# A Parameterized Complexity View on Description Logic Reasoning

**Ronald de Haan**
Institute for Logic, Language and Computation
University of Amsterdam
me@ronalddehaan.eu

## Abstract

Description logics are knowledge representation languages that have been designed to strike a balance between expressivity and computational tractability. Many different description logics have been developed, and numerous computational problems for these logics have been studied for their computational complexity. However, essentially all complexity analyses of reasoning problems for description logics use the one-dimensional framework of classical complexity theory. The multi-dimensional framework of parameterized complexity theory is able to provide a much more detailed image of the complexity of reasoning problems.

In this paper we argue that the framework of parameterized complexity has a lot to offer for the complexity analysis of description logic reasoning problems—when one takes a progressive and forward-looking view on parameterized complexity tools. We substantiate our argument by means of three case studies. The first case study is about the problem of concept satisfiability for the logic $\mathcal{ALC}$ with respect to nearly acyclic TBoxes. The second case study concerns concept satisfiability for $\mathcal{ALC}$ concepts parameterized by the number of occurrences of union operators and the number of occurrences of full existential quantification. The third case study offers a critical look at data complexity results from a parameterized complexity point of view. These three case studies are representative for the wide range of uses for parameterized complexity methods for description logic problems.

## Introduction

Description logics have been designed as knowledge representation formalisms that have good computational properties (Baader et al. 2003). Correspondingly, there has been a lot of research into the computational complexity of reasoning problems for different description logics. This research has, however, focused entirely on the framework of classical complexity theory to study the computational complexity (see, e.g., Baader et al. 2003; Baader, Horrocks, and Sattler 2008).

The more fine-grained and multi-dimensional framework of parameterized complexity theory has hardly been applied to study the complexity of reasoning problems for description logics. Only a few works used the framework of parameterized complexity to study description logic problems (Bienvenu et al. 2017a; 2017b; Ceylan and Peñaloza 2014; Kikot, Kontchakov, and Zakharyaschev 2011; Motik 2012; Simančík, Motik, and Horrocks 2014; Simančík, Motik, and Krötzsch 2011). Moreover, these works all use the framework in a traditional way, focusing purely on one commonly used notion of tractability (namely that of fixed-parameter tractability).

Parameterized complexity is designed to address the downside of classical complexity theory that it is largely ignorant of structural properties of problem inputs that can potentially be exploited algorithmically. It does so by distuinguishing a problem parameter $k$, in addition to the input size $n$, and measuring running times in terms of both of these. The parameter $k$ can be used to measure various types of structure that are present in the problem input. Parameterized complexity theory has grown into a large and thriving research community over the last few decades (see, e.g., Bodlaender et al. 2012; Downey 2012; Downey and Fellows 2013). Most results and techniques in parameterized complexity theory revolve around the notion of *fixed-parameter tractability*—a relaxation of polynomial-time solvability based on running times of the form $f(k) \cdot n^{O(1)}$, for some computable function $f$ (possibly exponential or worse).

Due to the fact that reasoning problems related to description logics are typically of high complexity (e.g., complete for classes like PSPACE and EXPTIME), it is unsurprising that one would need very restrictive parameters to obtain fixed-parameter tractability results for such problems. It has been proposed recently that the investigation of problems that are of higher complexity can also benefit from the parameterized complexity point of view (De Haan 2016; De Haan and Szeider 2014a; 2014b; 2016; 2017)—using tools and methods that overstep the traditional focus on fixed-parameter tractability as positive results.

In this paper, we show how the complexity study of description logic problems can benefit from using the framework of parameterized complexity and all the tools and methods that it offers. We do so using three case studies: (1) parameterized results for concept satisfiability for $\mathcal{ALC}$ with respect to nearly acyclic TBoxes, (2) parameterized results for concept satisfiability for fragments of $\mathcal{ALC}$ that are close to $\mathcal{ALE}$, $\mathcal{ALU}$ and $\mathcal{AL}$, respectively, and (3) parameterized results addressing the notion of data complexity for instance checking and conjunctive query entailment for $\mathcal{ELI}$. The complexity

results that we obtain are summarized in Tables 1, 3 and 4—at the end of the sections where we present the case studies.

**Outline.** We begin by giving an overview of the theory of parameterized complexity—including commonly used (and more traditional) concepts and tools, as well as more progressive notions. Then we present our three case studies in three separate sections, before sketching directions for future research and concluding.

## Parameterized Complexity Theory

We begin by introducing relevant concepts from the theory of parameterized complexity. For more details, we refer to textbooks on the topic (Downey and Fellows 2013; Flum and Grohe 2006). We introduce both concepts that are used commonly in parameterized complexity analyses in the literature and less commonly used concepts, that play a role in this paper.

**FPT and XP.** The core notion in parameterized complexity is that of fixed-parameter tractability, which is a relaxation of the traditional notion of polynomial-time solvability. Fixed-parameter tractability is a property of parameterized problems. A *parameterized problem* $Q$ is a subset of $\Sigma^* \times \mathbb{N}$, for some finite alphabet $\Sigma$. An instance of a parameterized problem is a pair $(x, k)$ where $x$ is the main part of the instance, and $k$ is the parameter. Intuitively, the parameter captures some type of structure of the instance that could potentially be exploited algorithmically—the smaller the value of the parameter $k$, the more structure there is in the instance. (When considering multiple parameters, we take their sum as a single parameter.) A parameterized problem is *fixed-parameter tractable* if instances $(x, k)$ of the problem can be solved by a deterministic algorithm that runs in time $f(k)|x|^{O(1)}$, where $f$ is a computable function of $k$. Algorithms running within such time bounds are called *fpt-algorithms*. FPT denotes the class of all parameterized problems that are fixed-parameter tractable.

Intuitively, the idea behind fixed-parameter tractability is that whenever the parameter value $k$ is small, the overall running time is reasonably small—assuming that the constant hidden behind $O(1)$ is small. In fact, for every fixed parameter value $k$, the running time of an fpt-algorithm is polynomial (where the order of the polynomial is constant).

A related parameterized complexity class is XP, which consists of all parameterized problems for which instances $(x, k)$ can be solved in time $n^{f(k)}$, for some computable function $f$. Algorithms running within such time bounds are called *xp-algorithms*. That is, a parameterized problem $Q$ is in XP if there is an algorithm that solves $Q$ in polynomial time for each fixed value $k$ of the parameter—where the order of the polynomial may grow with $k$. It holds that FPT $\subsetneq$ XP. Intuitively, if a parameterized problem is in XP $\setminus$ FPT, it is not likely to be efficiently solvable in practice. Suppose, for example, that a problem is solvable in time $n^k$ in the worst case. Then already for $n = 100$ and $k = 10$, it could take ages to solve this problem (see, e.g., Downey 2012).

**Completeness Theory.** Parameterized complexity also offers a *completeness theory*, similar to the theory of NP-completeness, that provides a way to obtain evidence that a parameterized problem is not fixed-parameter tractable. Hardness for parameterized complexity classes is based on fpt-reductions, which are many-one reductions where the parameter of one problem maps into the parameter for the other. More specifically, a parameterized problem $Q$ is fpt-reducible to another parameterized problem $Q'$ if there is a mapping $R$ that maps instances of $Q$ to instances of $Q'$ such that (i) $(I, k) \in Q$ if and only if $R(I, k) = (I', k') \in Q'$, (ii) $k' \leq g(k)$ for a computable function $g$, and (iii) $R$ can be computed in time $f(k)|I|^c$ for a computable function $f$ and a constant $c$. A problem $Q$ is *hard* for a parameterized complexity class K if every problem $Q' \in$ K can be fpt-reduced to $Q$. A problem $Q$ is *complete* for a parameterized complexity class K if $Q \in$ K and $Q$ is K-hard.

