that $\Sigma_0^{\mathsf{P}} = \Pi_0^{\mathsf{P}} = \mathsf{PTIME}$, $\Sigma_1^{\mathsf{P}} = \mathsf{NP}$, and $\Pi_1^{\mathsf{P}} = \mathsf{co\text{-}NP}$. For each $i \geq 0$, let us say that the classes Σ_i and Π_i are at the *ith level* of the PH. Let us say that a class $\mathcal{C}'$ of the PH is *above* another class $\mathcal{C}$ of the PH if they are equal or if the level j of $\mathcal{C}'$ is strictly greater than the level i of $\mathcal{C}$.

▶ **Theorem 19** (follows from [12])**.** *Suppose that $\mathcal{C}$ and $\mathcal{C}'$ are classes of the PH such that $\mathcal{C}'$ is not above $\mathcal{C}$.*

- *If* chopped-$\mathcal{C} \subseteq$ chopped-$\mathcal{C}'$*, then the PH collapses.*
- *A parameterized problem (Q, κ) that is* chopped-$\mathcal{C}$*-hard is not in* chopped-$\mathcal{C}'$*, unless the PH collapses.*

Proof. For the first claim, it follows from Theorem 17 that $\mathcal{C} \subseteq \mathcal{C}'/\mathrm{poly}$; by [12], it follows that the PH collapses. For the second claim, observe that if (Q, κ) is chopped-$\mathcal{C}$-hard, then $(Q, \kappa) \in$ chopped-$\mathcal{C}'$ implies that chopped-$\mathcal{C} \subseteq$ chopped-$\mathcal{C}'$, by the closure of chopped-$\mathcal{C}'$ under poly-comp reduction (Theorem 12). ◀

5 Relationship to the CDLS framework

In this section, we discuss the relationship between our framework and the CDLS framework. We in particular show that, in a sense that we make precise, the completeness results that they obtain for their problem classes can be formulated and obtained in our framework.

By a *language of pairs*, we refer to a subset of $\Sigma^* \times \Sigma^*$.

The CDLS framework defines, for each classical complexity class, a class which they refer to as the *class of problems non-uniformly compilable to a class C*, and which contains languages of pairs [2, Definition 2.7]. We give the following formulation of this definition.

▶ **Definition 20.** A language $B \subseteq \Sigma^* \times \Sigma^*$ of pairs is in mixed-$\mathcal{C}$ if there exists a poly-length, computable function $f : \Sigma^* \times \Sigma^* \to \Sigma^*$ and a language $B' \in \mathcal{C}$ of pairs such that $(x, y) \in B \Leftrightarrow (f(x, \mathrm{un}(|y|)), y) \in B'$.

Note that our definition is not exactly equivalent to theirs; we require that the function f is computable, while they do not. We do not know of any natural language of pairs for which this makes a difference; assuming computability of f will allow us to more readily relate the defined classes to those of our framework.

To illustrate how classification results on languages obtained in the CDLS framework can be obtained in our framework, we discuss three running examples (studied in [2]):

- Define CI *(clause inference)* to be the set of pairs (x, y) where x is a propositional 3CNF formula, y is a clause, and $x \models y$. We assume here that clauses do not contain repeated literals.
- Define MMC *(minimal model checking)* to be the set of pairs (x, y) where x is a propositional formula and y is a minimal model of x. By *minimal*, we mean with respect to the order $\leq$ where $z \leq z'$ if and only if all variables true under z are also true under z'.
- Define CMI *(clause minimal inference)* to be the set of pairs (x, y) where x is a propositional formula and y is a clause that is satisfied by all minimal models of x.

It is known and straightforward to verify that $\mathrm{CI}, \mathrm{MMC} \in$ co-NP and $\mathrm{CMI} \in \Pi_2^{\mathsf{p}}$. It follows immediately that $\mathrm{CI}, \mathrm{MMC} \in$ mixed-co-NP and $\mathrm{CMI} \in$ mixed-Π_2^{p}.

Let us say that a parameterized problem (Q, κ) has *poly-bounded slices* if there exists a polynomial p such that, for each $x \in Q$, it holds that $|x| \leq p(|\kappa(x)|)$. Each of the three parameterized problems (CI, π_1), (MMC, π_1), and (CMI, π_1) have poly-bounded slices (as is readily verified), and it can consequently be verified that $(\mathrm{CI}, \pi_1), (\mathrm{MMC}, \pi_1) \in$ chopped-co-NP and that $(\mathrm{CMI}, \pi_1) \in$ chopped-Π_2^{p}. It is indeed a general fact that when B is a language of pairs where (B, π_1) has poly-bounded slices, the classes mixed-$\mathcal{C}$ and chopped-$\mathcal{C}$ coincide, as made precise by the following theorem.

