Efficient Evaluation and Static Analysis for Well-Designed Pattern Trees with Projection

PABLO BARCELÓ, Dept. of Comp. Science, Univ. of Chile and IMFD Chile, Chile
MARKUS KRÖLL, REINHARD PICHLER, and SEBASTIAN SKRITEK, TU Wien, Austria

Conjunctive queries (CQs) fail to provide an answer when the pattern described by the query does not exactly match the data. CQs might thus be too restrictive as a querying mechanism when data is semistructured or incomplete. The semantic web therefore provides a formalism—known as *(projected) well-designed pattern trees* (pWDPTs)—that tackles this problem: pWDPTs allow us to formulate queries that match parts of the query over the data if available, but do not ignore answers of the remaining query otherwise. Here we abstract away the specifics of semantic web applications and study pWDPTs over arbitrary relational schemas. Since the language of pWDPTs subsumes CQs, their evaluation problem is intractable. We identify structural properties of pWDPTs that lead to (fixed-parameter) tractability of various variants of the evaluation problem. We also show that checking if a pWDPT is equivalent to one in our tractable class is in 2EXPTIME. As a corollary, we obtain fixed-parameter tractability of evaluation for pWDPTs with such good behavior. Our techniques also allow us to develop a theory of approximations for pWDPTs.

CCS Concepts: • **Information systems → Query languages**;

Additional Key Words and Phrases: RDF, well-designed pattern trees, SPARQL, query answering, treewidth, hyper-treewidth, containment, subsumption, approximations

ACM Reference format:
Pablo Barceló, Markus Kröll, Reinhard Pichler, and Sebastian Skritek. 2018. Efficient Evaluation and Static Analysis for Well-Designed Pattern Trees with Projection. *ACM Trans. Database Syst.* 43, 2, Article 8 (August 2018), 44 pages.
https://doi.org/10.1145/3233983

1 INTRODUCTION

Conjunctive queries (CQs) constitute the core of the query languages for relational databases and also the most intensively studied querying mechanism in the database theory community. But CQs suffer from a serious drawback when dealing with information that is semistructured or incomplete, or when users do not have a good understanding of the schema that underlies the data: CQs fail to provide an answer when the pattern described by the query does not exactly match the data.

The work of P. Barceló is funded by the Millenium Institute for Foundational Research on Data and by Fondecyt under Grant 1170109. R. Pichler and S. Skritek were supported by the Vienna Science and Technology Fund (WWTF) through project ICT12-015 and by the Austrian Science Fund (FWF): P25207-N23 and P30930-N35. M. Kröll was funded by FWF project W1255-N23.

Authors' addresses: P. Barceló (corresponding author), Dept. of Computer Science & IMFD Chile, Beauchef 851, Santiago, 837 0456, Chile; email: pbarcelo@dcc.uchile.cl; M. Kroll, R. Pichler, and S. Skritek, DBAI, TU Wien, Favoritenstrasse 9-11 / 3rd floor A-1040 Vienna, Austria; emails: {kroell, pichler, skritek}@dbai.tuwien.ac.at.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2018 ACM 0362-5915/2018/08-ART8 $15.00

https://doi.org/10.1145/3233983

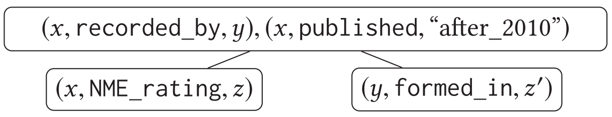
$(x, \texttt{recorded\_by}, y), (x, \texttt{published}, \text{“after\_2010”})$
$(x, \texttt{NME\_rating}, z)$
$(y, \texttt{formed\_in}, z')$

Fig. 1. WDPT representing query (1) from Example 1.1.

The semantic web therefore provides formalisms to overcome this problem [39]. We concentrate on the simplest such formalism, which corresponds to the {AND,OPTIONAL}-fragment of SPARQL—the standard query language for the semantic web data model, the *Resource Description Framework* (RDF). This fragment allows one not only to specify patterns by taking conjunctions of atoms (using the AND operator)—in the same way as CQs do—but also to match patterns over the data, if available, without failing to give an answer otherwise. This is precisely the role of the OPTIONAL operator, which allows for optional matching and essentially corresponds to the left outer join in relational algebra.

