Parameterized Compilability

Hubie Chen
Departament de Tecnologia
Universitat Pompeu Fabra
Barcelona, Spain
hubie.chen@upf.edu

Abstract

Compilability is a measure of how effectively compilation (or preprocessing) can be applied to knowledge bases specified in a particular knowledge representation formalism; the aim of compilation is to allow for efficient, on-line query processing. A theory of compilability has been established for organizing knowledge representation formalisms according to a scheme of "compilability classes", and bears strong analogies to the classical theory of complexity, which permits the organization of computational problems according to complexity classes. We develop a novel theory of compilability, called *parameterized compilability*, which incorporates the notion of parameterization as used in parameterized complexity and permits for refined analysis of compilability.

1 Introduction

Compilability. Many computational reasoning problems involve deciding whether or not a query is entailed by a knowledge base (or database). Such reasoning problems are often computationally intractable. A methodology for coping with this intractability is *compilation* (also called *preprocessing*), which involves translating a knowledge base into a form that allows for more efficient query processing. Compilation is appropriate for knowledge bases that remain stable over time and will be queried numerous times, even if the compilation itself cannot be performed efficiently, since the cost of the compilation will be amortized by the speed-up in query processing. Although recent years have seen many interesting results on compilation, the basic idea of simplifying a sequence of computations by preprocessing a set of useful values is as old as mathematics, as discussed in [Cadoli *et al.*, 2002].

A systematic theory. A number of results in the nineties demonstrated non-compilability results, which show, for instance, that the knowledge bases of a particular knowledge representation formalism are not compilable to those of another formalism in which query processing can be performed in polynomial time. The initial results of this type, including [Cadoli *et al.*, 1996; 1997; 1999; Gogic *et al.*, 1995], were based on *ad hoc* proofs that did not explicitly use a unified proof technique.

In [Cadoli *et al.*, 2002], a robust theory of compilability was developed that made it possible to systematically organize knowledge representation formalisms according to their compilability, much in the way that classical complexity theory made it possible to systematically organize languages according to their (in)tractability. In particular, the theory of compilability provides a way to define, for every classical complexity class C, an analogous compilability class comp-C; the theory also provides a notion of reduction for use with the compilability classes comp-C. The compilability class comp-P, which is the analog of P (those languages decidable in polynomial time), contains those formalisms that are compilable to a second formalism in which query processing can be performed in polynomial time. In saying that a formalism is compilable to a second formalism, we mean (roughly) that for every knowledge base x of the first formalism, there is a knowledge base $f(x)$ in the second formalism having the same answers to queries as x and such that the size of $f(x)$ is always polynomial in the size of x. This "polynomial size" requirement models the intuition that compilation is only useful if it does not greatly increase the size of a knowledge base.

Demonstration that a formalism is in comp-P is a positive compilation result, analogous to demonstration that a language is in P, a positive tractability result. Likewise, demonstration that a formalism is comp-NP-hard is a negative compilation result and implies that the formalism is not in comp-P (under a complexity-theoretic assumption), just as demonstration that a language is NP-hard implies that the language is not in P (under the standard assumption that P $\neq$ NP).

First in [Cadoli *et al.*, 2002; 2000] and later in other papers including [Liberatore, 1997; Liberatore and Schaerf, 2000; Liberatore, 2000; 2001], the theory of compilability presented in [Cadoli *et al.*, 2002] was applied successfully to classify knowledge representation formalisms that have appeared in the artificial intelligence literature. In particular, a multitude of formalisms, including propositional logic, circumscription, and default logic, have been demonstrated to be contained in comp-P or complete for the analog comp-C of a standard complexity class C. The downside of these

classification results is that many of them are negative, showing that a formalism is complete for the compilability version of an intractable complexity class, such as comp-NP or comp-co-NP.

Parameterized complexity. The abundance of negative compilability results mirrors a state of affairs in complexity theory: although many problems of interest can be classified according to the developed scheme of complexity classes (P, NP, co-NP, etc.), there is an abundance of negative results that show completeness or hardness for classes believed to be strictly larger than P. Because such generally intractable problems do need to be dealt with in practice, a number of theoretical frameworks have been developed to provide a finer analysis of hard problems than that offered by standard complexity theory. Examples of such frameworks include the theory of approximation algorithms and average-case analysis. A relatively recent alternative framework is *parameterized complexity* [Downey and Fellows, 1999b].