Central to the completeness theory are the classes W[1] $\subseteq$ W[2] $\subseteq \ldots \subseteq$ W[P] $\subseteq$ XP of the Weft hierarchy. We will not define the classes W[$t$] in detail (for details, see, e.g., Flum and Grohe 2006). It suffices to note that it is widely believed that W[1] $\neq$ FPT.[1] Thus, showing that a problem $Q$ is W[1]-hard gives evidence that $Q$ is not fpt-time solvable.

An example of a W[1]-complete parameterized problem is CLIQUE (Downey and Fellows 1995; 2013). Instances for this problem consist of $(G, k)$, where $G = (V, E)$ is an undirected graph, and $k \in \mathbb{N}$. The parameter is $k$, and the question is to decide whether $G$ contains a clique of size $k$.

**Para-K.** For each classical complexity class K, we can construct a parameterized analogue para-K (Flum and Grohe 2003). Let K be a classical complexity class, e.g., NP. The parameterized complexity class para-K is then defined as the class of all parameterized problems $L \subseteq \Sigma^* \times \mathbb{N}$ for which there exist a computable function $f : \mathbb{N} \to \Sigma^*$ and a problem $Q' \subseteq \Sigma^* \times \Sigma^*$ in $K$, such that for all instances $(x, k) \in \Sigma^* \times \mathbb{N}$ it holds that $(x, k) \in Q$ if and only if $(x, f(k)) \in Q'$. Intuitively, the class para-$K$ consists of all problems that are in K after a precomputation that only involves the parameter. A common example of such parameterized analogues of classical complexity classes is the parameterized complexity class para-NP. Another example is para-P = FPT.

If (the unparameterized variant of) a parameterized problem $Q$ is in the class K, then $Q \in$ para-K. Also, if $Q$ is already K-hard for a finite set of parameter values, then $Q$ is para-K-hard (Flum and Grohe 2003).

Using the classes para-K and the notion of fpt-reductions, one can also provide evidence that certain parameterized problems are not fixed-parameter tractable. If a para-K-hard parameterized problem is fixed-parameter tractable, then K = P. For example, a para-NP-hard parameterized problem is not fixed-parameter tractable, unless P = NP.

**Para-NP and para-co-NP.** The classes para-NP and para-co-NP are parameterized analogues of the classes NP and co-NP. The class para-NP can alternatively be defined as the class of parameterized problems that are solvable

[1] In fact, it holds that W[1] $\neq$ FPT, assuming that $n$-variable 3SAT cannot be solved in subexponential time, that is, in time $2^{o(n)}$ (Chen et al. 2005; Chen and Kanj 2012; Downey and Fellows 2013).

in fpt-time by a non-deterministic algorithm (Flum and Grohe 2003). Similarly, para-co-NP can be defined using fpt-algorithms using universal nondeterminism—i.e., nondeterministic fpt-algorithms that reject the input if at least one sequence of nondeterministic choices leads the algorithm to reject. It holds that W[1] $\subseteq$ W[2] $\subseteq \cdots \subseteq$ W[P] $\subseteq$ para-NP.

Another alternative definition of the class para-NP—that can be motivated by the amazing practical performance of SAT solving algorithms (see, e.g., Biere et al. 2009)—is using the following parameterized variant of the propositional satisfiability problem (De Haan 2016; De Haan and Szeider 2014a; 2014b; 2017). Let $\mathrm{SAT}_1 = \{\, (\varphi, 1) : \varphi \in \mathrm{SAT} \,\}$ be the problem SAT with a constant parameter $k = 1$. The class para-NP consists of all problems that can be fpt-reduced to $\mathrm{SAT}_1$. In other words, para-NP can be seen as the class of all parameterized problems that can be solved by (1) a fixed-parameter tractable encoding into SAT, and (2) using a SAT solving algorithm to then decide the problem. The class para-co-NP can be characterized in a similar way, using UNSAT instead of SAT. Consequently, problems in para-co-NP can also be solved using the combination of an fpt-time encoding and a SAT solving algorithm.

**Para-PSPACE.** The class para-PSPACE can alternatively be defined as the class of all parameterized problems $Q$ for which there exists a (deterministic or nondeterministic) algorithm deciding whether $(x, k) \in Q$ using space $f(k)|x|^{O(1)}$, for some computable function $f$. It holds that para-NP $\cup$ para-co-NP $\subseteq$ para-PSPACE.

Another alternative characterization of para-PSPACE is using a parameterized variant of TQBF—the problem of deciding whether a given quantified Boolean formula is true. Let $\mathrm{TQBF}_1 = \{\, (\varphi, 1) : \varphi \in \mathrm{TQBF} \,\}$ be the problem TQBF with a constant parameter $k = 1$. The class para-PSPACE consists of all problems that can be fpt-reduced to $\mathrm{TQBF}_1$. In other words, para-PSPACE can be seen as the class of all parameterized problems that can be solved by (1) an fpt-time encoding into TQBF, and (2) using a TQBF solver to then decide the problem (see, e.g., Biere et al. 2009).

Yet another characterization of para-PSPACE uses alternating Turing machines (ATMs). An ATM is a nondeterministic Turing machine where the states are partitioned into existential and universal states (see, e.g., Flum and Grohe 2006, Appendix A.1). A configuration of the ATM with an existential state is accepting if at least one successor configuration is accepting, and a configuration with a universal state is accepting if all successor configurations are accepting. Intuitively, an ATM can alternate between existential and universal nondeterminism. The class para-PSPACE consists of all parameterized problems that can be decided by an ATM in fixed-parameter tractable time.

**Para-EXPTIME.** The class para-EXPTIME can be defined as the class of all parameterized problems $Q$ for which there exists a deterministic algorithm deciding whether $(x, k) \in Q$ in time $f(k)2^{|x|^{O(1)}}$, for some computable function $f$. It holds that para-PSPACE $\subseteq$ para-EXPTIME and that XP $\subseteq$ para-EXPTIME.

For an overview of all parameterized complexity classes that feature in this paper—and their relation—see Figure 1.

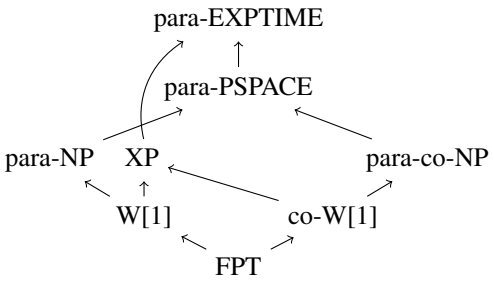


Figure 1: An overview of the landscape of parameterized complexity classes that play a role in this paper.

## Case Study 1: Concept Satisfiability for $\mathcal{ALC}$ with respect to Nearly Acyclic TBoxes

In this section, we provide our first case study to illustrate how parameterized complexity can be used to obtain a more detailed image of the computational complexity of description logic reasoning. In particular, we consider the problem of concept satisfiability for the description logic $\mathcal{ALC}$ with respect to general TBoxes. This problem is EXPTIME-complete in general. We consider two parameters for this problem. One of these parameters does not help to reduce the complexity of the problem—that is, for this parameter the problem is para-EXPTIME-complete. The other of the two parameters does help to reduce the complexity of the problem—that is, for this parameter the problem is para-PSPACE-complete.

We begin by revisiting the description logic $\mathcal{ALC}$, the problem of concept satisfiability with respect to acyclic and general TBoxes, and classical complexity results for this problem. We then discuss our parameterized complexity results, and how to interpret these results.

### The Description Logic $\mathcal{ALC}$

Let $N_C$, $N_R$ and $N_O$ be sets of *atomic concepts*, *roles*, and *individuals*, respectively. The triple $(N_C, N_R, N_O)$ is called the *signature*. (We will often omit the signature if this is clear from the context.)