Example 1.1. Consider the following {AND,OPTIONAL}-SPARQL query that is posed over a database storing information about bands and records:

$$\{\,\{?\mathtt{x}\ \mathtt{recorded\_by}\ ?\mathtt{y}\ .\ ?\mathtt{x}\ \mathtt{published}\ \text{“after\_2010”}\}\ \mathtt{OPTIONAL}\ \{?\mathtt{x}\ \mathtt{NME\_rating}\ ?\mathtt{z}\}\,\}$$
$$\mathtt{OPTIONAL}\ \{?\mathtt{y}\ \mathtt{formed\_in}\ ?\mathtt{z}'\}. \qquad (1)$$

This query retrieves all values (b, r) such that r is a record of band b that was published after 2010 (as specified by the pattern $\{?\mathtt{x}\ \mathtt{recorded\_by}\ ?y\ .\ ?x\ \mathtt{published}\ \text{“after\_2010”}\}$), and, whenever possible, (one or both of) the following pieces of data: the rating s of record r as declared by the NME magazine and the year s' in which band b was formed. In other words, in addition to (b, r) we also retrieve s and/or s' if they can be found in the database. This is specified by the patterns $\{?\mathtt{x}\ \mathtt{NME\_rating}\ ?\mathtt{z}\}$ and $\{?\mathtt{y}\ \mathtt{formed\_in}\ ?\mathtt{z}'\}$ following the respective OPTIONAL operators.

Pérez et al. [36] noticed that the unconstrained interaction of AND and OPTIONAL in SPARQL may lead to undesired behavior. This motivated the definition of a better behaved restriction of the language, known as *well-designed* {AND,OPTIONAL}-SPARQL; e.g., the query in Example 1.1 is well-designed. Among other things, queries in this fragment have a lower complexity of evaluation [36] and lend themselves to optimization techniques [31, 38]. Moreover, they allow for a natural tree representation, known as *well-designed pattern trees*, or WDPTs [31]. Overall, several aspects of well-designed queries have been studied in recent years [3, 44], including an extension of this fragment that subsumes many of the queries encountered in practice [29].

Intuitively, a WDPT p consists of a tree T rooted in a distinguished node r and a function that labels each node of T with a set of triple patterns. The condition of being well-designed requires that occurrences of the same variable in different nodes of T are connected. Each node of a WDPT p represents a conjunction of triples, while the nesting of optional matching is represented by the tree structure of p. For instance, the query in Example 1.1 can be represented as the WDPT in Figure 1. Observe that, unlike in Example 1.1, in the following we use the algebraic-style notation from Pérez et al. [36] rather than the official SPARQL syntax.

In intuitive terms, the semantics of a WDPT p is as follows. With each subtree T' of T rooted in r, we associate a CQ $q_{T'}$ defined by the conjunction of all atoms in the nodes of T', where we identify triples with atoms over a single ternary relation symbol. Then the evaluation of p over database $\mathcal{D}$ consists of all “maximal” answers to the CQs of the form $q_{T'}$. That is, we take the union of all answers to the CQs of the form $q_{T'}$, for T' a subtree of T rooted in r, and then remove all answers for which this union contains an “extension.” We revisit Example 1.1 to illustrate these ideas.

Example 1.2. Consider an RDF database $\mathcal{D}$ consisting of the following triples:

$$(\text{“Our\_love”}, \texttt{recorded\_by}, \text{“Caribou”}),\ (\text{“Our\_love”}, \texttt{published}, \text{“after\_2010”}),$$
$$(\text{“Swim”}, \texttt{recorded\_by}, \text{“Caribou”}),\ (\text{“Swim”}, \texttt{published}, \text{“after\_2010”}),$$
$$(\text{“Swim”}, \texttt{NME\_rating}, \text{“8”}).$$

The evaluation of the WDPT in Figure 1 over $\mathcal{D}$, and, thus, of the query in Equation (1), consists of partial mappings h_1 and h_2 defined on variables x, y, z, z' such that h_1 is defined on x and y in such a way that $h_1(x)$ = “Our_love” and $h_1(y)$ = “Caribou,” and h_2 is defined on x, y and z with $h_2(x)$ = “Swim”, $h_2(y)$ = “Caribou,” and $h_2(z)$ = “8.” Observe that h_3 defined on x and y as $h_3(x)$ = “Swim” and $h_3(y)$ = “Caribou” is not in the result: while it is an answer to the CQ defined from the set of atoms at the root node, it is not maximal in the sense that h_2 (an answer to the CQ for the subtree consisting of the root node and the left child node) is an extension of h_3.

The expressive power of WDPTs is limited due to the absence of projection, a feature that CQs enjoy. Consequently, WDPTs are often enhanced with projection as a way to increase their expressiveness and obtain a proper extension of the class of CQs over RDF [31, 38]. In this article, we concentrate on this extended class of WDPTs, which we call pWDPTs.