Parameterized complexity is a theory that, as with classical complexity theory and compilability theory, provides classes and a notion of reduction for categorizing objects. In the case of parameterized complexity, the objects categorized are parameterized languages: languages with instances consisting of two parts, a *main part* and a *parameter*. Many languages that have been classically studied can be viewed naturally as parameterized languages. For example, each instance of the VERTEX COVER problem can be viewed as consisting of a graph G as the main part, and a natural number k as the parameter; the question is to decide whether or not the graph G has a vertex cover of size at most k. Similarly, the INDEPENDENT SET problem may be viewed as a parameterized language: an instance is again a graph G paired with a natural number k, and the question is to decide whether or not the graph G has an independent set of size at least k.

The parameterized flavor of tractability is called *fixed-parameter tractability*; formally, a parameterized language is fixed-parameter tractable (or, in the class FPT) if there exists a constant c such that for every k, instances with main part of size n and parameter k can be solved in time $O(n^c)$. As an example, VERTEX COVER can be solved in time $O(r^k k^2 + kn)$, where r is a constant, k is the parameter, and n is the size of the main part; consequently, for any fixed k, instances of VERTEX COVER with main part of size n and parameter k can be solved in time $O(n)$, and so VERTEX COVER is considered to be fixed-parameter tractable. Because the running time of fixed-parameter tractable languages exhibits a restricted form of dependence on the parameter, such languages are often solvable in practice when the parameter falls into a limited range. (In fact, one can see from the given time bound for VERTEX COVER that this problem can be solved in polynomial time when the parameter k is $O(\log n)$.) On the other hand, INDEPENDENT SET is believed to require time where the main part exhibits an exponential dependence on the parameter – that is, a time of form similar to $\Omega(n^k)$. Indeed, INDEPENDENT SET is complete for a parameterized complexity class believed to properly contain FPT, and accordingly is believed to be outside of FPT.

Viewed classically, these two example languages are both NP-complete, and hence demonstrate that parameterized complexity is somewhat orthogonal to classical complexity; moreover, the example of VERTEX COVER indicates that parameterized complexity can provide tractability results where classical complexity does not. Indeed, the allowance of a non-polynomial dependence on the parameter of an input instance in fixed-parameter tractability permits the use of algorithmic ideas not utilized in proving standard polynomial-time tractability results, and the field of parameterized complexity has developed a rich and mathematically deep suite of algorithmic techniques.

Parameterized compilability. The contribution of this paper is *parameterized compilability*, a new theory of compilability incorporating the notion of parameterization as used in parameterized complexity. Our theory provides a novel set of concepts for proving compilability results, allowing the use of the sophisticated toolkit that has been developed for proving fixed-parameter tractability and intractability results, and provides an avenue for refining the many negative non-compilability results that have been proved using the established theory of compilability. As articulated in [Downey *et al.*, 1999], one benefit of using a parameterized notion of computational tractability is that it becomes possible to engage in an extended dialogue with a single problem, by the investigation of a variety of parameters. We thus provide a theoretical framework in which we anticipate that the formulation and exploration of interesting parameters in the context of parameterized compilability will lead to refined analysis of compilability results.

The results in this paper are as follows. For every parameterized complexity class C, we define a compilability variant par-comp-C. We also define a notion of reduction for use with the par-comp-C classes, and demonstrate its robustness – in particular, that it is transitive and compatible (in a formal sense) with the classes par-comp-C. We also develop a general method for demonstrating hardness of a formalism for a class par-comp-C, and study the structure of the classes par-comp-C, relating this structure directly to the structure of non-uniform versions of parameterized complexity classes C.

2 Preliminaries

While we have attempted to make this paper as self-contained as possible, some familiarity with the basic notions of complexity theory (including non-uniform complexity classes), parameterized complexity, and compilability theory will be helpful; we name [Balcázar *et al.*, 1995], [Downey and Fellows, 1999b], and [Cadoli *et al.*, 2002] as references for these topics, respectively. We also recommend the surveys [Downey and Fellows, 1999a; Downey *et al.*, 1999] on parameterized complexity.