Concepts $C$ are defined by the following grammar in Backus-Naur form, for $R \in N_R$ and $A \in N_C$:

$$C := A \mid \top \mid \bot \mid \neg C \mid C \sqcap C \mid C \sqcup C \mid \exists R.C \mid \forall R.C.$$

An *interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ over a signature $(N_C, N_R, N_O)$ consists of a non-empty set $\Delta^{\mathcal{I}}$ called the *domain*, and an interpretation function $\cdot^{\mathcal{I}}$ that maps (1) every individual $a \in N_O$ to an element $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$, (2) every concept $C$ to a subset of $\Delta^{\mathcal{I}}$, and (3) every role $R \in N_R$ to a subset of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$, such that:

- $\top^{\mathcal{I}} = \Delta^{\mathcal{I}}$; $\bot^{\mathcal{I}} = \emptyset$;
- $(C_1 \sqcap C_2)^{\mathcal{I}} = (C_1)^{\mathcal{I}} \cap (C_2)^{\mathcal{I}}$;
- $(\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$;
- $(C_1 \sqcup C_2)^{\mathcal{I}} = (C_1)^{\mathcal{I}} \cup (C_2)^{\mathcal{I}}$;
- $(\exists R.C)^{\mathcal{I}} = \{\, x \in \Delta^{\mathcal{I}} :$ there exists some $y \in C^{\mathcal{I}}$ such that $(x, y) \in R^{\mathcal{I}} \,\}$; and
- $(\forall R.C)^{\mathcal{I}} = \{\, x \in \Delta^{\mathcal{I}} :$ for each $y$ such that $(x, y) \in R^{\mathcal{I}}$ it holds that $y \in C^{\mathcal{I}} \,\}$.

A *general concept inclusion (GCI)* is a statement of the form $C \sqsubseteq D$, where $C, D$ are concepts. We write $\mathcal{I} \models C \sqsubseteq D$ (and say that $\mathcal{I}$ satisfies $C \sqsubseteq D$) if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$. A *(general) TBox* $\mathcal{T}$ is a finite set of GCIs. A *concept definition* is a statement of the form $A \equiv C$, where $A \in N_C$ is an atomic concept, and $C$ is a concept. We write $\mathcal{I} \models A \equiv C$ (and say that $\mathcal{I}$ satisfies $A \equiv C$) if $A^{\mathcal{I}} = C^{\mathcal{I}}$. An *acyclic TBox* $\mathcal{T}$ is a finite set of concept definitions such that (1) $\mathcal{T}$ does not contain two different concept definitions $A \equiv C_1$ and $A \equiv C_2$ for any $A \in N_C$, and (2) $\mathcal{T}$ contains no (direct or indirect) cyclic definitions—that is, the graph $G_{\mathcal{T}}$ with vertex set $N_C$ that contains an edge $(A, B)$ if and only if $\mathcal{T}$ contains a concept definition $A \equiv C$ where $B$ occurs in $C$ is acyclic. An interpretation $\mathcal{I}$ satisfies a (general or acyclic) TBox $\mathcal{T}$ if $\mathcal{I}$ satisfies all GCIs or concept definitions in $\mathcal{T}$.

A *concept assertion* is a statement of the form $C(a)$, where $a \in N_O$ and $C$ is a concept. A *role assertion* is a statement of the form $R(a, b)$, where $a, b \in N_O$ and $R \in N_R$. We write $\mathcal{I} \models C(a)$ (and say that $\mathcal{I}$ satisfies $C(a)$) if $a^{\mathcal{I}} \in C^{\mathcal{I}}$. Moreover, we write $\mathcal{I} \models R(a, b)$ (and say that $\mathcal{I}$ satisfies $R(a, b)$) if $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$. An *ABox* $\mathcal{A}$ is a finite set of concept and role assertions.

## Classical Complexity Results

An important reasoning problem for description logics is the problem of *concept satisfiability*. In this decision problem, the input consists of a concept $C$ and a TBox $\mathcal{T}$, and the question is whether $C$ is satisfiable with respect to $\mathcal{T}$—that is, whether there exists an interpretation $\mathcal{I}$ such that $\mathcal{I} \models \mathcal{T}$ and $C^{\mathcal{I}} \neq \emptyset$. The problem of concept satisfiability is PSPACE-complete, both for the case where $\mathcal{T}$ is empty and for the case where $\mathcal{T}$ is an acyclic TBox. For the case where $\mathcal{T}$ is a general TBox, the problem is EXPTIME-complete.

**Proposition 1** (Donini and Massacci 2000; Schild 1991)**.** *Concept satisfiability for the logic $\mathcal{ALC}$ with respect to general TBoxes is* EXPTIME*-complete.*

**Proposition 2** (Baader et al. 2005; Schmidt-Schauß and Smolka 1991)**.** *Concept satisfiability for the logic $\mathcal{ALC}$ with respect to acyclic TBoxes is* PSPACE*-complete.*

## Parameterized Complexity Results

We consider a parameterized variant of the problem of concept satisfiability for $\mathcal{ALC}$ where the parameter captures the distance towards acyclicity for the given TBox. That is, for this parameterized problem, the input consists of a concept $C$, an acyclic TBox $\mathcal{T}_1$, and a general TBox $\mathcal{T}_2$. The parameter is $k = |\mathcal{T}_2|$, and the question is whether $C$ is satisfiable with respect to $\mathcal{T}_1 \cup \mathcal{T}_2$—that is, whether there exists an interpretation $\mathcal{I}$ such that $\mathcal{I} \models \mathcal{T}_1$, $\mathcal{I} \models \mathcal{T}_2$, and $C^{\mathcal{I}} \neq \emptyset$.

Parameterizing by the size of $\mathcal{T}_2$ does not offer an improvement in the complexity of the problem—that is, this parameter leads to para-EXPTIME-completeness.

**Theorem 3.** *Concept satisfiability for $\mathcal{ALC}$ with respect to both an acyclic TBox $\mathcal{T}_1$ and a general TBox $\mathcal{T}_2$ is* para-EXPTIME*-complete when parameterized by $|\mathcal{T}_2|$.*

*Proof.* Membership in para-EXPTIME follows from the fact that the unparameterized version of the problem is in EXPTIME. To show para-EXPTIME-hardness, it suffices to show that the problem is already EXPTIME-hard for a constant value of the parameter (Flum and Grohe 2003). We do so by giving a reduction from the problem of concept satisfiability for $\mathcal{ALC}$ with respect to general TBoxes.

Let $C$ be a concept and let $\mathcal{T}$ be a general TBox. Moreover, let $\mathcal{T} = \{C_1 \sqsubseteq D_1, \ldots, C_m \sqsubseteq D_m\}$. We construct an acyclic TBox $\mathcal{T}_1$ and a general TBox $\mathcal{T}_2$ such that $C$ is satisfiable with respect to $\mathcal{T}$ if and only if it is satisfiable with respect to $\mathcal{T}_1 \cup \mathcal{T}_2$. Let $A$ be a fresh atomic concept. We let $\mathcal{T}_1 = \{A \equiv \bigsqcap_{i=1}^{m} (\neg C_i \sqcup D_i)\}$, and we let $\mathcal{T}_2 = \{\top \sqsubseteq A\}$. It is straightforward to verify that $C$ is satisfiable with respect to $\mathcal{T}$ if and only if it is satisfiable with respect to $\mathcal{T}_1 \cup \mathcal{T}_2$. Moreover, $|\mathcal{T}_2|$ is constant. From this, we can conclude that the problem is para-EXPTIME-hard. □

Intuitively, restricting only the number (and size) of the general TBox $\mathcal{T}_2$ does not restrict the problem, as we can encode a general TBox of arbitrary size in the acyclic TBox (together with a small general TBox). If we restrict the number of concepts impacted by the general TBox, however, we do get an improvement in the complexity of the problem.

Let $\mathcal{T}_1$ be an acyclic TBox and let $\mathcal{T}_2$ be a general TBox. We define the set of *concepts impacted by $\mathcal{T}_2$ (w.r.t. $\mathcal{T}_1$)* as the smallest set $I$ of concepts that is closed under (syntactic) subconcepts and that satisfies that (A) whenever $C \sqsubseteq D \in \mathcal{T}_2$, then $C, D \in I$, and (B) whenever $A \in I$ and $A \equiv C \in \mathcal{T}_1$, then $C \in I$. If we parameterize the problem of concept satisfiability with respect to both an acyclic TBox $\mathcal{T}_1$ and a general TBox $\mathcal{T}_2$ by the number of concepts impacted by $\mathcal{T}_2$, the complexity of the problem jumps down to para-PSPACE.

