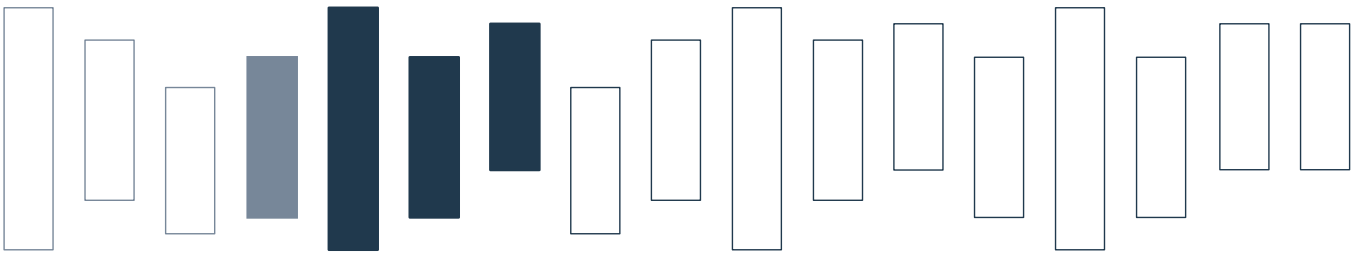
Technical Report AC-TR-16-003

April 2016

Positive and Negative Results for Parameterized Compilability

Simone Bova, Ronald de Haan, Neha Lodha, and Stefan Szeider



Positive and Negative Results for Parameterized Compilability

Simone Bova, Ronald de Haan, Neha Lodha, Stefan Szeider
Algorithms and Complexity Group, TU Wien, Vienna, Austria

Abstract

Most relevant computational tasks are not polynomial-size compilable, even though they are compiled rather efficiently in practice. Parameterized compilability offers a framework to narrow this gap between theoretical incompilability and practical compilation by relaxing the size bound posed on compilations.

We study various computational problems (related to clause entailment, the classical problem of checking whether a CNF formula entails a clause) in the framework of parameterized compilability. We establish several positive and negative results. On the positive side we show, for instance, that clause entailment is fixed-parameter compilable parameterized by the minimum incidence treewidth over all equivalent CNF formulas. On the negative side we show, for instance, that the size of any compilation of the clause entailment problem restricted to clauses of bounded size has an exponential dependency on the size bound posed on the clauses.

Technical Report AC-TR-16-003

1 Introduction

Most reasoning tasks in artificial intelligence involve querying some knowledge base. These tasks are usually computationally intractable. The idea of *knowledge compilation* is to deal with this intractability using the fact that, in many applicative scenarios, the knowledge base is known in advance and is queried multiple times. The knowledge base can therefore be compiled (that is, preprocessed in an offline, typically computationally hard phase) to obtain an equivalent knowledge base verbose enough to support upcoming queries in polynomial time. To support practical use of this method the compilation must be succinct, which is typically enforced by asking for the compiled knowledge to have *polynomial size* in the size of the original knowledge base.

Unfortunately, most relevant reasoning tasks do not admit a compilation having polynomial size in the worst case. Selman and Kautz [24] first observed that a fundamental such task is the *clause entailment* problem (deciding whether a CNF formula entails a clause), a result that started the theoretical investigation of the compilability of intractable problems, systematized by Cadoli et al. [3] within *classical compilability theory*. At the same time, researchers have developed many compilation algorithms that can be used to transform CNF formulas into different representations in order to facilitate entailment queries to be performed tractably. These compilation algorithms produce exponential-size results in the worst case, but can be used efficiently in practice [6, 8, 14, 17, 21, 22].

This discrepancy between theoretical incompilability and practical algorithms leads to a gap between theory and practice. Chen [4] proposed and developed the idea of using *parameterized complexity theory* to narrow this gap, an idea recently revisited by De Haan [13]. In parameterized complexity one measures the instances of a computational problem not only by their size but also by a *parameter*, and aims for *fixed-parameter tractable algorithms*, whose running time depends polynomially on the size and arbitrarily (say, exponentially) on the parameter [9, 10, 12, 20]. Intuitively, a fixed-parameter tractable running time is computationally feasible for small parameter values.

Fixed-parameter compilability naturally arises in this setting as the relaxation of polynomial-size compilability obtained by allowing an arbitrary dependence on the parameter in the compilation size; it presents itself as the central notion of a *parameterized compilability theory*, which generalizes both classical compilability and parameterized complexity.[1] Parameters can be ordered with respect to their

[1] Recently, the combination of knowledge compilation and parameterized complexity has been studied from a different angle [5]. Unlike this paper and Chen's earlier work [4], where the notion of compilability is extended by means of a parameterized perspective, Chen's recent work [5] investigates the (classical) theory of compilability using parameterized complexity tools.

generality in terms of the *dominance relation*, such that any positive fixed-parameter compilability result with respect to one parameter extends to any parameter that it dominates, and negative results extend in the other direction.

Contribution. In this paper, we investigate the potential and limits of parameterized compilability by establishing parameterized compilation and parameterized incompilability results for several compilation problems related to clause entailment. Under standard assumptions in complexity theory, not only is clause entailment not polynomial-time tractable but, as already mentioned, it also does not admit a polynomial-size compilation [24].

On the other hand, clause entailment is (unsurprisingly) fixed-parameter compilable under basic parameterizations where the satisfiability problem is fixed-parameter tractable, for instance incidence treewidth. We therefore consider parameterizations of CNF formulas for which the satisfiability problem is *not* fixed-parameter tractable (under standard assumptions in complexity theory, see Proposition 3). In particular, we consider *incidence treewidth modulo equivalence* and *incidence treewidth modulo backbones*, the former strictly dominating the latter (which, in turn, strictly dominates incidence treewidth), Therefore fixed-parameter compilability results for these (and alike) parameterizations effectively extend the realm of feasibility indicated by polynomial-size compilability and fixed-parameter tractability for the clause entailment problem.

Relative to the above two incidence treewidth measures, we obtain positive results of two types.

(1) We prove that clause entailment parameterized by incidence treewidth modulo equivalence is fixed-parameter *linear* compilable (Theorem 1). Here we exploit an algorithm by Oztok and Darwiche [21] that compiles CNF formulas into DNNFs (decomposable negation normal form, [7]) of fixed-parameter tractable size with respect to incidence treewidth.

We also prove that this compilation result is *optimal* in the sense that clause entailment is not fixed-parameter *sublinear* compilable under "natural parameterizations" (Proposition 4), including incidence treewidth modulo equivalence.

(2) By the dominance relation, the previous fixed-parameter linear compilability result extends to the parameter incidence treewidth modulo backbones (Corollary 2). The special interest of incidence treewidth modulo backbones relies in its being, in contrast to incidence treewidth modulo equivalence, *practically measurable* using existing tools.

We therefore can present CNF formulas, originating from practical settings, where the incidence treewidth modulo backbones is *significantly smaller* than the incidence treewidth (Table 1), indicating that the considered parameters are stronger not only in theory, but also in practice.

We contrast the above contributions on the positive side with two contributions on the negative side.

(3) Using the parameterized incompilability framework by Chen [4], we show that clause entailment is not fixed-parameter compilable (a) when parameterized by the size of the smallest backdoor to the fragment of CNF formulas whose satisfiability can be decided by pure literal elimination (Proposition 5), nor (b) when parameterized by the weight of assignments for a notion of entailment restricted to assignments of bounded weight (Proposition 6).

(4) Building forth on the recent work of De Haan [13], we show that the clause entailment problem is not fixed-parameter compilable when parameterized by the size of clauses for which entailment is to be checked (Proposition 7).

2 Parameterized Compilation

Throughout the paper, Σ denotes an alphabet (a nonempty finite set), Σ^* denotes the set of strings over Σ, and $|x|$ denotes the length of $x \in \Sigma^*$. For technical reasons, we assume that Σ contains a designated symbol, $\square$, called *placeholder*. We freely view pairs of strings $(x, y) \in \Sigma^* \times \Sigma^*$ as represented by strings in Σ^* by a (reasonable) encoding.