2.1 Definitions and Notation

Strings. We assume Σ to be a fixed finite alphabet that is used to form strings. The length of a string x is denoted by $|x|$. We will at times assume that pairs of strings (that

is, elements of $\Sigma^* \times \Sigma^*$) are represented as strings (that is, elements of Σ^*) via a pairing function $\langle \cdot, \cdot \rangle$. We make standard assumptions about this pairing function, namely, that the length of $\langle x, y \rangle$ is linear in $|x| + |y|$; $\langle x, y \rangle$ can be computed in time polynomial in $|x| + |y|$; there are polynomial-time computable projection functions $\pi_1, \pi_2 : \Sigma^* \to \Sigma^*$ such that $\pi_1(\langle x, y \rangle) = x$ and $\pi_2(\langle x, y \rangle) = y$; and $\langle \cdot, \cdot \rangle$ is length-monotone, that is, if $|y| \leq |z|$ then $|\langle x, y \rangle| \leq |\langle x, z \rangle|$ and $|\langle y, x \rangle| \leq |\langle z, x \rangle|$. We also assume that there is a tripling function $\langle \cdot, \cdot, \cdot \rangle$ for representing triples of strings as single strings, for which similar assumptions hold.

Functions. As usual, we consider a function $p : \mathbb{N} \to \mathbb{N}$ to be a *polynomial* if there exists a constant $c \in \mathbb{N}$ such that $p(n)$ is $O(n^c)$. A function $p : \mathbb{N} \times \Sigma^* \to \mathbb{N}$ is a *parameterized polynomial* if there exists a constant $c \in \mathbb{N}$ and a function $h : \Sigma^* \to \mathbb{N}$ such that for all $(n, k) \in \mathbb{N} \times \Sigma^*$, $p(n, k) \leq h(k)(n + 1)^c$. In other words, $p : \mathbb{N} \times \Sigma^* \to \mathbb{N}$ is a parameterized polynomial if there exists a constant $c \in \mathbb{N}$ such that for each $k \in \Sigma^*$, the function $p_k : \mathbb{N} \to \mathbb{N}$ defined by $p_k(n) = p(n, k)$ is $O(n^c)$.

A function $f : \Sigma^* \to \Sigma^*$ is *polynomial-size* if there exists a polynomial $p : \mathbb{N} \to \mathbb{N}$ such that for all $x \in \Sigma^*$, $|f(x)| \leq p(|x|)$. Similarly, a function $f : \Sigma^* \times \Sigma^* \to \Sigma^*$ is *parameterized polynomial-size* if there exists a parameterized polynomial $p : \mathbb{N} \times \Sigma^* \to \mathbb{N}$ such that for all $(x, k) \in \Sigma^* \times \Sigma^*$, $|f(x, k)| \leq p(|x|, k)$.

A function $g : \Sigma^* \to \Sigma^*$ is *polynomial-time computable* if there exist a polynomial $p : \mathbb{N} \to \mathbb{N}$ and a Turing machine M such that for all $x \in \Sigma^*$, the Turing machine M, on input x, produces output $g(x)$ in time less than $p(|x|)$. Similarly, a function $g : \Sigma^* \times \Sigma^* \to \Sigma^*$ is *parameterized polynomial-time computable* if there exist a parameterized polynomial $p : \mathbb{N} \times \Sigma^* \to \mathbb{N}$ and a Turing machine M such that for all $(x, k) \in \Sigma^* \times \Sigma^*$, the Turing machine M, on input $\langle x, k \rangle$, produces output $g(x, k)$ in time less than $p(|x|, k)$.

A function $h : \Sigma^* \to \Sigma^*$ is *recursively computable* if there exists a Turing machine M such that for all $x \in \Sigma^*$, the Turing machine M, on input x, produces output $h(x)$.