**Theorem 4.** *Concept satisfiability for $\mathcal{ALC}$ with respect to both an acyclic TBox $\mathcal{T}_1$ and a general TBox $\mathcal{T}_2$ is* para-PSPACE*-complete when parameterized by the number $k$ of concepts that are impacted by $\mathcal{T}_2$ (w.r.t. $\mathcal{T}_1$).*

*Proof.* Hardness for para-PSPACE follows directly from the fact that the problem is already PSPACE-hard when $\mathcal{T}_2$ is empty (Proposition 2)—and thus the number of concepts impacted by $\mathcal{T}_2$ is 0. We show membership in para-PSPACE by exhibiting a nondeterministic algorithm to solve the problem that runs in space $f(k) \cdot n^{O(1)}$, for some computable function $f$. Let $C$ be a concept, let $\mathcal{T}_1$ be an acyclic TBox, and let $\mathcal{T}_2$ be a general TBox. We may assume without loss of generality that all concepts occurring in $\mathcal{T}_1$ are in negation normal form—that is, negations occur only directly in front of atomic concepts. If this were not the case, we could straightforwardly transform $\mathcal{T}_1$ to a TBox that does have this property in polynomial time, by introducing new atomic concepts $A'$ for any negated concept $\neg A$.

The algorithm that we use is the usual tableau algorithm (with static blocking) for $\mathcal{ALC}$—see, e.g., (Baader and Sattler 2001). That is, it aims to construct a tree that can be used to construct an interpretation satisfying $C$, $\mathcal{T}_1$ and $\mathcal{T}_2$. For each node in the tree, it first exhaustively applies the rules for the $\sqcup$ and $\sqcap$ operators, the rules for the concept definitions in $\mathcal{T}_1$, and the rules for the GCIs in $\mathcal{T}_2$ (for each $C \sqsubseteq D \in \mathcal{T}_2$ adding the concept $\neg C \sqsubseteq D$ to a node), before applying the rules for the $\exists$ and $\forall$ operators. Moreover, it applies all rules

exhaustively to one node of the tree before moving to another node. Additionally, the algorithm uses the following usual blocking condition (subset blocking): the rule for the $\exists$ operator cannot be applied to a node $x$ that has a predecessor $y$ in the tree that is labelled with all concepts that $x$ is labelled with (and possibly more). It is straightforward to verify that this tableau algorithm correctly decides the problem.

We argue that this algorithm requires space $2^k \cdot n^{O(1)}$, where $k$ is the number of concepts impacted by $\mathcal{T}_2$ and $n$ denotes the input size. It is straightforward to verify that there is a polynomial $p$ such that each node in the tree constructed by the tableau algorithm that is more than $p(n)$ steps away from the root of the tree is only labelled with concepts that are impacted by $\mathcal{T}_2$. Since there are only $k$ concepts that are impacted by $\mathcal{T}_2$, we know that in each branch of the tree, the blocking condition applies at depth at most $2^k \cdot p(n)$, and thus that each branch is of length at most $2^k \cdot p(n)$. From this, it follows that this algorithm requires space $2^k \cdot n^{O(1)}$, and thus that the problem is in para-PSPACE. □

### Interpretation of the Results

The results in this section are summarized in Table 1. The parameterized results of Theorems 3 and 4 show that parameterized complexity theory can make a distinction between the complexity of the two variants of the problem that classical complexity theory is blind to. Classically, both variants are EXPTIME-complete, but one parameter can be used to get a polynomial-space algorithm (when an additional $2^k$ factor for the parameter $k$), whereas the other parameter requires exponential space, no matter what additional $f(k)$ factor is allowed. The para-PSPACE result of Theorem 4 also yields an algorithm solving the problem using (1) an fpt-time encoding into the problem TQBF, and then (2) using a TQBF solver to decide the problem (see, e.g., Biere et al. 2009).

| *parameter* | *complexity of $\mathcal{ALC}$ concept satisfiability w.r.t. $\mathcal{T}_1$ and $\mathcal{T}_2$* | |
|---|---|---|
| – | EXPTIME-c | (Proposition 1) |
| $\lvert\mathcal{T}_2\rvert$ | para-EXPTIME-c | (Theorem 3) |
| # of concepts impacted by $\mathcal{T}_2$ (w.r.t. $\mathcal{T}_1$) | para-PSPACE-c | (Theorem 4) |

Table 1: The parameterized complexity of $\mathcal{ALC}$ concept satisfiability w.r.t. both an acyclic TBox $\mathcal{T}_1$ and a general TBox $\mathcal{T}_2$, for different parameters.

## Case Study 2: Concept Satisfiability for $\mathcal{ALC}$, $\mathcal{ALE}$, $\mathcal{ALU}$ and $\mathcal{AL}$

In this section, we provide our second case study to illustrate how parameterized complexity can be used to obtain a more detailed image of the computational complexity of description logic reasoning. In particular, we consider the problem of concept satisfiability for the description logic $\mathcal{ALC}$. This problem is PSPACE-complete in general. We consider several parameters that measure the distance to the logics $\mathcal{ALE}$ and $\mathcal{ALU}$. The logics $\mathcal{ALE}$ and $\mathcal{ALU}$ are obtained from $\mathcal{ALC}$ by disallowing concept union and full existential qualification, respectively. The parameters that we consider both help to reduce the complexity of the problem. One parameter renders the problem para-co-NP-complete. The other parameter renders the problem para-NP-complete. The combination of both parameters renders the problem fixed-parameter tractable.

We begin by revisiting the description logics $\mathcal{ALE}$ and $\mathcal{ALU}$ (and their intersection $\mathcal{AL}$), and classical complexity results for the problem of concept satisfiability for these logics. We then discuss our parameterized complexity results, and how to interpret these results.

### The Description Logics $\mathcal{ALE}$, $\mathcal{ALU}$ and $\mathcal{AL}$

In order to obtain the description logics $\mathcal{ALE}$, $\mathcal{ALU}$ and $\mathcal{AL}$, we consider a (syntactic) variant of the logic $\mathcal{ALC}$ where all concepts are in *negation normal form*. That is, negations only occur immediately followed by atomic concepts. Put differently, we consider concepts $C$ that are defined as follows, for $R \in N_R$ and $A \in N_C$:

$$C := A \mid \neg A \mid \top \mid \bot \mid C \sqcap C \mid C \sqcup C \mid \exists R.C \mid \forall R.C.$$

One can transform any $\mathcal{ALC}$ concept into negation normal form in linear time (see, e.g., Baader, Horrocks, and Sattler 2008). The semantics of this variant of $\mathcal{ALC}$ is defined exactly as described in the previous section. Throughout this section, we will only consider this variant of $\mathcal{ALC}$.

The description logic $\mathcal{ALE}$ is obtained from the logic $\mathcal{ALC}$ by forbidding any occurrence of the operator $\sqcup$. The description logic $\mathcal{ALU}$ is obtained from the logic $\mathcal{ALC}$ by requiring that for every occurrence $\exists R.C$ of the existential quantifier it holds that $C = \top$; that is, only limited existential quantification $\exists R.\top$ is allowed. The description logic $\mathcal{AL}$ contains those concepts that are concepts in both $\mathcal{ALE}$ and $\mathcal{ALU}$—that is, $\mathcal{AL}$ is the intersection of $\mathcal{ALE}$ and $\mathcal{ALU}$.

Thus, the logics $\mathcal{ALE}$ and $\mathcal{ALU}$ are obtained from $\mathcal{ALC}$ by means of two orthogonal restrictions: disallowing concept union and replacing full existential qualification by limited existential quantification, respectively. The logic $\mathcal{AL}$ is obtained from $\mathcal{ALC}$ by using both of these restrictions.