Example 1.3. For the WDPT from Example 1.1, suppose that we want to project out the variable x. This results in restricting the mappings h_1 and h_2 from Example 1.2 to h_1' and h_2' in such a way that: h_1' is only defined on y with $h_1'(y)$ = “Caribou,” and h_2' is defined on y and z with $h_2'(y)$ = “Caribou” and $h_2'(z)$ = “8.” Observe that both mappings are part of the result, despite the fact that h_2' is an extension of h_1' (but they originate from different mappings on the existential variables). This is a crucial difference between WDPTs and pWDPTs.

pWDPTs are of interest not only for semantic web applications, but also for applications that need to handle semistructured or incomplete data (e.g., graph databases and noSQL stores such as MongoDB). This motivates our study of pWDPTs over arbitrary relational schemas, abstracting away from the specifics of the semantic web data model RDF, which only allows for triples in the nodes of pWDPTs.

Despite the importance of pWDPTs, very little is known about some fundamental problems related to them. In particular, no in-depth study has been carried out regarding efficient evaluation of these queries, a problem that permeates the literature on CQs and extensions in terms of both classical complexity [10, 22, 23, 27, 43] and parameterized complexity [26, 28, 35]. Likewise, restrictions on pWDPTs to reduce the complexity of basic static query analysis tasks, such as testing containment [38], are largely unexplored. Other topics strongly related to the identification of tractable fragments of query evaluation correspond to the reformulation and approximation in such tractable classes. There we ask if some query is equivalent to, or can at least be “approximated” by, a query from a tractable class. These questions have received considerable attention in case of CQs and conjunctive regular path queries over graph databases [6, 10, 16]. So far, nothing is known in the case of pWDPTs.

Goals and Contributions. The main goal of this article is to initiate a systematic study of tractable fragments of pWDPTs for query evaluation and to apply these fragments to fundamental questions in the areas of static query analysis and approximation. We now explain the different aspects of the problem we study and our main results related to them.

Efficient Evaluation of pWDPTs. Evaluation of pWDPTs is defined in terms of CQ evaluation, which is an intractable problem in general. Therefore, our goal of identifying tractable classes of pWDPTs naturally calls for a restriction of the classes of CQ patterns allowed in them. In particular, there has been a flurry of activity around the topic of determining which classes of CQs admit efficient evaluation that could be reused in our scenario [22, 23, 43]. We concentrate here on two of the most fundamental classes: those of bounded *treewidth* [14, 16] and *hypertreewidth* [23]. We denote by $\mathsf{TW}(k)$ and $\mathsf{HW}(k)$ the CQs of treewidth and hypertreewidth at most k, for $k \geq 1$. Queries in these classes even lie in the parallelizable class LogCFL [22, 23].

The restriction to tractable classes of CQ evaluation has already been successfully applied in the context of (projection-free) WDPTs. In particular, a very mild condition known as *local tractability* leads to efficient evaluation [31]. This condition enforces each node in the WDPT to contain a set of relational atoms from one of our tractable classes of CQs, namely $\mathsf{TW}(k)$ or $\mathsf{HW}(k)$. Nevertheless, this condition does not lead to tractability for the more expressive pWDPTs with projection that we study here [31]. Then the question remains: When is the evaluation of pWDPTs tractable or, more precisely, which natural conditions can be added to local tractability to achieve tractable pWDPT evaluation? We show that one only needs to add a mild condition—called *bounded interface*—that limits by a constant the number of variables that each node can share with other nodes in a pWDPT. More specifically, our results imply that:

(1) pWDPTs that enjoy local tractability and bounded interface can be evaluated efficiently, more precisely, in LogCFL.

Since our classes properly contain CQs of bounded treewidth and hypertreewidth, we obtain relevant extensions of these well-known tractable classes of CQs. Interestingly, conditions very similar to bounded interface have been applied to obtain good bounds for the containment problem of Datalog into CQs [9] and fine-grained tractability conditions in constraint satisfaction [12].

Due to the nature of pWDPTs, two other evaluation problems—called the *partial* and *maximal* evaluation problems—are of importance [2, 5, 36]. The first refers to checking whether a mapping h is a *partial answer* to the evaluation $p(\mathcal{D})$ of a pWDPT p over a database $\mathcal{D}$; i.e., whether there is a mapping $h' \in p(\mathcal{D})$ that "extends" h. The second problem asks if h is maximal among all answers in $p(\mathcal{D})$. (As discussed in Example 1.3, in the presence of projection a partial mapping and also a proper extension thereof may be solutions of a pWDPT.) We shall identify tractable fragments also for these problems by introducing the notion of *global tractability*. This notion restricts every CQ $q_{T'}$ represented by a subtree T' of the tree structure T of the pWDPT p to belong to $\mathsf{TW}(k)$ or $\mathsf{HW}(k)$. More specifically, our results imply that:

(2) Global tractability ensures tractability of the partial and maximal evaluation problems.