Languages and Complexity Classes. A *language* is a subset of Σ^*, that is, a set of strings. A *parameterized language* is a subset of $\Sigma^* \times \Sigma^*$, that is, a set of pairs of strings. A *(parameterized) complexity class* is a set of (parameterized) languages. When C is a (parameterized) complexity class and $\leq$ is a reduction, we say that C is *compatible* with $\leq$ if for all (parameterized) languages A and B, $A \leq B$ and $B \in C$ imply that $A \in C$. Also, when C is a (parameterized) complexity class and $\leq$ is a reduction, we say that B is *hard* for C (alternatively, is C-*hard*) under $\leq$ reductions if for all $A \in C$, $A \leq B$; and, we say that B is *complete* for C (alternatively, is C-*complete*) under $\leq$ reductions if $B \in C$ and B is C-hard.

When C is a complexity class, the non-uniform version (or "advice" version) of C, denoted by C/poly, contains those languages A such that there exists a polynomial-size function $f : 1^* \to \Sigma^*$ and a language B in C such that for all $x \in \Sigma^*$, $x \in A$ if and only if $\langle f(1^{|x|}), x \rangle \in B$. When C is a parameterized complexity class, the non-uniform version of C, denoted by C/ppoly, contains those parameterized languages A such that there exists a parameterized polynomial-size function $f : 1^* \times \Sigma^* \to \Sigma^*$ and a language B in C such that for all $(x, k) \in \Sigma^* \times \Sigma^*$, $(x, k) \in A$ if and only if $(\langle f(1^{|x|}, k), x \rangle, k) \in B$.

When L is a parameterized language, define ϵL to be the set $\{(\epsilon, y, k) : (y, k) \in L\}$.

Knowledge Representation Formalisms. A *knowledge representation formalism (KRF)* is a subset of $\Sigma^* \times \Sigma^*$. A *parameterized knowledge representation formalism (PKRF)* is a subset of $\Sigma^* \times \Sigma^* \times \Sigma^*$.

Conventions. We view a ternary function $f : \Sigma^* \times \Sigma^* \times \Sigma^* \to \Sigma^*$ as a binary function $g : \Sigma^* \times \Sigma^* \to \Sigma^*$ by pairing together the first two arguments, that is, $g(\langle x, y \rangle, z) = f(x, y, z)$. For example, a ternary function $f : \Sigma^* \times \Sigma^* \times \Sigma^* \to \Sigma^*$ is considered to be parameterized polynomial-size if there exists a parameterized polynomial $p : \mathbb{N} \times \Sigma^* \to \mathbb{N}$ such that for all $(x, y, k) \in \Sigma^* \times \Sigma^* \times \Sigma^*$, $|f(x, y, k)| \leq p(|\langle x, y \rangle|, k)$. Similarly, a PKRF F is viewed as a parameterized language by pairing together the first two strings of each triple, and so is considered to belong to a parameterized complexity class C if the language $L = \{(\langle x, y \rangle, k) : (x, y, k) \in F\}$ belongs to C.

For readability, we at times use a natural number $m \in \mathbb{N}$ in place of its unary representation 1^m.

2.2 Compilability

In this subsection, we review the theory of compilability; the definitions and theorems of this subsection are all from [Cadoli *et al.*, 2002], on which our presentation is based. Throughout this subsection, the following two assumptions concerning complexity classes and languages are in effect, just as in [Cadoli *et al.*, 2002].

Assumption 1 *Every complexity class C is compatible with and has complete problems under the polynomial-time many-one reduction $\leq_m^p$.*

Assumption 2 *In languages, suffixed blanks are considered to be irrelevant, that is, a string $x\Box$ is considered to be in a language A if x is in A.*

For a KRF F, we will refer to each $x \in \pi_1(F)$ as a *knowledge base*; a knowledge base x is considered to represent the knowledge $\{y \in \Sigma^* : (x, y) \in F\}$. Roughly speaking, a KRF is considered to be compilable to a complexity class C if following a preprocessing of each knowledge base x, posing a query y to the knowledge base x – that is, deciding if $(x, y) \in F$ – can be performed in C. The formal definition of those KRFs compilable to a complexity class C includes two ingredients. The first is a preprocessing function f; because preprocessing of a knowledge base is not useful if it greatly increases the size of the knowledge base, f is required to be of polynomial size. The second is a "target" KRF F', mandated to be inside the class C, to which the KRF F is translated via the function f.