### Classical Complexity Results

In this section, we consider the problem of *concept satisfiability* with respect to empty TBoxes. In this decision problem, the input consists of a concept $C$, and the question is whether $C$ is satisfiable—that is, whether there exists an interpretation $\mathcal{I}$ such that $C^{\mathcal{I}} \neq \emptyset$. This problem is PSPACE-complete for $\mathcal{ALC}$, co-NP-complete for $\mathcal{ALE}$, NP-complete for $\mathcal{ALU}$, and polynomial-time solvable for $\mathcal{AL}$.

**Proposition 5** (Schmidt-Schauß and Smolka 1991)**.** *Concept satisfiability for the logic $\mathcal{ALC}$ is* PSPACE*-complete.*

**Proposition 6** (Donini et al. 1992)**.** *Concept satisfiability for the logic $\mathcal{ALE}$ is* co-NP*-complete.*

**Proposition 7** (Donini et al. 1997)**.** *Concept satisfiability for the logic $\mathcal{ALU}$ is* NP*-complete.*

**Proposition 8** (Schmidt-Schauß and Smolka 1991)**.** *Concept satisfiability for the logic $\mathcal{AL}$ is polynomial-time solvable.*

| | |
|---|---|
| **The $\sqcap$-rule** | |
| *Condition* | $\mathcal{A}$ contains $(C_1 \sqcap C_2)(x)$, but it does not contain both $C_1(x)$ and $C_2(x)$. |
| *Action* | $\mathcal{A}' = \mathcal{A} \cup \{C_1(x), C_2(x)\}$. |
| **The $\sqcup$-rule** | |
| *Condition* | $\mathcal{A}$ contains $(C_1 \sqcup C_2)(x)$, but neither $C_1(x)$ nor $C_2(x)$. |
| *Action* | $\mathcal{A}' = \mathcal{A} \cup \{C_1(x)\}$, $\mathcal{A}'' = \mathcal{A} \cup \{C_2(x)\}$. |
| **The $\exists$-rule** | |
| *Condition* | $\mathcal{A}$ contains $(\exists R.C)(x)$, but there is no individual $z$ such that $C(z)$ and $R(x, z)$ are in $\mathcal{A}$. |
| *Action* | $\mathcal{A}' = \mathcal{A} \cup \{C(y), R(x, y)\} \setminus \{ (\exists R'.C')(x') \in \mathcal{A} : x' = x, (\exists R'.C') \neq (\exists R.C) \}$, where $y$ is an arbitrary individual not occurring in $\mathcal{A}$. |
| **The $\forall$-rule** | |
| *Condition* | $\mathcal{A}$ contains $(\forall R.C)(x)$ and $R(x, y)$, but it does not contain $C(y)$. |
| *Action* | $\mathcal{A}' = \mathcal{A} \cup \{C(y)\}$. |
| **The $\bot$-rule** | |
| *Condition* | $\mathcal{A}$ contains $A(x)$ and $(\neg A)(x)$, but it does not contain $\bot$. |
| *Action* | $\mathcal{A}' = \mathcal{A} \cup \{\bot\}$. |

Table 2: Transformation rules of the tableau algorithm for $\mathcal{ALC}$ concept satisfiability.

## Parameterized Complexity Results

In order to conveniently describe the parameterized complexity results that we will establish in this section, we firstly describe an algorithm for deciding concept satisfiability for $\mathcal{ALC}$ in polynomial space (see, e.g., Baader et al. 2003, Chapter 2). To use this algorithm to prove the parameterized complexity results in this section, we describe a variant of the algorithm that can be implemented by a polynomial-time alternating Turing machine—i.e., a nondeterministic Turing machine that can alternate between existential and universal nondeterminism.

The algorithm uses ABoxes $\mathcal{A}$ as data structures, and works by extending these ABoxes by means of several transformation rules. These rules are described Table 2—however, not all rules are applied in the same fashion. The $\sqcap$-rule, the $\forall$-rule and the $\bot$-rule are used as deterministic rules, and are applied greedily whenever they apply. The $\sqcup$-rule and the $\exists$-rule are nondeterministic rules, but are used in a different fashion. The $\sqcup$-rule transforms an ABox $\mathcal{A}$ into one of two different ABoxes $\mathcal{A}'$ or $\mathcal{A}''$ nondeterministically. The $\sqcup$-rule is implemented using existential nondeterminism—i.e., the algorithm succeeds if at least one choice of $\mathcal{A}'$ and $\mathcal{A}''$ ultimately leads to the algorithm accepting. (For more details on existential and universal nondeterminism and alternating Turing machines, see, e.g., Flum and Grohe 2006, Appendix A.1.) The $\exists$-rule, on the other hand, transforms an ABox $\mathcal{A}$ into a unique next ABox $\mathcal{A}'$, but it is a nonmonotonic rule that can be applied in several ways—the condition can be instantiated in different ways, and these instantiations are not all possible anymore after having applied the rule. The $\exists$-rule is implemented using universal nondeterminism—i.e., the algorithm succeeds if all ways of instantiating the condition of the $\exists$-rule (and applying the rule accordingly) ultimately lead to the algorithm accepting.

The tableau algorithm works as follows. Let $C_0$ be an $\mathcal{ALC}$ concept for which we want to decide satisfiability. We construct an initial ABox $\mathcal{A}_0 = \{C_0(x_0)\}$, where $x_0 \in N_O$ is an arbitrary individual. We proceed in two alternating phases: (I) and (II)—starting with phase (I).

In phase (I), we apply the deterministic rules (the $\sqcap$-rule, the $\forall$-rule and the $\bot$-rule) and the nondeterministic $\sqcup$-rule exhaustively, until none of these rules is applicable anymore. For the $\sqcup$-rule we use existential nondeterminism to choose which of $\mathcal{A}'$ and $\mathcal{A}''$ to use. When none of these rules is applicable anymore, we proceed to phase (II). In phase (II), we apply the $\exists$-rule once, using universal nondeterminism to choose how to instantiate the condition (and we apply the rule accordingly). Then, we go back to phase (I).

Throughout the execution of the algorithm, there is always a single current ABox $\mathcal{A}$. Whenever it holds that $\bot \in \mathcal{A}$, the algorithm rejects. If at some point no rule is applicable anymore—that is, if at some point we are in phase (II) and the $\exists$-rule is not applicable—the algorithm accepts.

This algorithm essentially works the same way as known tableau algorithms for $\mathcal{ALC}$ concept satisfiability (see, e.g., Baader et al. 2003, Chapter 2). The only difference is that in the algorithm described above the implementation of the $\sqcup$-rule using existential nondeterminism and the implementation of the $\exists$-rule using universal nondeterminism is built in. In the literature, typically descriptions of tableau algorithms leave freedom for different implementations of the way in which the search tree is traversed. One can think of the algorithm described above as traversing a search tree that is generated by the different (existential and universal) nondeterministic choices that are made in the execution of the algorithm. This search tree is equivalent to the search tree of the usual tableau algorithm for $\mathcal{ALC}$ concept satisfiability. Thus, we get that the algorithm is correct. In fact, this algorithm is a reformulation of the standard algorithm known from the literature (Baader et al. 2003).

**Proposition 9** (Baader et al. 2003)**.** *The tableau algorithm described above for an alternating polynomial-time Turing machine correctly decides concept satisfiability for $\mathcal{ALC}$.*

We will now consider several parameterized variants of the problem of concept satisfiability for $\mathcal{ALC}$. These parameters, in a sense, measure the distance of an $\mathcal{ALC}$ concept to the logics $\mathcal{ALE}$, $\mathcal{ALU}$ and $\mathcal{AL}$, respectively. We will make use of the tableau algorithm described above to establish upper bounds on the complexity of these problems. Lower bounds follow directly from Propositions 6–8.

We begin with the parameterized variant of $\mathcal{ALC}$ concept satisfiability where the parameter measures the distance to $\mathcal{ALE}$.