▶ **Theorem 21.** *Let $\mathcal{C}$ be a classical complexity class closed under many-one polynomial-time reduction. Let B be a language of pairs such that (B, π_1) has poly-bounded slices. Then, B is in* mixed-$\mathcal{C}$ *if and only if (B, π_1) is in* chopped-$\mathcal{C}$.

We now present a formulation of the notion of reduction used in the CDLS framework (see [2, Definition 2.8]).

▶ **Definition 22.** Let A and B be languages of pairs. A *mixed reduction* from A to B is a triple (f_1, f_2, g) of mappings from $\Sigma^* \times \Sigma^*$ to Σ where f_1 and f_2 are poly-length and computable, and g is polynomial-time computable, such that $(x, y) \in A \Leftrightarrow (f_1(x, \mathrm{un}(|y|)), g(f_2(x, \mathrm{un}(|y|)), y)) \in B$.

In analogy to Definition 20, here we require that the functions f_1 and f_2 are computable.

As a way of showing hardness, CDLS present mixed-reductions from languages of the form $\{\epsilon\} \times Q^+$ where Q^+ is a classical language that is hard. For example, they present the following reductions.

▶ **Theorem 23** (follows from [2, Proof of Theorem 2.10])**.** *There exists a* co-NP*-complete problem Q^+ such that there exists a mixed-reduction from $\{\epsilon\} \times Q^+$ to* CI.

▶ **Theorem 24** (follows from [2, Proof of Theorem 3.2]). *There exists a* Π_2^{p}*-complete problem* Q^+ *such that there exists a mixed-reduction from* $\{\epsilon\} \times Q^+$ *to* CMI.

We now present a general theorem showing that exhibiting a reduction from a language of the form $\{\epsilon\} \times Q^+$ yields a hardness result with respect to the classes chopped-$\mathcal{C}$, made precise as follows.

▶ **Theorem 25.** *Suppose that* A *and* B *are languages of pairs such that there exists a mixed reduction from* A *to* B*, and let* $\mathcal{C}$ *be a classical complexity class. If* $A = \{\epsilon\} \times Q^+$ *where* Q^+ *is* $\mathcal{C}$*-complete, then* (B, π_1) *is* chopped-$\mathcal{C}$*-hard.*

▶ **Corollary 26.** *The problem* (CI, π_1) *is* chopped-co-NP*-hard; the problem* (CMI, π_1) *is* chopped-Π_2^{p}*-hard.*

Proof. Follows from Theorems 23, 24 and 25. ◀

The other way in which CDLS show hardness is by presenting a mixed-reduction from a problem that has poly-bounded slices. For example, they prove the following.

▶ **Theorem 27** (follows from [2, Proof of Theorem 3.1]). *There exists a mixed-reduction from* CI *to* MMC.

We show that this form of reduction can be interpreted as a poly-comp reduction, made precise as follows.

▶ **Theorem 28.** *Suppose that* A *and* B *are languages of pairs such that there exists a mixed reduction from* A *to* B*. If* (A, π_1) *has poly-bounded slices, then* (A, π_1) *poly-comp reduces to* (B, π_1)*.*

▶ **Corollary 29.** *There exists a poly-comp reduction from* (CI, π_1) *to* (MMC, π_1)*, and hence (by Corollary 26) the problem* (MMC, π_1) *is* chopped-co-NP*-hard.*

Proof. Immediate from Theorems 27 and 28. ◀

At this point, we can observe that the non-compilability results that CDLS obtain can be obtained in our framework. For example, consider the following. As we have seen (and as stated in Corollaries 26 and 29), the problems (CI, π_1) and (MMC, π_1) are chopped-co-NP-hard. This implies that these two problems are not in chopped-PTIME, unless the PH collapses, via Theorem 19. We can also obtain the non-compilability results in (essentially) the form stated by CDLS: by invoking Theorem 21, it immediately follows that the problems CI and MMC are not in mixed-PTIME, unless the PH collapses. We want to emphasize here that the hardness proofs can be carried out using the notions and concepts of our framework.

6 Completeness and hardness for chopped-NP

In this section, we present completeness and hardness results for the class chopped-NP.