Definition 3 *(comp-C) Let C be a complexity class. A KRF F belongs to comp-C if there exist*

- *a binary polynomial-size function* $f : \Sigma^* \times 1^* \rightarrow \Sigma^*$, *and*
- *a KRF* F' *in* C

such that for all pairs $(x, y) \in \Sigma^* \times \Sigma^*$,

$$(x, y) \in F \text{ if and only if } (f(x, |y|), y) \in F'.$$

One can obtain a compilability class comp-C from any complexity class C. Hence, the comp-C classes offer a rich scheme by which one can classify KRFs, just as conventional complexity classes offer a rich scheme by which one can classify computational problems. Upon a first reading of Definition 3, it may be conceptually easiest for one to take the complexity class C to be equal to P (polynomial-time). In this case, the requirement that F' is in C means that queries (x, y) posed to F' can be processed efficiently, so f translates each F-knowledge base x – which is not necessarily in a form that allows for efficient resolution of queries (x, y) – into a F'-knowledge base $f(x)$ – which *is* in a form that allows for efficient resolution of queries (x, y).[1]

The following notion of reduction, associated with the comp-C classes, allows one to compare the compilability of different KRFs.

Definition 4 *(comp reducibility) A KRF* F *is comp reducible to a KRF* F' *(denoted by* $F \leq_{\text{comp}} F'$*) if there exist*

- *binary polynomial-size functions* $f_1, f_2 : \Sigma^* \times 1^* \rightarrow \Sigma^*$ *and*
- *a binary polynomial-time computable function* $g : \Sigma^* \times \Sigma^* \rightarrow \Sigma^*$

such that for all pairs $(x, y) \in \Sigma^* \times \Sigma^*$,

$$(x, y) \in F \text{ if and only if } (f_1(x, |y|), g(f_2(x, |y|), y)) \in F'.$$

Theorem 5 *For every complexity class* C*, the comp reduction is transitive and is compatible with the class* comp-C.

The structure of the compilability classes comp-C can be directly related to the structure of the complexity classes C/poly.

Theorem 6 *Let* C *and* C' *be complexity classes. The containment* comp-$C \subseteq$ comp-C' *holds if and only if the containment* C/poly $\subseteq C'$/poly *holds.*

When C and C' are complexity classes that obey the proper containment $C \subsetneq C'$, Theorem 6 gives strong evidence that the corresponding compilability classes comp-C and comp-C' obey a similar proper containment: comp-$C \subsetneq$ comp-C'. Note that in making this observation, we appeal to the widely held belief that relationships between "standard" complexity classes C, C' are not affected by non-uniformity, that is, C is properly contained in C' if and only if C/poly is properly contained in C'/poly.[2]

[1] This definition of comp-C is "non-uniform" in that the preprocessing or translation function f not only takes the knowledge base x as input, but in addition is given the length of the query y. A "uniform" version of the classes comp-C where the translation function f is only given the knowledge base x is studied and compared to the comp-C classes in [Cadoli *et al.*, 2002]. In this paper, we present a non-uniform theory of parameterized compilability, and do not further discuss the uniform comp-C classes. One can use the ideas and concepts we present in this paper to develop a corresponding uniform theory.

2.3 Parameterized Complexity

Parameterized complexity is a theory for the classification of parameterized languages – languages whose constituents have two components. In classical complexity theory, the base complexity class modeling feasible computation is P, which is defined using the notion of polynomial-time; in parameterized complexity theory the base complexity class is FPT ("fixed-parameter tractable"), which is definable using the notion of parameterized polynomial-time.