Parameterized Problems. Let $g\colon \Sigma^* \to \Sigma^*$ be a partial function. We say that g is *computable* if there is an algorithm A computing g for all x in the domain of g. We say that g is *poly-time computable* if A runs in at most $\text{poly}(|x|)$ time.[2]

[2]The behavior of A in terms of correctness and termination is arbitrary on instances $x \in \Sigma^*$ not in the domain of g.

Technical Report AC-TR-16-003

A *parameterization* is a computable function $k\colon \Sigma^* \to \mathbb{N}$.[3] Parameterizations are partially ordered as follows: $k' \leq k$ (in words, k *dominates* k') if and only if there exists a computable function g over nonnegative integers such that $k(x) \leq g(k'(x))$ for all $x \in \Sigma^*$. We say that k *strictly dominates* k' if k dominates k' and k' does not dominate k.

We say that g is *fpt-time computable* with respect to a parameterization k if there exist a computable function f and a polynomial function p, both over nonnegative integers, and an algorithm A computing g in time at most $f(k(x))p(|x|)$, for all $x \in \Sigma^*$ in the domain of g.

A *(decision) problem* is a language $L \subseteq \Sigma^*$. An algorithm A (on input alphabet Σ) *decides* a problem L if A computes the characteristic function of L. A problem L is *polynomial-time decidable* if the characteristic function of L is poly-time computable. A *parameterized problem* is a pair (L, k) where $L \subseteq \Sigma^*$ is a problem and k is a parameterization that is fpt-time computable with respect to itself. A parameterized problem (L, k) is *fixed-parameter tractable* with respect to the parameterization k if the characteristic function of L is fpt-time computable with respect to k.

For convenience, we make the following technical assumptions on the considered problems. For every problem $L \subseteq \Sigma^*$, we assume that $x \in L$ if and only if $xs \in L$, for any $s \in \{\square\}^*$; in words, we assume that suffixing placeholders is irrelevant.

The assumption is justified by the observation that if the problem L', obtained by deleting placeholder suffixes from strings in L, is decidable in time $t(|x|)$ by an algorithm A', then L is decidable in time $O(|xs|) + t(|x|)$ by an algorithm A that reads the input xs, deletes the placeholder suffix, and invokes A' on x. Similarly, if $L \subseteq \Sigma^* \times \Sigma^*$, we assume that $(x, y) \in L$ if and only if $(xr, ys) \in L$, where $r, s \in \{\square\}^*$.

Analogously, for every parameterized problem (L, k), we assume that $x \in L$ if and only if $xs \in L$, where $s \in \{\square\}^*$, and that $k(xs) = k(x)$ for every $s \in \{\square\}^*$. It is readily verified that if (L', k) is decidable in time $t(k(x))t'(|x|)$, then (L, k) is decidable in time $O(|xs|) + t(k(x))t'(|x|)$.

Parameterized Compilation Problems. Throughout the paper, we refer to a problem C consisting of pairs, $C \subseteq \Sigma^* \times \Sigma^*$, as a *compilation problem*; moreover, we view pairs $(x, y) \in \Sigma^* \times \Sigma^*$ as instances of a compilation problem, and call x and y, respectively, the *offline* and *online* instance.

A function $c\colon \Sigma^* \to \Sigma^*$ is *poly-size* if there exists a polynomial function p over nonnegative integers such that $|c(x)| \leq p(|x|)$ for all $x \in \Sigma^*$. A compilation problem C is "compilable" in the classical sense if it has a *poly-size compilation (into a polynomial-time computable problem)*, that is, if there is a computable function $c\colon \Sigma^* \to \Sigma^*$ (a *compilation*) and a polynomial-time computable problem L such that c is poly-size and $(x, y) \in C$ if and only if $(c(x), y) \in L$.

A *parameterized compilation problem* is a pair (C, k) where C is a compilation problem and k is a parameterization that only depends on the offline instance, that is, $k((x, y)) = k(x)$ for all $(x, y) \in \Sigma^* \times \Sigma$. A function $c\colon \Sigma^* \to \Sigma^*$ is called *fpt-size* with respect to a parameterization k if there exist a computable function f and a polynomial function p, both over nonnegative integers, such that $|c(x)| \leq f(k(x))p(|x|)$ for all $x \in \Sigma^*$. If p is a linear function, we call c *fpt-linear-size*.

Fpt-Size Compilability. We are now ready to define the notion of a "compilable" problem in the parameterized setting.

Definition 1. *Let (C, k) be a parameterized compilation problem. We say that (C, k) is* fpt-size compilable *if there exist a computable function $c\colon \Sigma^* \to \Sigma^*$ (a* compilation*) and a poly-time problem L such that c is fpt-size with respect to k and $(x, y) \in C$ if and only if $(c(x), y) \in L$.*

Let (C, k) and (C, k') be parameterized compilation problems. It is readily verified that, if k dominates k' and (C, k) has an fpt-size compilation, then so does (C, k').

In our formalization, both the classical and parameterized frameworks refer to the same notion of compilation, and require an unparameterized, poly-time tractable problem in the online phase. Nevertheless, allowing fpt-time in the online phase does not deliver additional computational power, in the following sense.

Proposition 1. *Let (C, k) be a parameterized compilation problem. Then (C, k) is fpt-size compilable if and only if there exist an fpt-size compilation c, a fixed-parameter tractable problem (L, k'), and a computable function g such that, for all $(x, y) \in \Sigma^* \times \Sigma^*$, it holds that $k'(c(x), y) \leq g(k(x))$ and $(x, y) \in C$ if and only if $(c(x), y) \in L$.*

[3] In [12], parameterizations are denoted by κ instead of k; here, we reserve lowercase greek letters for formulas.

Technical Report AC-TR-16-003

Proof. The forward direction is clear. For the backward direction, note that there exist functions f and h over nonnegative integers (without loss of generality, h is increasing) such that, for all $(x, y) \in \Sigma^* \times \Sigma^*$, it holds that

$$|c(x)| \leq f(k(x))\text{poly}(|x|)$$

and there exists an algorithm A deciding whether $(c(x), y)$ is in L in time at most

$$h(k'(c(x), y))\text{poly}(|(c(x), y)|);$$

hence, putting $h' = h \circ g$, A decides $(c(x), y) \in L$ in time at most

$$h'(k(x))\text{poly}(|(c(x), y)|).$$

It follows that there exist a computable function h'' and polynomials p_1 and p_2 (h'', p_1, and p_2 over nonnegative integers, p_1 increasing) such that A decides $(c(x), y) \in L$ in time at most

$$h''(k(x)) + p_1(|c(x)|) + p_2(|y|).$$

Let c' be the compilation sending $x \in \Sigma^*$ to the concatenation of $c(x)$ and $h''(k(x)) + p_1(|c(x)|)$ many placeholders. Hence, for a suitable function f' over nonnegative integers,

$$\begin{aligned} |c'(x)| &= |c(x)| + h''(k(x)) + p_1(|c(x)|) \\ &\leq f'(k(x))\text{poly}(|x|), \end{aligned}$$

that is, c' is an fpt-size compilation with respect to the parameterization k.

Since $c'(x)$ is obtained by suffixing placeholders to $c(x)$, by our technical assumptions it holds that $(c'(x), y) \in L$ if and only if $(c(x), y) \in L$. Moreover, algorithm A decides the instance $(c'(x), y)$ in time bounded above by $O(|(c'(x), y)|)$ plus the runtime of A on instance $(c(x), y)$, that is,

$$h''(k(x)) + p_1(|c(x)|) + p_2(|y|) \leq |c'(x)| + p_2(|y|),$$

which is $\text{poly}(|(c'(x), y)|)$ time. Hence L is polynomial-time decidable. □

3 Positive Results

In this section, we illustrate the potential of parameterized compilation using the *clause entailment* problem as a case study: Given a CNF formula ϕ and a clause δ, does ϕ entail δ? We view clause entailment as a compilation problem on instances (ϕ, δ), where the offline instance is ϕ and the online instance is δ.