**Theorem 10.** *Concept satisfiability for the logic $\mathcal{ALC}$, parameterized by the number of occurrences of the union operator $\sqcup$ in $C$, is* para-co-NP-*complete.*

*Proof.* Hardness for para-co-NP follows from the fact that $\mathcal{ALE}$ concept satisfiability is co-NP-complete (Proposition 6). Any $\mathcal{ALE}$ concept is an $\mathcal{ALC}$ concept with zero occurrences of the union operator $\sqcup$. Therefore, the problem of $\mathcal{ALC}$ concept satisfiability parameterized by the number $k$ of occurrences of the union operator $\sqcup$ in $C$ is already co-NP-hard for the parameter value $k = 0$. From this, it follows that the parameterized problem is para-co-NP-hard (Flum and Grohe 2003).

To show that the parameterized problem is also contained in para-co-NP, we describe an algorithm that can be implemented by an alternating Turing machine that only makes use of universal nondeterminism and that runs in fixed-parameter tractable time. This algorithm is similar to the tableau algorithm for $\mathcal{ALC}$ described above, with the only difference that the $\sqcup$-rule is now not implemented using existential nondeterminism. Instead, we deterministically iterate over all possible choices that can be made in executions of the $\sqcup$-rule. That is, whenever the $\sqcup$-rule is applied, resulting in two possible next ABoxes $\mathcal{A}'$ and $\mathcal{A}''$, we firstly continue the algorithm with $\mathcal{A}'$, and if the continuation of the algorithm with $\mathcal{A}'$ failed, we then continue the algorithm with $\mathcal{A}''$ instead.

Let $k$ be the number of occurrences of the union operator $\sqcup$ in $C$. For each occurrence, the $\sqcup$-rule is applied at most once. Therefore, the total number of possible choices resulting from executions of the $\sqcup$-rule is at most $2^k$. Therefore, this modification of the algorithm can be implemented by an alternating Turing machine that only uses universal nondeterminism and that runs in time $2^k \cdot |C|^{O(1)}$. In other words, the problem is in para-co-NP, and thus is para-co-NP-complete. □

**Theorem 11.** *Concept satisfiability for the logic $\mathcal{ALC}$, parameterized by the number of occurrences of full existential qualification $\exists R.C$ in $C$, is* para-NP-*complete.*

*Proof.* Hardness for para-NP follows from the fact that $\mathcal{ALU}$ concept satisfiability is NP-complete (Proposition 7). Any $\mathcal{ALU}$ concept is an $\mathcal{ALC}$ concept with zero occurrences of full existential qualification $\exists R.C$. Therefore, the problem of $\mathcal{ALC}$ concept satisfiability parameterized by the number of occurrences of full existential qualification $\exists R.C$ in $C$ is already NP-hard for the parameter value $k = 0$. From this, it follows that the parameterized problem is para-NP-hard (Flum and Grohe 2003).

To show membership in para-NP, we modify the tableau algorithm for $\mathcal{ALC}$, similarly to the way we did in the proof of Theorem 10. In particular, we describe an algorithm that can be implemented by an alternating Turing machine that only makes use of existential nondeterminism and that runs in fixed-parameter tractable time. We do so by executing the $\exists$-rule deterministically, instead of using universal nondeterminism. That is, instead of using universal nondeterminism to choose which instantiation of the condition of the $\exists$-rule to use, we iterate over all possibilities deterministically.

Let $k$ be the number of occurrences of full existential quantification $\exists R.C$ in $C$. At each point, there are at most $k$ different ways of instantiating the $\exists$-rule. Moreover, after having applied the $\exists$-rule for at most $k$ times, the $\exists$-rule is not applicable anymore. Therefore, the total number of possible choices to iterate over is at most $k^k$. Therefore, this modification of the algorithm can be implemented by an alternating Turing machine that only uses existential nondeterminism and that runs in time $k^k \cdot |C|^{O(1)}$. In other words, the problem is in para-NP, and thus is para-NP-complete. □

**Theorem 12.** *Concept satisfiability for the logic $\mathcal{ALC}$, parameterized by both (i) the number of occurrences of the union operator $\sqcup$ in $C$ and (ii) the number of occurrences of full existential qualification $\exists R.C$ in $C$, is fixed-parameter tractable.*

*Proof (sketch).* We can modify the alternating polynomial-time tableau algorithm for $\mathcal{ALC}$ concept satisfiability to work in deterministic fpt-time by implementing both the $\sqcup$-rule and the $\exists$-rule deterministically, iterating sequentially over all possible choices that can be made for these rules. That is, we combine the ideas behind the proofs of Theorems 10 and 11. We omit the details of this fpt-time algorithm. □

## Interpretation of the Results

The results in this section are summarized in Table 3. Similarly as for the first case study, the results for the second case study show that parameterized complexity theory can make distinctions that classical complexity theory does not see. The problems studied in Theorems 10, 11 and 12 are all PSPACE-complete classically, yet from a parameterized point of view their complexity goes down to para-NP, para-co-NP and FPT. The para-NP- and para-co-NP-completeness results of Theorems 10 and 11 also yield algorithms that (1) firstly use an fpt-encoding to an instance of SAT and (2) then use a SAT solver to decide the problem (see, e.g., Biere et al. 2009).

| *parameter* | *complexity of $\mathcal{ALC}$ concept satisfiability* | |
|---|---|---|
| – | PSPACE-c | (Proposition 5) |
| # of occurrences of $\sqcup$ | para-co-NP-c | (Theorem 10) |
| # of occurrences of $\exists$ | para-NP-c | (Theorem 11) |
| # of occurrences of $\sqcup$ and $\exists$ | FPT | (Theorem 12) |

Table 3: The parameterized complexity of $\mathcal{ALC}$ concept satisfiability (with no TBoxes) for different parameters.

## Case Study 3: A Parameterized Complexity View on Data Complexity

In this section, we provide our third case study illustrating the use of parameterized complexity for the analysis of description logic reasoning. This third case study is about refining the complexity analysis for cases where one part of the input is much smaller than another part. Typically, these cases occur where there is a small TBox and a small query, but where there is a large database of facts (in the form of an ABox). What is often done is that the size of the TBox and the query are seen as fixed constants—and the complexity results are grouped under the name of "data complexity."

In this section, we will look at two concrete polynomial-time data complexity results for the description logic $\mathcal{ELI}$. Even though the data complexity view gives the same outlook on the complexity of these problems, we will use the viewpoint of parameterized complexity theory to argue that these two problems in fact have a different complexity. One of these problems is more efficiently solvable than the other.

We chose the example of $\mathcal{ELI}$ to illustrate our point because it is technically straightforward. More intricate fixed-parameter tractability results for conjunctive query answering in description logics have been obtained in the literature (Bienvenu et al. 2017a; 2017b; Kikot, Kontchakov, and Zakharyaschev 2011).

We begin by reviewing the description logic $\mathcal{ELI}$, and the two reasoning problems for this logic that we will look at (instance checking and conjunctive query entailment). We will review the classical complexity results for these two problems, including the data complexity results. We will then use results from the literature to give a parameterized complexity analysis for these two problems, and argue why the parameterized complexity perspective gives a more accurate view on the complexity of these problems.

### The Description Logic $\mathcal{ELI}$

To define the logic $\mathcal{ELI}$, we first consider the logic $\mathcal{EL}$. The description logic $\mathcal{EL}$ is obtained from the logic $\mathcal{ALC}$ by forbidding any use of the negation operator ($\neg$), the empty concept ($\bot$), the union operator ($\sqcup$), and universal quantification ($\forall R.C$). The description logic $\mathcal{ELI}$ is obtained from the logic $\mathcal{EL}$ by introducing *inverse roles*. That is, $\mathcal{ELI}$ concepts are defined by the following grammar in Backus-Naur form, for $R \in N_R$ and $A \in N_C$:

$$C := A \mid \top \mid C \sqcap C \mid \exists R.C \mid \exists R^{-}.C.$$

Interpretations $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ for $\mathcal{ELI}$ are defined as interpretations for $\mathcal{EL}$ with the following addition:

- $(\exists R^{-}.C)^{\mathcal{I}} = \{\, x \in \Delta^{\mathcal{I}} : \text{ there exists some } y \in C^{\mathcal{I}} \text{ such that } (y, x) \in R^{\mathcal{I}} \,\}$.