Definition 7 *(FPT) A parameterized language* L *belongs to the parameterized complexity class FPT if there exists a parameterized polynomial-time computable function* $g : \Sigma^* \times \Sigma^* \rightarrow \{0, 1\}$ *such that for all pairs* $(x, k) \in \Sigma^* \times \Sigma^*$,

$$(x, k) \in L \text{ if and only if } g(x, k) = 1.$$

Roughly speaking, a parameterized language L is in the class FPT if there exists a constant $c \in \mathbb{N}$ such that each "k-slice" $L_k \stackrel{\text{def}}{=} \{x : (x, k) \in L\}$ of L is computable in polynomial time via a polynomial of degree c. The degree of the polynomials bounding the time for L_k must be a "universal" constant c and cannot depend on k; on the other hand, the coefficients of these polynomials may have arbitrary dependence on k.

The following notion of reducibility may be used to compare the complexity of parameterized languages.

Definition 8 *(parameterized reducibility) A parameterized language* L *is* parameterized reducible *to a parameterized language* L' *(denoted by* $L \leq^{\text{par}} L'$*) if there exist*

- *a parameterized polynomial-time computable function* $f : \Sigma^* \times \Sigma^* \rightarrow \Sigma^*$*, and*
- *a recursively computable function* $h : \Sigma^* \rightarrow \Sigma^*$

such that for all pairs $(x, k) \in \Sigma^* \times \Sigma^*$,

$$(x, k) \in L \text{ if and only if } (f(x, k), h(k)) \in L'.$$

We now define two parameterized complexity classes. In contrast to the class FPT, each of these two classes is defined directly as those languages reducible to a particular language.

Definition 9 *We define the following parameterized languages; the* weight *of a truth assignment is defined to be the number of variables it maps to true.*

- WEIGHTED FORMULA SATISFIABILITY
  Input: A boolean formula F.
  Question: Does F *have a satisfying assignment of weight at most* k*?*
- WEIGHTED CIRCUIT SATISFIABILITY
  Input: A boolean circuit C.
  Question: Does C *have a satisfying assignment of weight at most* k*?*

[2] Note that the term "non-uniformity" is intended here to take on the usual complexity-theoretic sense, as opposed to the slightly different sense in which comp-C was called non-uniform.

Definition 10 *We define the following parameterized complexity classes.*

- *W[SAT] is the class of all parameterized languages that are parameterized reducible to* WEIGHTED FORMULA SATISFIABILITY .
- *W[P] is the class of all parameterized languages that are parameterized reducible to* WEIGHTED CIRCUIT SATISFIABILITY .

It is straightforward to verify that the parameterized complexity classes defined here (FPT, W[SAT], and W[P]) are compatible with the parameterized reduction. Throughout this paper, we will assume that all discussed parameterized complexity classes have this property, and also that they have complete problems.

Assumption 11 *Every parameterized complexity class C is compatible with and has complete problems under the parameterized reduction $\leq^{\mathrm{par}}$.*

3 Parameterized Compilability

3.1 Classes and reductions

As with the theory of compilability, the starting point for the theory of parameterized compilability is the definition of compilability classes. Whereas a compilability version of each non-parameterized complexity class was defined (Definition 3), here we define a compilability version par-comp-C of each *parameterized* complexity class C.

As with the definition of comp-C, the definition of par-comp-C includes a translation function f with a space bound, as well as a "target" PKRF F', which is required to be inside C.

Definition 12 *Let C be a parameterized complexity class. A PKRF F belongs to par-comp-C if there exist*

- *a parameterized polynomial-size function $f : \Sigma^* \times 1^* \times \Sigma^* \rightarrow \Sigma^*$, and*
- *a PKRF F' in C*

such that for all triples $(x, y, k) \in \Sigma^* \times \Sigma^* \times \Sigma^*$ and natural numbers $m \geq |y|$,

$$(x, y, k) \in F \text{ if and only if } (f(x, m, k), y, k) \in F'.$$

An implication of the containment of a PKRF F inside par-comp-C via the function f and the PKRF F' is the following. Define, for each $k \in \Sigma^*$, the "k-slice" of a PKRF G to be $G_k \stackrel{\text{def}}{=} \{(x, y) : (x, y, k) \in G\}$. Moreover, define $f_k : \Sigma^* \times 1^* \rightarrow \Sigma^*$ to be the function $f_k(x, m) = f(x, m, k)$. Notice that for each $k \in \Sigma^*$, the KRFs F_k, F'_k and the binary function f_k satisfy the condition of Definition 3. That is, each k-slice F_k of F is compilable to the corresponding k-slice F'_k of F' by the function f_k. Because the size of f is bounded by a parameterized polynomial, for k falling into a limited range, the functions f_k will behave polynomially; likewise, when $F' \in$ FPT, for k falling into a limited range, the KRF slices F'_k will have time complexity behaving polynomially.