Encoding Circuits and CNF formulas. Our forthcoming compilation results depend on certain basic properties satisfied by (any) reasonable encoding of boolean circuits; for the sake of clarity, we start by making these assumptions explicit.

Let Y be a finite subset of $X = \{x_i \colon i \in \mathbb{N}\}$. A circuit C on input variables Y is built in the usual way. Its underlying directed acyclic graph (DAG) has source nodes (indegree 0) called input gates, labelled by a constant ($\bot$ or $\top$) or a variable in Y, and a unique sink node called output gate. Nodes that are not sources are called internal gates and are labelled by $\neg$ (indegree 1), or by $\wedge$ or $\vee$ (unbounded indegree).

Let C be a circuit whose n input gates are labelled by the n variables $x_{i_1}, \ldots, x_{i_n}$ (we write $\mathsf{var}(C) = \{x_{i_1}, \ldots, x_{i_n}\}$); moreover, C has m internal gates and w wires (arcs in the underlying DAG).

We fix an alphabet Σ and encode C by the adjacency list representation of its underlying DAG, as follows. The input gates are encoded by the index of their variable and the internal gates are encoded by indices from 1 to m (in binary). We denote the size of the encoding of C by $|C|$. It is readily verified that $|C|$ is in $O(\log i_1 + \cdots + \log i_n + w \log m)$ and moreover that $|C| \geq \log i_1 + \cdots + \log i_n$.

A *CNF formula* is a set of sets of literals (clauses) on variables in $X = \{x_i \colon i \in \mathbb{N}\}$. The *size* of a CNF formula ϕ, denoted by $|\phi|$, is the size of its encoding when represented as a circuit. Namely, a CNF formula $\{C_1, \ldots, C_m\}$ on variables $x_{i_1}, \ldots, x_{i_n}$ corresponds to the circuit with n input gates labelled $x_{i_1}, \ldots, x_{i_n}$ and m internal $\vee$-gates $c_1, \ldots, c_m$ wiring the output $\wedge$-gate. The input gate x_{i_j} wires the internal gate c_i, directly or through a $\neg$-gate, if x_{i_j} occurs, positively or negatively, in C_i. For a CNF formula ϕ, and a (partial) truth assignment $f : \text{var}(\phi) \to \{0, 1\}$, we let $\phi[f]$ denote the CNF formula obtained from ϕ by (1) deleting all clauses containing a literal that is satisfied by f and (2) deleting literals falsified by f from all remaining clauses.

Technical Report AC-TR-16-003

Parameterizing Clause Entailment. Clause entailment is not only a basic example of a co-NP-complete problem (not polynomial-time solvable unless P = NP), but is also a fundamental example of a classically incompilable problem: it is not poly-size compilable unless the Polynomial Hierarchy (PH) collapses to the second level [24].

It is readily verified that clause entailment is fixed-parameter tractable for every parameterization of CNF formulas for which the satisfiability problem (SAT) is fixed-parameter tractable, and that does not increase when variables are instantiated. An important case is incidence treewidth [25]; the *incidence treewidth* of a CNF formula ϕ is the treewidth [2] of its incidence graph, that is the bipartite graph on variables and clauses of ϕ where a variable is adjacent to a clause if and only if it occurs in a literal in the clause.

We therefore introduce two parameterizations of CNF formulas, namely incidence treewidth modulo backbones and incidence treewidth modulo equivalence, both strictly dominating incidence treewidth, where SAT is *not fixed-parameter tractable*, and hence the quest for parameterized compilability is interesting: positive fpt-size compilation results indicate the additional computational power of fpt-size compilation over fixed-parameter tractability (and poly-size compilability).

The *incidence treewidth modulo equivalence* of a CNF formula ϕ is the minimum incidence treewidth attained over all CNF formulas that are logically equivalent to ϕ. The *incidence treewidth modulo backbones* of a CNF formula ϕ is the incidence treewidth of the CNF formula obtained from ϕ by first instantiating the literals over $\mathrm{var}(\phi)$ that are entailed by ϕ, and next conjoining them to the resulting CNF formula. As an example, consider the CNF formula $\phi_1 = \{\{x_1, x_2\}, \{\neg x_3, x_4\}, \{x_3, x_4\}\}$, which has incidence treewidth 2. The only entailed literal over $\mathrm{var}(\phi)$ is x_4, and the resulting CNF formula $\{\{x_1, x_2\}, \{x_4\}\}$ (after instantiation and conjunction of the entailed literals) has incidence treewidth 1. Therefore, the incidence treewidth modulo backbones of ϕ_1 is 1. Another example is an unsatisfiable CNF ϕ_2 over the variables $x_1, \ldots, x_n$. Since ϕ_2 entails all the literals over $x_1, \ldots, x_n$, the resulting CNF formula is $\{\{x_1\}, \{\neg x_1\}, \ldots, \{x_n\}, \{\neg x_n\}\}$, which has incidence treewidth 1. In other words, the incidence treewidth modulo backbones of every unsatisfiable CNF formula is 1.

Proposition 2. *Incidence treewidth modulo equivalence strictly dominates incidence treewidth modulo backbones, which strictly dominates incidence treewidth.*

Proof. For the first statement, observe that the CNF obtained from a CNF ϕ by instantiating and conjoining entailed literals is logically equivalent to ϕ; it follows that incidence treewidth modulo equivalence dominates incidence treewidth modulo backbones. For strictness, take the CNF family

$$\phi_n = \bigwedge_{i=1}^{n} (x_1 \vee \neg x_1 \vee \cdots \vee x_n \vee \neg x_n \vee y_i \vee \neg y_i),$$

where $n \geq 1$. Since each ϕ_n is logically equivalent to $\top$, the family has incidence treewidth modulo equivalence bounded above by 1. On the other hand, since ϕ_n does not entail any literal, its incidence treewidth modulo backbones coincides with its incidence treewidth, and the latter is bounded below by n, the incidence treewidth of the complete bipartite graph $K_{n,n}$.

For the second statement, observe that the incidence graph of the CNF obtained by instantiating and conjoining entailed literals in a CNF ϕ is the disjoint union of a subgraph of the incidence graph of ϕ (obtained by deleting edges and vertices) and a forest; hence its treewidth is bounded above by the incidence treewidth of ϕ. For strictness, take the CNF family ($n \geq 1$)

$$\phi_n = \bigwedge_{(b_1,\ldots,b_n) \in \{0,1\}^n} (y \vee \bigvee_{i=1}^{n} x_i^{b_i}),$$

where $x_i^{b_i} = \neg x_i$ if $b_i = 0$ and $x_i^{b_i} = x_i$ if $b_i = 1$. Note that y is the only literal entailed by ϕ_n, and that instantiating and conjoining y in ϕ_n yields the CNF whose only clause is y itself. The latter has incidence treewidth 1. On the other hand, the incidence treewidth of ϕ_n is bounded below by the treewidth of the complete bipartite graph $K_{n+1,2^n}$, which is larger than n. □

No parameterization of clause entailment is poly-size compilable (unless the PH collapses to the second level). Moreover, whereas clause entailment parameterized by incidence treewidth is fixed-parameter tractable [25], this does not hold for incidence treewidth modulo backbones (unless P = NP), as we now prove.

Technical Report AC-TR-16-003

Proposition 3. *Clause entailment parameterized by incidence treewidth modulo backbones is not fixed-parameter tractable (unless* P = NP*).*

The assumption P ≠ NP is already in the background (otherwise, clause entailment is poly-time).

Proof. We assume that clause entailment parameterized by incidence treewidth modulo backbones is fixed-parameter tractable and derive P = NP.