### Classical Complexity Results

We consider two reasoning problems for the logic $\mathcal{ELI}$. The first problem that we consider is the problem of *instance checking*. In this problem, the input consists of an ABox $\mathcal{A}$, a (general) TBox $\mathcal{T}$, an individual name $a$ and a concept $C$, and the question is whether $\mathcal{A}, \mathcal{T} \models C(a)$—that is, whether for each interpretation $\mathcal{I}$ such that $\mathcal{I} \models \mathcal{A}$ and $\mathcal{I} \models \mathcal{T}$ it holds that $\mathcal{I} \models C(a)$.

The second problem that we consider is the problem of *conjunctive query entailment* (which can be seen as a generalization of the problem of instance checking). A *conjunctive query* is a set $q$ of atoms of the form $C(v)$ and $R(u, v)$, where $C$ is a concept, where $R \in N_R$, and where $u, v$ are *variables*. Let $\mathrm{Var}(q)$ denote the set of variables occurring in $q$. Let $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ be an interpretation and let $\pi$ be a mapping from $\mathrm{Var}(q)$ to $\Delta^{\mathcal{I}}$. We write $\mathcal{I} \models^{\pi} C(v)$ if $\pi(v) \in C^{\mathcal{I}}$, we write $\mathcal{I} \models^{\pi} R(u, v)$ if $(\pi(u), \pi(v)) \in R^{\mathcal{I}}$, we write $\mathcal{I} \models^{\pi} q$ if $\mathcal{I} \models^{\pi} \alpha$ for all $\alpha \in q$, and we write $\mathcal{I} \models q$ if $\mathcal{I} \models^{\pi} q$ for some $\pi : \mathrm{Var}(q) \to \Delta^{\mathcal{I}}$. For any ABox $\mathcal{A}$ and TBox $\mathcal{T}$, we write $\mathcal{A}, \mathcal{T} \models q$ if $\mathcal{I} \models q$ for each interpretation $\mathcal{I}$ such that $\mathcal{I} \models \mathcal{A}$ and $\mathcal{I} \models \mathcal{T}$. In the problem of conjunctive query entailment, the input consists of a (general) TBox $\mathcal{T}$, an ABox $\mathcal{A}$, and a conjunctive query $q$, and the question is to decide whether $\mathcal{A}, \mathcal{T} \models q$.

Both the problem of instance checking and the problem of conjunctive query entailment for $\mathcal{ELI}$ are EXPTIME-complete in general.

**Proposition 13** (Baader, Brandt, and Lutz 2005; 2008)**.** *Instance checking for $\mathcal{ELI}$ is* EXPTIME*-complete.*

**Corollary 14** (Baader, Brandt, and Lutz 2005; 2008)**.** *Conjunctive query entailment for $\mathcal{ELI}$ is* EXPTIME*-complete.*

The results of Propositions 13 and Corollary 14 are typically called "combined complexity" results—meaning that all elements of the problem statement are given as inputs for the problem. To study how the complexity of these problems increases when the size of the ABox $\mathcal{A}$ grows—and when the size of the TBox $\mathcal{T}$ and the size of the query $q$ remain the same—often different variants of the problems are studied. In these variants, the TBox $\mathcal{T}$ and the query $q$ are fixed (and thus not part of the problem input), and only the ABox $\mathcal{A}$ is given as problem input. That is, there is a variant of the problem for each choice of $\mathcal{T}$ and $q$. The computational complexity of these problem variants are typically called the "data complexity" of the problem.

From a data complexity perspective, the problems of instance checking and conjunctive query entailment for the logic $\mathcal{ELI}$ are both polynomial-time solvable. In other words, from a data complexity point of view these problems are of the same complexity.

**Claim 15** (Krisnadhi 2007)**.** *Instance checking for $\mathcal{ELI}$ is polynomial-time solvable regarding data complexity.*

**Claim 16** (Krisnadhi and Lutz 2007)**.** *Conjunctive query entailment for $\mathcal{ELI}$ is polynomial-time solvable regarding data complexity.*

### Parameterized Complexity Results

We will argue that the computational complexity of the problems of instance checking and conjunctive query entailment for $\mathcal{ELI}$—when only the ABox $\mathcal{A}$ grows in size—is of a vastly different nature. We will do so by using the parameterized complexity methodology. Concretely, we will take (the size of) the TBox $\mathcal{T}$ and (for the case of conjunctive query entailment) the query $q$ as parameters, and observe that the parameterized complexity of these two problems is different.

We begin by observing that the algorithm witnessing polynomial-time data complexity for the problem of instance checking for $\mathcal{ELI}$ corresponds to an fpt-algorithm for the problem when parameterized by the size of the TBox $\mathcal{T}$.

**Observation 17.** *Instance checking for $\mathcal{ELI}$ is fixed-parameter tractable when parameterized by $|\mathcal{T}|$.*

*Proof.* The algorithm to solve the problem of instance checking for $\mathcal{ELI}$ described by Krisnadhi (2007, Proposition 4.3) runs in time $2^{|\mathcal{T}|^{O(1)}} \cdot |\mathcal{A}|^{O(1)}$. □

The polynomial-time data complexity algorithm for the problem of conjunctive query entailment, on the other hand, does not translate to an fpt-algorithm, but to an xp-algorithm instead—when the parameter is (the sum of) the size of the TBox $\mathcal{T}$ and the size of the query $q$.

**Observation 18.** *Conjunctive query entailment for $\mathcal{ELI}$ is in* XP *when parameterized by $|\mathcal{T}|$ and $|q|$.*

*Proof.* The algorithm to solve the problem of conjunctive query entailment for $\mathcal{ELI}$ described by Krisnadhi and Lutz (2007, Theorem 4) runs in time $(|\mathcal{A}| + |\mathcal{T}|)^{|q|^{O(1)}}$. □

For this parameter, the problem of conjunctive query entailment for $\mathcal{ELI}$ is in fact W[1]-hard—and thus not fixed-parameter tractable, assuming the widely believed conjecture that FPT $\neq$ W[1]. This follows immediately from the W[1]-hardness of conjunctive query answering over databases when parameterized by the size of the query (Papadimitriou and Yannakakis 1999, Theorem 1).

**Corollary 19** (Papadimitriou and Yannakakis 1999)**.** *Conjunctive query entailment for $\mathcal{ELI}$ is* W[1]*-hard when parameterized by $|\mathcal{T}|$ and $|q|$.*

### Interpretation of the Results

The results in this section are summarized in Table 4. Observation 17 and Corollary 19 show that parameterized complexity can give a more accurate view on data complexity results than classical complexity theory. From a classical complexity perspective, the data complexity variants of both problems are polynomial-time solvable, whereas the parameterized data complexity variants of the problems differ in complexity. Both problems are solvable in polynomial time when only the ABox $\mathcal{A}$ grows in size. However, for instance checking the order of the polynomial is constant (Observation 17), and for conjunctive query entailment the order of the polynomial grows with the size of the query $q$ (Corollary 19). This is a difference with enormous effects on the practicality of algorithms solving these problems (see, e.g., Downey 2012).

| | *instance checking* | *conjunctive query entailm.* |
|---|---|---|
| combined complexity | EXPTIME-c (Prop 13) | EXPTIME-c (Cor 14) |
| data complexity | in P (Claim 15) | in P (Claim 16) |
| combined complexity with parameter $\|\mathcal{T}\|+\|q\|$ | in FPT (Obs 17) | in XP (Obs 18)<br>W[1]-h (Cor 19) |

Table 4: (Parameterized) complexity results for instance checking and conjunctive query entailment for $\mathcal{ELI}$.