Upon initial acquaintance, Definition 12 may appear to be complex. However, we believe that the classes par-comp-C unify the classes comp-C with parameterized complexity classes in a natural way. We substantiate this claim by considering two special cases of Definition 12, which may lend some intuition to the reader.

First, consider the case where the preprocessing function f simply returns as output its first argument, that is, f is equal to the projection π_1. It follows that F must equal F'; consequently, a PKRF F is contained in par-comp-C via such an f if and only if F is contained in C itself. Intuitively, the function f does not perform any preprocessing, so the compilability class par-comp-C simplifies out to the original class C. Next, consider the case where the PKRF F has the property that for all strings $x, y, k, k' \in \Sigma^*$, it holds that $(x, y, k) \in F$ if and only if $(x, y, k') \in F$; that is, F "ignores" the third component (the "parameter") of its strings. In this case, the PKRF F naturally induces a KRF $\tilde{F}$, defined to be

$$\{(x, y) : \exists k.(x, y, k) \in F\} = \{(x, y) : \forall k.(x, y, k) \in F\}.$$

Such a PKRF F is contained in par-comp-FPT if and only if the induced KRF $\tilde{F}$ is contained in comp-P.[3] In "ignoring" the parameter of the PKRF F, the compilability class par-comp-FPT simplifies out to its non-parameterized variant, namely, comp-P. We mention that by imposing the restrictions discussed in the two cases simultaneously, we effectively obtain the class P. Thus, one can think of FPT and comp-P as two different generalizations of P which are unified together in and can be obtained as specializations of par-comp-FPT.

We have the following notion of reducibility for comparing the compilability of PKRFs.

Definition 13 *(par-comp reducibility) A PKRF F is par-comp-reducible to a PKRF F' (denoted by $F \leq^{\mathrm{par}}_{\mathrm{comp}} F'$) if there exist*

- *parameterized polynomial-size functions $f_1, f_2 : \Sigma^* \times 1^* \times \Sigma^* \rightarrow \Sigma^*$,*
- *a parameterized polynomial-time function $g : \Sigma^* \times \Sigma^* \times \Sigma^* \rightarrow \Sigma^*$, and*
- *a recursively computable function $h : \Sigma^* \rightarrow \Sigma^*$*

such that for all triples $(x, y, k) \in \Sigma^* \times \Sigma^* \times \Sigma^*$ and natural numbers $m \geq |y|$,

$$(x, y, k) \in F \text{ if and only if}$$
$$(f_1(x, m, k), g(f_2(x, m, k), y, k), h(k)) \in F'.$$

We note that our definition of par-comp reducibility is not a direct analog of the definition of comp-reducibility; in Definition 13 the "preprocessing phase" f is given an upper bound m on the size of the query y, as opposed to the length of the query $|y|$, as in Definition 4. We believe that our notion of reduction is preferable because an assumption like Assumption 2 is not necessary to establish transitivity, and also because it allows for a simpler proof of transitivity. Our proof of transitivity can be specialized to a proof of transitivity for an "upper bound" variant of Definition 4 by removing the parameter k.

[3] Sketch of one direction of this claim: suppose that the PKRF F is of the described form and in par-comp-FPT. Fix an arbitrary $s \in \Sigma^*$, define the function $\tilde{f} : \Sigma^* \times 1^* \rightarrow \Sigma^*$ by $\tilde{f}(x, m) = f(x, m, s)$, and define the KRF $\tilde{F'}$ as $\{(x, y) : (x, y, s) \in F'\}$. It can be verified that $\tilde{F}$ is contained in comp-P via $\tilde{f}$ and $\tilde{F'}$.

Theorem 14 *The par-comp reduction is transitive.*

Theorem 15 *For every parameterized complexity class C, the par-comp reduction is compatible with the class* par-comp-C.