Let A be an algorithm deciding clause entailment in time at most $f(k)\text{poly}(|(\phi,\delta)|)$ where k is the incidence treewidth modulo backbones of ϕ; without loss of generality, f is increasing [12]. Let $d = f(1)$. Let A' be the algorithm that, when given a CNF formula $\phi = \delta_1 \wedge \cdots \wedge \delta_m$, runs $d \cdot \text{poly}(|(\phi', \{x\})|)$ steps of A on input $(\phi', \{x\})$, returns the output of A if A terminates, and rejects otherwise; here, x is a variable not in ϕ and $\phi' = (\delta_1 \vee x) \wedge \cdots \wedge (\delta_m \vee x)$. We claim that A' decides UNSAT in polynomial time.

Observe that $\phi' \equiv x$ if and only if $\phi \in$ UNSAT. If A terminates in $d \cdot \text{poly}(|(\phi', \{x\})|)$ steps, it correctly establishes whether or not $\phi' \models x$, that is, whether or not $\phi \in$ UNSAT. Otherwise, it follows that the incidence treewidth modulo backbones of ϕ' is at least 2, which implies $\phi \notin$ UNSAT. Namely, if $\phi \in$ UNSAT, then $\phi' \equiv x$ and thus the incidence treewidth modulo backbones of ϕ' is 1. In either case, the output of A' is correct. □

Since incidence treewidth modulo equivalence dominates incidence treewidth modulo backbones, we immediately get the following.

Corollary 1. *Clause entailment parameterized by incidence treewidth modulo equivalence is not fixed-parameter tractable (unless* P = NP*).*

Treewidth Modulo Equivalence. We now prove that the most general parameter introduced so far, incidence treewidth modulo equivalence, yields an fpt-size compilation of clause entailment.

Theorem 1. *Clause entailment parameterized by incidence treewidth modulo equivalence is fpt-linear-size compilable.*

Proof. Let ϕ be a CNF formula and let k be the incidence treewidth modulo equivalence of ϕ. Let s be the minimum size attained by a CNF formula equivalent to ϕ, and having incidence treewidth k. Let ψ be a CNF formula equivalent to ϕ having size s and incidence treewidth k, and let $x_{i_1}, \ldots, x_{i_n}$ be the variables occurring in ψ.

We first observe that every variable x_{i_j} occurring in ψ is essential, otherwise by deleting inessential variables we decrease the size of ψ (and perhaps its incidence treewidth as well), contradicting minimality.

It is known that there exists a DNNF[4] C equivalent to ψ having $w = O(3^k n)$ wires [21, Theorem 2]. We assume that input gates with the same label are identified. Since C is equivalent to ψ, and $x_{i_1}, \ldots, x_{i_n}$ are essential in ψ, we have n input gates in C labelled by $x_{i_1}, \ldots, x_{i_n}$. Moreover, as the number of gates is bounded above by the number of wires, we also have that C has $m = O(3^k n)$ internal gates. Thus, by our stipulations, the size of the encoding of C is $|C| = O(\log i_1 + \cdots + \log i_n + w \log m)$. We claim that this upper bound on $|C|$ is fpt-linear-size with respect to k. The statement then follows since C computes ϕ (which is equivalent to ψ), and clause entailment is polynomial-time tractable on DNNFs [7].

To prove the claim, we first observe that

$$|\phi| \geq \sum_{j=1}^{n} \log i_j \geq \sum_{j=1}^{n} \log j = \Omega(n \log n), \tag{1}$$

where the first inequality follows from the properties of our encoding of CNF formulas, and the fact that $x_{i_1}, \ldots, x_{i_n}$ must occur in ϕ (it being equivalent to ψ and $x_{i_1}, \ldots, x_{i_n}$ being essential in ψ). The last equality is well known.

By the upper bounds on w and m, we have that $w \log m$ is in $O(3^k kn \log n)$, and thus, by (1), in $O(3^k k|\phi|)$. We therefore get that $|C| = O(3^k k|\phi|)$. □

In the proof of Theorem 1, it is crucial that the compilation of CNF formulas into DNNF by Darwiche and Oztok [21] does not depend on the size of the CNF formula, but only on the number of variables; we do not know whether or not there exists a CNF class where incidence treewidth modulo equivalence is minimized by CNF formulas of superpolynomial size.

[4] A DNNF is a circuit in negation normal form where the input gates of the subcircuits leading into each ∧-gate are labelled over disjoint sets of variables [7].

Technical Report AC-TR-16-003

Fpt-Linear-Size Compilation. Interestingly, we can argue that the fpt-linear-size compilation of the parameter incidence treewidth modulo equivalence is optimal (Thereom 1), in the sense that no fpt-sublinear-size compilation of clause entailment is possible as long as the parameterization is "natural".

More specifically, say that two CNF formulas $\phi(z_1, \ldots, z_n)$ and $\psi(y_1, \ldots, y_n)$ are *isomorphic* if there is a bijection $s\colon \mathrm{var}(\phi) \to \mathrm{var}(\psi)$ such that $\phi(s(z_1), \ldots, s(z_n))$ and $\psi(y_1, \ldots, y_n)$ are equal (as sets of sets of literals). Call a parameterization k of clause entailment *invariant* if $k(\phi) = k(\phi')$ whenever ϕ and ϕ' are isomorphic.

Proposition 4. *Clause entailment parameterized by any invariant parameterization is not fpt-sublinear-size compilable, that is, there does not exist a compilation c, a computable function f and a sublinear non-decreasing function g, and a poly-time problem L such that, for all instances (ϕ, δ), $|c(\phi)| \leq f(k(\phi))g(|\phi|)$ and $\phi \models \delta$ if and only if $(c(\phi), \delta) \in L$.*

Proof. Assume for a contradiction that clause entailment is fpt-sublinear-size compilable parameterized by an invariant parameterization k.

Let $|\Sigma| = r \geq 2$. For all $i \in \mathbb{N}$, let ϕ_i be the CNF formula whose only clause is $\{x_i\}$. Hence, by our assumptions on the encoding of CNF formulas, the size of ϕ_i is $d \log_r i$ for some constant d (depending only on the encoding and not on i). Moreover, for all i and j in $\mathbb{N}$, it holds that $k(\phi_i) = k(\phi_j)$ since k is invariant. Let then $K = k(\phi_i)$.

By assumption, there exist c, f, and g specified as in the statement such that, for all $i \in \mathbb{N}$, $|c(\phi_i)| \leq f(k(\phi_i))g(|\phi_i|) = f(K)g(d \log_r i)$. By the assumption on g, for sufficiently large S, we have $\log_r S - 1 > f(K)g(d \log_r S)$. Thus $|c(\phi_i)| \leq f(K)g(d \log_r i) \leq f(K)g(d \log_r S) < \log_r S - 1$ for all $i \leq S$. Since there are less than $r^{\log_r S} = S$ strings of length at most $\log_r S - 1$ on Σ, there must be $i, j \leq S$ such that $i \neq j$ and $c(\phi_i) = c(\phi_j)$. Then $\phi_i \models x_i$ and $\phi_j \not\models x_i$, and thus $(c(\phi_i), x_i) \in L$ and $(c(\phi_j), x_i) \notin L$, which is a contradiction. □

Treewidth Modulo Backbones. We now turn our attention to the parameter incidence treewidth modulo backbones. Since incidence treewidth modulo equivalence dominates incidence treewidth modulo backbones, Theorem 1 implies an fpt-linear-size compilation for the latter parameter as well.

Corollary 2. *Clause entailment parameterized by incidence treewidth modulo backbones is fpt-linear-size compilable.*

In fact, this parameter has a straightforward fpt-linear-size compilation: namely, the map sending a CNF formula ϕ to the CNF formula $\phi' \wedge \bigwedge_{1 \leq i \leq m} l_i$, where $l_1, \ldots, l_m$ are all the literals entailed by ϕ and ϕ' is the result of instantiating $l_1, \ldots, l_m$ in ϕ.