## Directions for Future Research

The results in this paper are merely an illustrative exposition of the type of parameterized complexity results that are possible for description logic reasoning problems when using less commonly studied concepts (e.g., the classes para-NP, para-co-NP and para-PSPACE). We hope that this paper sparks a structured investigation of the parameterized complexity of different reasoning problems for the wide range of description logics that have been studied. For this, it would be interesting to consider a large assortment of different parameters that could reasonably be expected to have small values in applications. It would also be interesting to investigate to what extent, say, para-NP-membership results can be used to develop practical algorithms based on the combination of fpt-time encodings into SAT and SAT solving algorithms.

## Conclusion

We showed how the complexity study of description logic problems can benefit from using the framework of parameterized complexity and all the tools and methods that it offers. We did so using three case studies. The first addressed the problem of concept satisfiability for $\mathcal{ALC}$ with respect to nearly acyclic TBoxes. The second was about the problem of concept satisfiability for fragments of $\mathcal{ALC}$ that are close to $\mathcal{ALE}$, $\mathcal{ALU}$ and $\mathcal{AL}$, respectively. The third case study concerned a parameterized complexity view on the notion of data complexity for instance checking and conjunctive query entailment for $\mathcal{ELI}$. Moreover, we sketched some directions for future research, applying (progressive notions from) parameterized complexity theory to the study of description logic reasoning problems.

**Acknowledgments.** This work was supported by the Austrian Science Fund (FWF), project J4047.

## References

Baader, F., and Sattler, U. 2001. An overview of tableau algorithms for description logics. *Studia Logica* 69(1):5–40.

Baader, F.; Calvanese, D.; McGuinness, D. L.; Nardi, D.; and Patel-Schneider, P. F. 2003. *The Description Logic Handbook: Theory, Implementation and Applications*.

Baader, F.; Lutz, C.; Miličić, M.; Sattler, U.; and Wolter, F. 2005. Integrating description logics and action formalisms for reasoning about web services.

Baader, F.; Brandt, S.; and Lutz, C. 2005. Pushing the $\mathcal{EL}$ envelope. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI 2005)*, 364–369.

Baader, F.; Brandt, S.; and Lutz, C. 2008. Pushing the $\mathcal{EL}$ envelope further. In *Proceedings of the OWLED 2008 DC Workshop on OWL: Experiences and Directions*.

Baader, F.; Horrocks, I.; and Sattler, U. 2008. Description logics. In *Handbook of Knowledge Representation*, volume 3 of *Foundations of Artificial Intelligence*. Elsevier. 135–179.

Bienvenu, M.; Kikot, S.; Kontchakov, R.; Podolskii, V. V.; Ryzhikov, V.; and Zakharyaschev, M. 2017a. The complexity of ontology-based data access with OWL 2 QL and bounded treewidth queries. In *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS 2017)*, 201–216. ACM.

Bienvenu, M.; Kikot, S.; Kontchakov, R.; Ryzhikov, V.; and Zakharyaschev, M. 2017b. On the parameterised complexity of tree-shaped ontology-mediated queries in OWL 2 QL. In *Proceedings of the 30th International Workshop on Description Logics (DL 2017)*.

Biere, A.; Heule, M.; van Maaren, H.; and Walsh, T., eds. 2009. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press.

Bodlaender, H. L.; Downey, R.; Fomin, F. V.; and Marx, D., eds. 2012. *The Multivariate Algorithmic Revolution and Beyond*. Springer Verlag.

Ceylan, I. I., and Peñaloza, R. 2014. The Bayesian description logic $\mathcal{BEL}$. In *Proceedings of the 7th International Joint Conference on Automated Reasoning (IJCAR 2014)*, 480–494.

Chen, J., and Kanj, I. A. 2012. Parameterized complexity and subexponential-time computability. In Bodlaender, H. L.; Downey, R.; Fomin, F. V.; and Marx, D., eds., *The Multivariate Algorithmic Revolution and Beyond*, 162–195.

Chen, J.; Chor, B.; Fellows, M.; Huang, X.; Juedes, D.; Kanj, I. A.; and Xia, G. 2005. Tight lower bounds for certain parameterized NP-hard problems. *Information and Computation* 201(2):216–231.

Donini, F. M., and Massacci, F. 2000. Exptime tableaux for $\mathcal{ALC}$. *Artificial Intelligence* 124(1):87–138.

Donini, F. M.; Lenzerini, M.; Nardi, D.; Hollunder, B.; Nutt, W.; and Spaccamela, A. M. 1992. The complexity of existential quantification in concept languages. 53(2-3):309–327.

Donini, F. M.; Lenzerini, M.; Nardi, D.; and Nutt, W. 1997. The complexity of concept languages. *Information and Computation* 134(1):1–58.

Downey, R. G., and Fellows, M. R. 1995. Fixed-parameter tractability and completeness. II. On completeness for $W[1]$. *Theoretical Computer Science* 141(1-2):109–131.

Downey, R. G., and Fellows, M. R. 2013. *Fundamentals of Parameterized Complexity*. Springer Verlag.

Downey, R. 2012. A basic parameterized complexity primer. In Bodlaender, H. L.; Downey, R.; Fomin, F. V.; and Marx, D., eds., *The Multivariate Algorithmic Revolution and Beyond*, 91–128.

Flum, J., and Grohe, M. 2003. Describing parameterized complexity classes. *Information and Computation* 187(2):291–319.

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

de Haan, R., and Szeider, S. 2014a. Fixed-parameter tractable reductions to SAT. In Egly, U., and Sinz, C., eds., *Proceedings of the 17th International Symposium on the Theory and Applications of Satisfiability Testing (SAT 2014)*, 85–102.

de Haan, R., and Szeider, S. 2014b. The parameterized complexity of reasoning problems beyond NP. In Baral, C.; De Giacomo, G.; and Eiter, T., eds., *Proceedings of the 14th International Conference on the Principles of Knowledge Representation and Reasoning (KR 2014)*. AAAI Press.

de Haan, R., and Szeider, S. 2016. Parameterized complexity results for symbolic model checking of temporal logics. In *Proceedings of the 15th International Conference on the Principles of Knowledge Representation and Reasoning (KR 2016)*, 453–462. AAAI Press.

de Haan, R., and Szeider, S. 2017. Parameterized complexity classes beyond para-NP. *J. of Computer and System Sciences* 87:16–57.

de Haan, R. 2016. *Parameterized Complexity in the Polynomial Hierarchy*. Ph.D. Dissertation, Technische Universität Wien.

Kikot, S.; Kontchakov, R.; and Zakharyaschev, M. 2011. On (in)tractability of OBDA with OWL 2 QL. In *Proceedings of the 24th International Workshop on Description Logics (DL 2011)*.

Krisnadhi, A., and Lutz, C. 2007. Data complexity in the $\mathcal{EL}$ family of description logics. In *Proceedings of the 14th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR 2007)*, 333–347.

Krisnadhi, A. A. 2007. Data complexity of instance checking in the $\mathcal{EL}$ family of description logics. Master's thesis, Technische Universität Dresden, Germany.

Motik, B. 2012. Parameterized complexity and fixed-parameter tractability of description logic reasoning. In Bjørner, N., and Voronkov, A., eds., *Proceedings of the 18th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR 2012)*, 13–14.

Papadimitriou, C. H., and Yannakakis, M. 1999. On the complexity of database queries. *J. of Computer and System Sciences* 58(3):407–427.

Schild, K. 1991. A correspondence theory for terminological logics: Preliminary report. In Mylopoulos, J., and Reiter, R., eds., *Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI 1991)*, 466–471.

Schmidt-Schauß, M., and Smolka, G. 1991. Attributive concept descriptions with complements. *Artificial Intelligence* 48(1):1–26.

Simančík, F.; Motik, B.; and Horrocks, I. 2014. Consequence-based and fixed-parameter tractable reasoning in description logics. *Artificial Intelligence* 209:29–77.

Simančík, F.; Motik, B.; and Krötzsch, M. 2011. Fixed parameter tractable reasoning in DLs via decomposition. In *Proceedings of the 24th International Workshop on Description Logics (DL 2011)*, 400–410.