3.2 Completeness and Structure

We now study the notion of completeness for the classes par-comp-C, as well as the structure of the classes par-comp-C. First, we make a simple observation about the relationship between C/ppoly and par-comp-C which will aid us in our study.

Proposition 16 *Let C be a parameterized complexity class. If the parameterized language A is in C/ppoly, then ϵA is in* par-comp-C.

The following theorem, which will be of help in proving hardness of PKRFs, demonstrates that we can easily obtain complete languages for the class par-comp-C from complete languages for the underlying class C.

Theorem 17 *Let C be a parameterized complexity class. If the parameterized language B is complete for C (under $\leq^{\mathrm{par}}$ reductions), then ϵB is complete for* par-comp-C *(under $\leq^{\mathrm{par}}_{\mathrm{comp}}$ reductions).*

The next theorem is an analog of Theorem 6, and correspondingly provides strong evidence that the structure of the compilability classes par-comp-C reflects the structure of the parameterized complexity classes C.

Theorem 18 *Let C and C' be parameterized complexity classes. The containment* par-comp-$C \subseteq$ par-comp-C' *holds if and only if the containment C/ppoly $\subseteq C'$/ppoly holds.*

References

[Balcázar *et al.*, 1995] J. L. Balcázar, J. Díaz, and J. Gabarró. *Structural Complexity I*. Texts in Theoretical Computer Science – An EATCS series. Springer-Verlag, Berlin, 2nd edition, 1995.

[Cadoli *et al.*, 1996] M. Cadoli, F.M. Donini, and M. Schaerf. Is intractability of non-monotonic reasoning a real drawback? *Artificial Intelligence*, 88:215–251, 1996.

[Cadoli *et al.*, 1997] M. Cadoli, F. M. Donini, M. Schaerf, and R. Silvestri. On compact representations of propositional circumscription. *Theoretical Computer Science*, 182:183–202, 1997.

[Cadoli *et al.*, 1999] Marco Cadoli, Francesco M. Donini, Paolo Liberatore, and Marco Schaerf. The size of a revised knowledge base. *Artificial Intelligence*, 115(1):25–64, 1999.

[Cadoli *et al.*, 2000] Marco Cadoli, Francesco M. Donini, Paolo Liberatore, and Marco Schaerf. Space efficiency of propositional knowledge representation formalisms. *Journal of Artificial Intelligence Research*, 13:1–31, 2000.

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

[Downey and Fellows, 1999a] R. Downey and M. Fellows. Parameterized complexity after (almost) 10 years: Review and open questions. In *Combinatorics, Computation and Logic, DMTCS'99 and CATS'99*, pages 1–33. Springer-Verlag, 1999.

[Downey and Fellows, 1999b] R.G. Downey and M.R. Fellows. *Parameterized Complexity*. Springer-Verlag, 1999.

[Downey *et al.*, 1999] R. Downey, M. Fellows, and U. Stege. Parameterized complexity: A framework for systematically confronting computational intractability. In *Contemporary Trends in Discrete Mathematics, (R. Graham, J. Kratochvil, J. Nesetril and F. Roberts, eds.), Proc. DIMACS-DIMATIA Workshop, Prague 1997*, volume 49 of *AMS-DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 49–99, 1999.

[Gogic *et al.*, 1995] Goran Gogic, Henry A. Kautz, Christos Papidimitriou, and Bart Selman. The comparative linguistics of knowledge representation. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 862–869, 1995.

[Liberatore and Schaerf, 2000] Paolo Liberatore and Marco Schaerf. Compilability of abduction. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI 2000)*, 2000.

[Liberatore, 1997] Paolo Liberatore. Compilability of domain descriptions in the language A. *Electronic Transactions on Artificial Intelligence*, 1(4):129–132, 1997.

[Liberatore, 2000] Paolo Liberatore. Compilability and compact representations of revision of horn knowledge bases. *ACM Transactions on Computational Logic*, 1(1):131–161, 2000.

[Liberatore, 2001] Paolo Liberatore. Monotonic reductions, representative equivalence, and compilation of intractable problems. *Journal of the ACM*, 48(6):1091–1125, 2001.