In addition to being theoretically easier to handle, the parameter incidence treewidth module backbones is also practically more manageable than incidence treewidth modulo equivalence. In fact, we can use existing tools to compute the incidence treewidth module backbones of various CNF instances, and therefore give a concrete indication of the additional power of this parameter over incidence treewidth.

In Table 1, we provide some preliminary experimental results indicating that there are instances occurring in practical settings where this additional power of the parameter is visible, i.e., where the incidence treewidth is significantly lower after instantiating entailed literals. This suggests that the setting of parameterized compilation does not only offer possibilities for more powerful parameters in theory, but also in practice.

To compute the parameter values in Table 1, we used an existing implementation of heuristics to compute upper bounds on the incidence treewidth [1] and algorithmic tools developed for computing entailed literals [15]. The instances have been used previously in the context of computing entailed unit clauses [18].

Note that there are sizable instances (e.g., AProVE09-08) where incidence treewidth modulo backbones is small enough, for instance, to solve clause entailment by dynamic programming (as the algorithm needs exponential space in the incidence treewidth, [23]), but incidence treewidth itself is not. Such instances are interesting, as incidence treewidth could in principle stay large even after the instantiation of many entailed literals (which is in fact what happens for other instances).

As the values that we computed for the incidence treewidth before and after instantiating backbones are upper bounds based on heuristics, one cannot be sure that the difference between the exact parameter values is as large. However, since there are no efficient methods of computing the exact incidence treewidth, upper bound values such as these are in many cases the best values available to work with in practice.

Technical Report AC-TR-16-003

Therefore the results in Table 1 are relevant when considering the development of algorithms in practical settings.

File	*#vars*	*#clauses*	itw	itwbb
3blocks	283	9690	35	22
4blocksb	410	24758	58	7
AProVE09-07	8567	28936	82	37
AProVE09-08	8564	28927	85	12
AProVE09-13	7606	26317	44	15
bw_large.b	920	11491	236	220
f8h_11	2883	37388	330	120
facts7h	2595	32952	286	163
ferry8_ks99i.renamed-4005	2547	32525	380	65
ferry8_v01i.renamed-4007	1745	31688	324	64
IBM_FV_2004_rule_batch_1	7410	29233	215	174
IBM_FV_2004_rule_batch_2	7416	29254	208	168
IBM_FV_2004_rule_batch_3	7421	29264	198	167
maris-s03-gripper11	3222	27199	305	263
medium	116	953	52	7
rovers5_ks99i.renamed-3974	1437	29170	397	235
rovers5_v01a.renamed-3981	981	25949	386	267
satellite2_v01i.shuffled-4055	853	27249	191	31

Table 1: Comparison of the incidence treewidth (itw) and incidence treewidth modulo backbones (itwbb) on instances of file size not larger than 2MB.

Technical Report AC-TR-16-003

4 Negative Results

We contrast the positive results in the previous section by presenting negative, parameterized incompilability results for (certain variants of) the clause entailment problem.

We use the parameterized complexity framework of Chen [4]. The framework generalizes the classical compilability framework of Cadoli et al. [3], where one can prove that a compilation problem does not admit a poly-size compilation unless all problems in NP are solvable in poly-time using poly-size advice (i.e., NP $\subseteq$ P/poly), which further implies a collapse of PH to the second level [16]. Analogously, using parameterized complexity, one can prove that a parameterized compilation problem does not admit an fpt-size compilation unless all problems in the class W[1] are solvable in fpt-time using fpt-size advice (i.e., W[1] $\subseteq$ FPT/fpt) or unless all problems in the class para-NP are solvable in fpt-time using fpt-size advice (i.e., para-NP $\subseteq$ FPT/fpt), the latter of which also implies a collapse of the PH to the second level.

We begin with explaining the basic notions of the parameterized incompilability framework [4], and how it can be used to show (under complexity-theoretic assumptions) that certain problems are not fpt-size compilable. In this section, in order to more conveniently apply the framework, we consider parameterized problems to be subsets of $\Sigma^* \times \mathbb{N}$. For an instance (x, k) of a parameterized problem, we say that k is the parameter value. One can consider this as a shorthand for the parameterization $k : \Sigma^* \times \mathbb{N} \to \mathbb{N}$ that maps any instance (x, k) to the value k. Similarly, we consider parameterized compilation problems as subsets of $\Sigma^* \times \Sigma^* \times \mathbb{N}$, where for each instance (x, y, k), the value k denotes the value computed (from x) by the parameterization.

Formally, a parameterized problem Q is in FPT/fpt, if there exist a computable function f and a constant c such that for each $(n, k) \in \mathbb{N} \times \mathbb{N}$ there exists some $\alpha(n, k) \in \Sigma^*$ of size $f(k)n^c$ with the property that the problem $\{ (x, \alpha(|x|, k), k) : (x, k) \in Q \}$ is fixed-parameter tractable. This definition straightforwardly extends to K/fpt for other parameterized complexity classes K. The classes K/fpt have been defined by Chen under the name K/ppoly [4].[5] Similarly, if we require the size of the advice string $\alpha(n, k)$ to be bounded by $n^{f(k)}$ (i.e., to be of xp-size), we obtain parameterized complexity classes K/xp.

[5]We use K/fpt to denote this class, rather than K/ppoly, to avoid confusion with the class K/poly, which has also been considered [13].

A central element of the parameterized incompilability framework is the concept of hardness for the incompilability class par-nucomp-W[1]. Intuitively, this class contains parameterized compilation problems that can be compiled in fpt-size such that the online problem is in the parameterized complexity class W[1]. The class W[1] consists of all parameterized problems that can be fpt-reduced to the problem Clique, where the problem is to decide whether a given graph $G = (V, E)$ contains a clique of a given size k, and where the parameter is k. The class par-nucomp-W[1] is defined as follows. A parameterized compilation problem Q belongs to par-nucomp-W[1] if there exists an fpt-size function $f : \Sigma^* \times 1^* \times \mathbb{N} \to \mathbb{N}$ such that the parameterized problem $\{ (f(x, 1^m, k), y, k) : (x, y, k) \in Q, m \geq |y| \}$ is in the class W[1].

In order to define hardness for this class, one needs a notion of reduction. This role is fulfilled by the concept of fpt-nucomp-reductions. A parameterized compilation problem Q is fpt-nucomp-reducible to a parameterized compilation problem Q' if there exist fpt-size functions $f_1, f_2 : \Sigma^* \times 1^* \times \mathbb{N} \to \Sigma^*$, an fpt-time computable function $g : \Sigma^* \times \Sigma^* \times \mathbb{N} \to \Sigma^*$ and a computable function $h : \mathbb{N} \to \mathbb{N}$ such that for all inputs (x, y, k) of Q and all $m \geq |y|$ it holds that $(x, y, k) \in Q$ if and only if $(f_1(x, 1^m, k), g(f_2(x, 1^m, k), y, k), h(k)) \in Q'$.

Intuitively, the function f_1 transforms the offline instance x of Q into an fpt-size offline instance of Q', and the function f_2 transforms the offline instance x of Q into an auxiliary offline instance x'. The function g transforms the online instance y of Q (together with the auxiliary offline instance x') into an online instance of Q'. Finally, the function h ensures that the parameter value for Q' is not unbounded.

A parameterized compilation problem Q is defined to be hard for par-nucomp-W[1] if each problem in par-nucomp-W[1] is fpt-nucomp-reducible to Q. If, additionally, the problem is in par-nucomp-W[1], then it is said to be par-nucomp-W[1]-complete.

An example of a par-nucomp-W[1]-complete problem is the problem $\epsilon\text{MCC} = \{ (\epsilon, G, k) : (G, k) \in \text{MCC} \}$, where ϵ denotes the empty string, and MCC (or *multicolored clique*) is the W[1]-complete problem of deciding whether a given graph G, whose vertex set is partitioned into k subsets $V_1, \ldots, V_k$, contains a clique containing one vertex from each V_i [11, 4, Theorem 17].