Define HAM-PATH to be the problem of deciding, given an undirected graph G, whether or not G contains a Hamiltonian path; define the parameterization γ so that $\gamma(G)$ is equal to the number of nodes in G. The problems 3-SAT and CIRCUIT-SAT are defined as usual. In the context of 3-SAT, ν is the parameterization that returns, given a formula ϕ, the number of variables that appear in ϕ. In the context of CIRCUIT-SAT, $\mu + \nu$ is the parameterization that returns, given a circuit C, the sum of the number of non-input gates and the number of input gates of C. For each $d \geq 2$, we consider d-HITTING-SET to be the problem where an

instance is a pair (H, k) consisting of a number $k \geq 1$ and a hypergraph H where each edge has size less than or equal to d, and one is to decide whether or not H has a hitting set of size less than or equal to k. Note that here, all numbers are represented in unary.

▶ **Theorem 30.** *The following problems are* chopped-NP*-complete:*
1. $(\text{HAM-PATH}, \gamma)$
2. $(\text{3-SAT}, \nu)$
3. $(\text{CIRCUIT-SAT}, \mu + \nu)$
4. $(d\text{-HITTING-SET}, \pi_2)$*, for each* $d \geq 2$

As a way of witnessing the utility of the presented framework, let us discuss how one of the non-compilability results from a previous paper [5] on the parameterized complexity of model checking can be formulated within this framework. Here, by a *unary signature*, we mean a signature containing only unary relation symbols. Define unary-EP-MC to be the problem of deciding, given a pair $(\phi, \mathbf{B})$ consisting of an existential positive sentence and a finite relational structure, each over the same unary signature, whether or not ϕ evaluates to true on $\mathbf{B}$ (see the paper [5] for definitions and background).

▶ **Proposition 31.** *The parameterized problem* $(\text{unary-EP-MC}, \pi_1)$ *is* chopped-NP*-hard.*

Proof. Let h be the reduction given in [5], which is a many-one polynomial-time reduction from the CNF satisfiability problem to unary-EP-MC where an instance having n variables and m clauses is mapped to an instance of the form $(S_n^m, \mathbf{B})$, where each S_n^m is a sentence. Let g be the map that, given a 3-SAT formula ϕ, eliminates duplicate clauses from ϕ and then maps the result under h. For a 3-SAT formula ϕ with n variables, it will thus hold that there exists a polynomial $C \in O(n^3)$ such that $\pi_1(g(\phi)) \in \{S_n^0, S_n^1, \ldots, S_n^{C(n)}\}$. If we define $s(n) = \{S_n^0, S_n^1, \ldots, S_n^{C(n)}\}$, we thus have that (g, s) is a poly-comp reduction from $(\text{3-SAT}, \nu)$ to $(\text{unary-EP-MC}, \pi_1)$, which yields the result by Theorem 30. ◀

Acknowledgements. This work was supported by the Spanish project TIN2013-46181-C2-2-R, by the Basque project GIU12/26, and by the Basque grant UFI11/45.

References

1 Simone Bova, Florent Capelli, Stefan Mengel, and Friedrich Slivovsky. Expander cnfs have exponential DNNF size. *CoRR*, abs/1411.1995, 2014.

2 Marco Cadoli, Francesco M. Donini, Paolo Liberatore, and Marco Schaerf. Preprocessing of intractable problems. *Information and Computation*, 176(2):89–120, 2002.

3 Hubie Chen. A theory of average-case compilability in knowledge representation. In *IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, Acapulco, Mexico, August 9–15, 2003*, pages 455–460, 2003.

4 Hubie Chen. Parameterized compilability. In *IJCAI-05, Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, Edinburgh, Scotland, UK, July 30-August 5, 2005*, pages 412–417, 2005.

5 Hubie Chen. On the complexity of existential positive queries. *ACM Trans. Comput. Log.*, 15(1), 2014.

6 Adnan Darwiche and Pierre Marquis. A knowledge compilation map. *J. Artif. Intell. Res. (JAIR)*, 17:229–264, 2002.

7 Wenfei Fan, Floris Geerts, and Frank Neven. Making queries tractable on big data with preprocessing. *PVLDB*, 6(9):685–696, 2013.

8 J. Flum and M. Grohe. *Parameterized Complexity Theory.* Springer, 2006.

9 Goran Gogic, Henry A. Kautz, Christos H. Papadimitriou, and Bart Selman. The comparative linguistics of knowledge representation. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, IJCAI 95, Montréal Québec, Canada*, pages 862–869, 1995.

10 Paolo Liberatore. The size of MDP factored policies. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence and Fourteenth Conference on Innovative Applications of Artificial Intelligence, July 28 – August 1, 2002, Edmonton, Alberta, Canada.*, pages 267–272, 2002.

11 Paolo Liberatore and Marco Schaerf. Compilability of propositional abduction. *ACM Trans. Comput. Log.*, 8(1), 2007.

12 Chee-Keng Yap. Some consequences of non-uniform conditions on uniform classes. *Theor. Comput. Sci.*, 26:287–300, 1983.