Fundamental to the use of this incompilability class to give evidence for the incompilability of certain problems, is the result that if a par-nucomp-W[1]-hard parameterized compilation problem is fpt-size compilable, then W[1] $\subseteq$ FPT/fpt [4, Theorem 18]; for a proof, see [13, Proposition 57]. For any parameterized complexity class K, the incompilability class par-nucomp-K is defined analogously to par-nucomp-W[1].

Moreover, for many classes par-nucomp-K, it can be shown that par-nucomp-K-hard problems are not fpt-size compilable, under complexity-theoretic assumptions; for more details, see [4].

Backdoor Size to Pure Literals. The first parameter that we consider is based on a measure of distance to a fragment of CNF formulas for which deciding satisfiability is tractable. The fragment PL that we consider consists of all CNF formulas whose satisfiability can be decided by iterative instantiation of *pure literals*, that is, literals that appear only positively or only negatively in the formula. The notion of distance that we use is based on (strong) backdoors [26]. A *strong backdoor to PL* for a CNF formula ϕ is a subset Y of $\text{var}(\phi)$ such that $\phi[f]$ is in the fragment PL for every assignment $f\colon Y \to \{0, 1\}$. This parameter is motivated by the observation that propositional satisfiability, parameterized by the size of a given strong backdoor to PL, is fixed-parameter tractable.

Observe that instantiating variables in a CNF formula ϕ can increase the size of the smallest strong backdoor to PL by an arbitrary amount. Suppose, for instance, that all clauses of ϕ contain some literal x positively. Then the smallest strong backdoor to PL for ϕ is $\emptyset$, but for $\phi[x \mapsto 0]$ it can be arbitrarily large. For this reason, fixed-parameter tractability of SAT for this parameter does not lead to fixed-parameter tractability for clause entailment. In fact, we get the following parameterized incompilability result for clause entailment.

Proposition 5. *Clause entailment parameterized by the size of the smallest strong backdoor to PL is not fpt-size compilable, unless the PH collapses to the second level.*

Proof. We describe how to transform an arbitrary instance of CE (in fpt-time) to an instance that has a constant-size strong backdoor to PL. Fpt-size compilability of the problem would thus result in poly-size compilability of clause entailment, which in turn results in the collapse of the PH at the second level [24]. Formally, this transformation can be seen as an fpt-nucomp-reduction from the par-nucomp-coNP-complete parameterized compilation problem $\{ (x, y, 0) : (x, y) \in \text{CE} \}$.

Let ϕ be an arbitrary CNF formula, and let δ be an arbitrary clause. Take a fresh variable $z \notin \text{var}(\phi)$. Without loss of generality, we can assume that z does not occur in δ. We construct the CNF formula $\phi' =$

Technical Report AC-TR-16-003

$\{ c \cup \{z\} : c \in \phi \}$, and the clause $\delta' = \delta \cup \{z\}$. We have that $\phi \models \delta$ if and only if $\phi' \models \delta'$. Moreover, because ϕ' is in PL, it has a strong backdoor to PL of size 0. Since the construction of ϕ' does not depend on the clause δ, this transformation can be used to obtain a poly-size compilation for CE from an fpt-size compilation of clause entailment parameterized by the size of the smallest strong backdoor to PL. □

Weighted Clause Entailment. Next, we consider the following (parameterized) variant of the clause entailment problem, *weighted clause entailment*. Offline instances consist of a pair (ϕ, w) and online instances consist of a clause δ, where (ϕ, δ) is an instance of clause entailment and $w \in \mathbb{N}$. The parameter is w, and the question is whether $\phi \models \delta$ holds relative to all truth assignments f to $\mathrm{var}(\phi)$ of *weight* w (written $\phi \models_w \delta$). The weight of an assignment f is $|\{ x \in \mathrm{var}(\phi) : f(x) = 1 \}|$.

In the foundations of parameterized complexity, truth assignments of restricted weight play an important role, providing the basis for the weighted satisfiability problems that are canonical for the Weft hierarchy (see, e.g., [12]). Therefore, the variant of weighted clause entailment is natural to study in the setting of parameterized compilability.

In this variant of the problem, the space of all truth assignments is significantly restricted (only assignments of weight w play a role). As a result, for constant values of the parameter w, weighted clause entailment is polynomial-time decidable (and thus also poly-size compilable). However, we show that under the assumption that W[1] $\not\subseteq$ FPT/fpt, an fpt-size compilation is not possible; in any poly-size compilation of clause entailment restricted to weight w, the order of the polynomial depends on w.

Proposition 6. *Weighted clause entailment parameterized by the weight of assignments is not fpt-size compilable, unless* W[1] $\subseteq$ FPT/fpt.

Proof. We show that the problem is par-nucomp-coW[1]-hard. This suffices, since any parameterized compilation problem is fpt-size compilable if and only if its co-problem (the problem consisting of all the no-instances) is fpt-size compilable. We do so by giving an fpt-nucomp-reduction from the problem ϵcoMCC $= \{ (\epsilon, G, k) : (G, k) \notin \mathrm{MCC} \}$. Let (ϵ, G, k) be an instance of ϵcoMCC, where G is a graph whose vertex set is partitioned into $V_1, \ldots, V_k$, and let $m \geq |G|$. Moreover, without loss of generality we may assume that all sets V_i have the same cardinality; for each $1 \leq i \leq k$, let $V_i = \{v_{i,1}, \ldots, v_{i,n}\}$. To describe the fpt-nucomp-reduction, we specify suitable functions f_1, f_2, g, h. We let $f_1(\epsilon, 1^m, k)$ be the CNF formula ϕ that we will define below. Moreover, we let $f_2(\epsilon, 1^m, k) = 1^m$ and we let $g(1^m, G, k)$ be the clause δ that we will define below. Finally, we let $h(k) = k' = k + \binom{k}{2}$.

We let $\mathrm{var}(\phi) = \bigcup_{1 \leq i \leq k} X_i \cup \bigcup_{1 \leq i < j \leq k} Y_{i,j}$. Here, for each i we let $X_i = \{ x_{i,\ell} : 1 \leq \ell \leq m \}$. Also, for each $1 \leq i < j \leq k$, we let $Y_{i,j} = \{ y_{i,\ell_1,j,\ell_2} : 1 \leq \ell_1 \leq k, 1 \leq \ell_2 \leq k \}$. Intuitively, the variables $x_{i,\ell}$ encode the choice of vertices in a clique, and the variables y_{i,ℓ_1,j,ℓ_2} encode the choice of edges. Then, for each set $Z \in \{ X_i, Y_{i,j} : 1 \leq i \leq k, i < j \leq k \}$ of variables, and for each two variables $z_1, z_2 \in Z$, we add the clause $(\neg z_1 \vee \neg z_2)$ to ϕ. This enforces that each satisfying truth assignment of ϕ of weight k' must satisfy exactly one variable in each set Z. Moreover, for each $1 \leq i < j \leq k$, each $1 \leq \ell_1 \leq m$ and each $1 \leq \ell_2 \leq m$, we add the clauses $(\neg y_{i,\ell_1,j,\ell_2} \vee x_{i,\ell_1})$ and $(\neg y_{i,\ell_1,j,\ell_2} \vee x_{j,\ell_2})$ to ϕ. Intuitively, these clauses enforce that the choice of edges is compatible with the choice of vertices.

We define the clause $\delta = g(1^m, G, k)$ as follows. Let n be the number of vertices in G. For each $1 \leq i < j \leq k$, each $1 \leq \ell_1 \leq m$, and each $1 \leq \ell_2 \leq m$, we add the literal y_{i,ℓ_1,j,ℓ_2} to δ if one of the following cases holds: (i) either $\ell_1 > n$ or $\ell_2 > n$, or (ii) there is no edge in G between v_{i,ℓ_1} and v_{j,ℓ_2}.

We claim that satisfying truth assignments of $\psi \wedge \neg\delta$ of weight k' are in one-to-one correspondence with cliques in G of size k containing exactly one vertex in each V_i. For each such clique $V' = \{v_{1,\ell_1}, \ldots, v_{k,\ell_k}\}$ in G (with $v_{i,\ell_i} \in V_i$), one can obtain the satisfying assignment that sets exactly those variables in the set $X' = \{ x_{i,\ell_i} : 1 \leq i \leq k \} \cup \{ y_{i,\ell_i,j,\ell_j} : 1 \leq i < j \leq k \}$ to true. Vice versa, from each satisfying assignment, one can construct a suitable clique in G. With this correspondence, one can verify straightforwardly that $\phi \models_{k'} \delta$ if and only if $(G, k) \notin$ MCC. This shows the correctness of our reduction.

For the sake of clarity, we show how this par-nucomp-coW[1]-hardness entails incompilability (under the assumption that W[1] $\not\subseteq$ FPT/fpt). Suppose that weighted clause entailment is fpt-size compilable. We show that W[1] $\subseteq$ FPT/fpt, by showing that MCC can be solved in fpt-time using fpt-size advice. For an instance (G, k) of MCC we firstly construct ϕ, δ and k' according to the construction discussed above. We showed that $\phi \not\models_{k'} \delta$ if and only if $(G, k) \in$ MCC. Since the offline instance (ϕ, k) of weighted clause entailment depends only on the pair (n, k), where $n = |G|$, we can use the fpt-size compilation $c(\phi, k)$ as the advice string to solve the problem MCC in fpt-time. □

Technical Report AC-TR-16-003

Small Clauses. Finally we consider parameterizing clause entailment by the size of clauses for which we want to decide entailment. Formally, we define a (parameterized) variant of clause entailment, *small clause entailment*, where offline instances are pairs (ϕ, s), where ϕ is a CNF formula and $s \in \mathbb{N}$, and online instances are clauses δ. The question is whether both $|\delta| \leq s$ and $\phi \models \delta$ and the parameter is s, which intuitively bounds the size of clauses δ.

For constant values of the parameter s, the problem can straightforwardly be compiled in poly-size, by creating a lookup table with an entry for each of the polynomially many clauses of size at most s. However, in this compilation, the order of the polynomial depends on s. We claim that (under the following complexity-theoretic assumption) small clause entailment parameterized by the clause size is not fpt-size compilable, and thus that for any poly-size compilation of clause entailment restricted to clauses of size at most s, the order of the polynomial depends on s.

Proposition 7. *Small clause entailment parameterized by the clause size is not fpt-size compilable, unless* few-NP $\subseteq$ FPT/fpt.

Before giving a proof, we informally explain the complexity assumption underlying this parameterized incompilability result. First we introduce the parameterized complexity class few-NP [13], which is a parameterized complexity counterpart of NP, like W[1]. For 3SAT, the archetypal NP-complete problem, there are 2^n many possible instances of input size n, and for each of these instances there are 2^m truth assignments that play a role, where m is the number of variables. The class W[1] is based on a parameterized restriction of 3SAT, where there still are 2^n many possible instances of input size n, but only m^k many truth assignments play a role, where k is an explicitly given parameter value. Intuitively, the class few-NP is based on a dual restriction of 3SAT, where for each instance with m variables, there are 2^m truth assignments that play a role, but there are only n^k many instances for each input size n and parameter value k.

To formally define the class few-NP we consider the notion of 3CNF generators. A *(3CNF) generator* is a function $\gamma : \mathbb{N}^3 \to \Sigma^*$ such that for all $(n, \ell, k) \in \mathbb{N}^3$ it holds that $\gamma(n, \ell, k)$ is a 3CNF formula (i.e., a CNF formula with clauses of size 3) with exactly n variables, and if $\ell > n^k$, then $\gamma(n, \ell, k) = \emptyset$ is the empty CNF formula. We say that a generator γ is *fpt-time computable* if there exists an algorithm A that, given an input $(n, \ell, k) \in \mathbb{N}^3$, computes $\gamma(n, \ell, k)$ in fpt-time (with respect to n and k). Then, for each generator, we define the parameterized problem FewSAT$_\gamma$ as follows. Instances consist of triples $(n, \ell, k) \in \mathbb{N}^3$, where n is given in unary and ℓ and k are given in binary. The parameter is k, and the question is to decide whether $\gamma(n, \ell, k)$ is satisfiable. The class few-NP then consists of all parameterized problems that are fpt-reducible to the problem FewSAT$_\gamma$, for some fpt-time computable generator γ.

To put the class few-NP in perspective, we describe how it relates to several other parameterized complexity classes. Like the class W[1] (and other classes of the Weft hierarchy), it contains FPT and is contained in para-NP. Moreover, similarly to W[1], containment in para-NP is strict, assuming that P $\neq$ NP. Both W[1] and few-NP are in the fragment of para-NP that is contained in XP/xp, which is the class consisting of parameterized problems that can be solved in time $n^{O(k)}$ using $n^{O(k)}$-size advice for each input size n and parameter value k.

With the definition of the complexity-theoretic assumption in place, we now set out to prove the incompilability result for small clause entailment. In order to do so, we will use two technical lemmas. Here we consider the problems FewSAT$_\gamma$ and coFewSAT$_\gamma = \{ (n, \ell, k) : (n, \ell, k) \notin$ FewSAT$_\gamma \}$ (for generators γ) as parameterized compilation problems.

Lemma 1. *Let γ be an fpt-time computable generator. Then* coFewSAT$_\gamma$ *is fpt-nucomp-reducible to small clause entailment, parameterized by the clause size.*

Proof (sketch). We describe an fpt-nucomp-reduction by specifying fpt-size functions f_1, f_2, an fpt-time computable function g, and a computable function h as follows. We let $f_2(n, 1^m, k) = n$. In addition, we let $g(n, \ell, k)$ be a clause δ_ℓ that encodes ℓ using k literals over the variables $x_{i,j}$, for $1 \leq i \leq n, 1 \leq j \leq k$. Then, we let $f_1(n, 1^m, k)$ be a 3CNF formula φ that is satisfiable in conjunction with $\neg\delta_\ell$ if and only if $\gamma(n, \ell, k)$ is satisfiable, for each $1 \leq \ell \leq n^k$. This can be done as follows.

Firstly, the formula φ contains clauses to ensure that at most k variables $x_{i,j}$ are true. Then, we add $8n^3$ many variables y_i, corresponding to the $8n^3$ possible clauses $c_1, \ldots, c_{8n^3}$ of size 3 over the variables $x_1, \ldots, x_n$. Then, since for each $(n, k) \in \mathbb{N}^2$, the function $\gamma(n, \cdot, k)$ is computable in ftp-time, we can construct (in fpt-time) a set of clauses that ensure that whenever $\neg\delta_\ell$ is satisfied the variables y_i

Technical Report AC-TR-16-003

must be set to true for the clauses $c_i \in \gamma(n, \ell, k)$. We add these clauses to φ as well. Finally, for each such possible clause c_i, we add clauses to φ to ensure that whenever y_i is set to true, then the clause c_i must be satisfied. □

Lemma 2. *Let γ be a generator. If* FewSAT$_\gamma$ *is fpt-size compilable, then* FewSAT$_\gamma \in$ FPT/fpt.

Proof. Assume that there exist an fpt-size function f_1, an fpt-time function g, a computable function h, and a parameterized problem $Q \in$ FPT such that for each instance (n, ℓ, k) of FewSAT$_\gamma$ and each $m \geq |\ell|$ it holds that $(n, \ell, k) \in$ FewSAT$_\gamma$ if and only if $(f_1(n, 1^m, k), g(\ell, k), h(k)) \in Q$. We show that FewSAT$_\gamma \in$ FPT/fpt. Take an arbitrary $(n, k) \in \mathbb{N}^2$. Consider the string $\alpha = f_1(n, 1^m, k)$ as advice, for some $m \geq |\ell| = k \log n$. Then deciding for some $1 \leq \ell \leq n^k$ whether $(n, \ell, k) \in$ FewSAT$_\gamma$ can be done in fpt-time using the advice string α, by checking whether $(\alpha, g(\ell, k), h(k)) \in Q$. □

With these technical lemmas in place, we can now prove the incompilability result for small clause entailment parameterized by the clause size.

Proof of Proposition 7. Suppose that small clause entailment parameterized by the clause size is fpt-size compilable. We show that few-NP $\subseteq$ FPT/fpt. Let γ be an arbitrary fpt-time computable generator. We show that FewSAT$_\gamma \in$ FPT/fpt. By Lemma 1, we know that coFewSAT$_\gamma$ is fpt-nucomp-reducible to small clause entailment parameterized by the clause size. By assumption, we know that the latter problem is fpt-size compilable. Composing this fpt-nucomp-reduction and the fpt-size compilation gives us an fpt-size compilation for coFewSAT$_\gamma$. Then, we also know that FewSAT$_\gamma$ is fpt-size compilable. Finally, by Lemma 2, we know that FewSAT$_\gamma \in$ FPT/fpt. □

As a final remark, we point out that the incompilability result for small clause entailment in Proposition 7 can even be shown under the weaker complexity-theoretic assumption that nu-few-NP $\not\subseteq$ FPT/fpt, where nu-few-NP is defined similarly to few-NP, based on 3CNF generators that are computable in non-uniform fpt-time. For more details, see the work of [13].

Technical Report AC-TR-16-003

5 Conclusion

We proved parameterized incompilability and compilability results for various computational problems related to clause entailment. Interestingly, our contributions on compilability and entailment fit exactly two of the three topics recently selected by Marquis as representative and central in the area of knowledge compilation [19].

On the negative side, we used the incompilability framework introduced by Chen [4] to establish fixed-parameter incompilability for variants of clause entailment, including clause entailment restricted to clauses of bounded size.

On the positive side, we proved that clause entailment is not fixed-parameter tractable but is fixed-parameter compilable when parameterized by incidence treewidth modulo equivalence and incidence treewidth modulo backbones; we provided evidence that the investigated parameters are more powerful than pure incidence treewidth not only in theory but also in practice. These compilability results witness that the notion of fixed-parameter compilability properly extends both the notions of polynomial-size compilability and fixed-parameter tractability; we hope this will stimulate future work on parameterized compilation.

Acknowledgments. This research was supported by the FWF Austrian Science Fund (Parameterized Compilation, P26200).

References

[1] Bernhard Bliem, Michael Morak, and Stefan Woltran. D-FLAT: declarative problem solving using tree decompositions and answer-set programming. *Theory Pract. Log. Program.*, 12(4-5):445–464, 2012.

[2] Hans L. Bodlaender. Treewidth: Structure and algorithms. In Giuseppe Prencipe and Shmuel Zaks, editors, *Proceedings of SIROCCO, the 14th International Colloquium on Structural Information and Communication Complexity*, volume 4474 of *Lecture Notes in Computer Science*, pages 11–25. Springer Verlag, 2007.

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

[4] Hubie Chen. Parameterized compilability. In *Proceedings of IJCAI 2005, the 19th International Joint Conference on Artificial Intelligence*, 2005.

[5] Hubie Chen. Parameter compilation. In *Proceedings of IPEC, the 10th International Symposium on Parameterized and Exact Computation*, 2015.

[6] Arthur Choi and Adnan Darwiche. Dynamic minimization of sentential decision diagrams. In Marie desJardins and Michael L. Littman, editors, *Proceedings of AAAI, the 27th AAAI Conference on Artificial Intelligence*. AAAI Press, 2013.

[7] A. Darwiche. Decomposable negation normal form. *J. of the ACM*, 48(4):608–647, 2001.

[8] Adnan Darwiche. New advances in compiling CNF into decomposable negation normal form. In *Proceedings of ECAI, the 16th Eureopean Conference on Artificial Intelligence*, pages 328–332, 2004.

[9] Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity*. Monographs in Computer Science. Springer Verlag, New York, 1999.

[10] Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer Verlag, 2013.

[11] Michael R. Fellows, Danny Hermelin, Frances A. Rosamond, and Stéphane Vialette. On the parameterized complexity of multiple-interval graph problems. *Theoretical Computer Science*, 410(1):53–61, 2009.

[12] Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*, volume XIV of *Texts in Theoretical Computer Science. An EATCS Series*. Springer Verlag, Berlin, 2006.

[13] Ronald de Haan. An overview of non-uniform parameterized complexity. Technical Report TR15–130, *Electronic Colloquium on Computational Complexity* (ECCC), 2015.

[14] Jinbo Huang and Adnan Darwiche. Using DPLL for efficient OBDD construction. In *Proceedings of SAT, the 7th International Conference on Theory and Applications of Satisfiability Testing*, 2004.

[15] Mikolás Janota, Inês Lynce, and Joao Marques-Silva. Algorithms for computing backbones of propositional formulae. *AI Commun.*, 28(2):161–177, 2015.

[16] Richard M. Karp and Richard J. Lipton. Some connections between nonuniform and uniform complexity classes. In *Proceedings of STOC, the 12th Annual ACM Symposium on Theory of Computing*, pages 302–309, New York, NY, USA, 1980. Assoc. Comput. Mach., New York.

[17] Frédéric Koriche, Jean-Marie Lagniez, Pierre Marquis, and Samuel Thomas. Knowledge compilation for model counting: Affine decision trees. In Francesca Rossi, editor, *Proceedings of IJCAI, the 23rd International Joint Conference on Artificial Intelligence*. AAAI Press, 2013.

[18] Joao Marques-Silva, Mikoláš Janota, and Inês Lynce. On computing backbones of propositional theories. In *Proceedings of ECAI 2010, the 19th European Conference on Artificial Intelligence*. IOS Press, 2010.

[19] Pierre Marquis. Compile! In Blai Bonet and Sven Koenig, editors, *Proceedings of AAAI, the 29th AAAI Conference on Artificial Intelligence*, pages 4112–4118. AAAI Press, 2015.

[20] Rolf Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford Lecture Series in Mathematics and its Applications. Oxford University Press, Oxford, 2006.

[21] Umut Oztok and Adnan Darwiche. CV-width: A new complexity parameter for CNFs. In Torsten Schaub, Gerhard Friedrich, and Barry O'Sullivan, editors, *Proceedings of ECAI, the 21st European Conference on Artificial Intelligence*, volume 263 of *Frontiers in Artificial Intelligence and Applications*, pages 675–680. IOS Press, 2014.

Technical Report AC-TR-16-003

[22] Umut Oztok and Adnan Darwiche. A top-down compiler for sentential decision diagrams. In *Proceedings of IJCAI, the 24th International Joint Conference on Artificial Intelligence*, pages 3141–3148, 2015.

[23] Marko Samer and Stefan Szeider. Algorithms for propositional model counting. *J. Discrete Algorithms*, 8(1):50–64, 2010.

[24] B. Selman and H. A. Kautz. Knowledge compilation and theory approximation. *J. of the ACM*, 43:193–224, 1996.

[25] Stefan Szeider. On fixed-parameter tractable parameterizations of SAT. In Enrico Giunchiglia and Armando Tacchella, editors, *Theory and Applications of Satisfiability, 6th International Conference, SAT 2003, Selected and Revised Papers*, volume 2919 of *Lecture Notes in Computer Science*, pages 188–202. Springer Verlag, 2004.

[26] Ryan Williams, Carla Gomes, and Bart Selman. Backdoors to typical case complexity. In Georg Gottlob and Toby Walsh, editors, *Proceedings of IJCAI, the 18th International Joint Conference on Artificial Intelligence*, pages 1173–1178. Morgan Kaufmann, 2003.

Technical Report AC-TR-16-003