This is interesting since we also show that global tractability does not suffice to obtain tractability for the exact evaluation problem. In fact, global tractability is strictly weaker than local tractability plus bounded interface.

Parameterized Complexity. Parameterized complexity theory [17] has developed into a well established approach to dealing with intractability. The ideal result of a parameterized complexity analysis is fixed-parameter tractability. This means that the problem can be solved in time $f(k) \cdot n^{O(1)}$, where n is the size of the input and $f(k)$ is a function depending solely on the parameter k. In other words, the exponential explosion can thus be confined to the parameter. The class of problems with this behavior is denoted by FPT. If such a behavior cannot be achieved, we speak of fixed-parameter intractability (a notion which depends on the generally agreed assumption that FPT$\neq$ W[1]—for a definition of W[1], see Section 2). This means that we can at best get an upper bound $O(n^{f(k)})$ on the time complexity. In other words, the size of the parameter occurs in the exponent of the input size n. As customary in the database theory literature, we take the size of the query as parameter. The parameterized complexity approach allows us to obtain a more fine-grained picture of the intractable cases by exploring the boundary between FPT-evaluation and fixed-parameter intractable evaluation of pWDPTs.

We introduce a new kind of restriction on pWDPTs, which we will call *semi-bounded interface* (this notion relaxes the bounded interface restriction introduced above). Together with global tractability, it will allow us to delineate the border between fixed-parameter tractability and fixed-parameter intractability. In particular, we show that:

(3) To obtain fixed-parameter tractability for the evaluation of pWDPTs we only require global tractability and semi-bounded interface.

This is in contrast to the classical complexity-theoretical setting, where the semi-bounded interface does not allow us to define further tractable classes of pWDPT evaluation (in particular, even under such restrictions, pWDPT evaluation is NP-complete).

Containment and Subsumption. Containment is a crucial static analysis task that amounts to checking whether the answers to a query q_1 are necessarily contained in the answers to another query q_2 (often written as $q_1 \subseteq q_2$). The containment problem for CQs is NP-complete [13]. In contrast, it becomes undecidable for pWDPTs [38], and remains so even for our restriction to local tractability and bounded interface. The same holds for the equivalence problem (i.e., checking whether the answers to q_1 necessarily coincide with the answers to q_2).

It is known that pWDPT containment may display some unintuitive behavior, which motivated the introduction of a meaningful variant of it known as *subsumption* [4]. This is the problem of checking whether every answer of a pWDPT p_1 over any database $\mathcal{D}$ can be "extended" to an answer of pWDPT p_2 over $\mathcal{D}$ (we denote this by $p_1 \sqsubseteq p_2$). Then, the corresponding notion of equivalence is *subsumption-equivalence*, where we ask if both directions $p_1 \sqsubseteq p_2$ and $p_2 \sqsubseteq p_1$ hold. In sharp contrast to containment, subsumption for pWDPTs is known to be decidable and complete for the class Π_2^P [31]. Subsumption-equivalence can be shown to have the same behavior. We investigate in this context to what extent the restriction to tractable classes of pWDPT evaluation alleviates the complexity of checking subsumption or subsumption-equivalence. In particular, we establish the following:

(4) The restriction to tractable classes of query evaluation reduces the complexity of subsumption and subsumption-equivalence to coNP.

Reformulation in Tractable Classes of pWDPTs. We introduce syntactic restrictions on pWDPTs that lead to tractability of evaluation. A general method for finding larger classes of queries with good evaluation properties is to explore the *semantic space* defined by these syntactical restrictions; this space is defined by all queries that are equivalent to ones in the syntactically defined class (see, e.g., References [10, 16]). In this context, the two most important questions are:

(a) Is it decidable to check whether a query is equivalent to one in the desired, syntactically defined class? That is, can the query be reformulated as one in such classes?
(b) Can the evaluation problem be solved more efficiently for queries equivalent to one in such a "well-behaved" class?

Positive answers to these questions have been provided in the context of CQs [16]. In particular, regarding question (a), it is known that verifying if a CQ is equivalent to one in TW(k) is in NP. For question (b), it can be proved that the evaluation problem for those CQs that are equivalent to one in TW(k) is in PTIME [16]. Here we investigate these questions for pWDPTs.

Some care is required in fixing the appropriate setting for this investigation. For instance, since classical equivalence is undecidable for pWDPTs [38], we have to content ourselves with the relaxed notion of equivalence based on subsumption introduced above. But subsumption-equivalence only preserves partial and maximal answers. We shall therefore focus on the partial and maximal evaluation problems and choose global tractability as the corresponding tractability criterion of pWDPTs. Our main finding will be a positive answer to both questions (a) and (b) in this setting. In particular, our main result in this regard establishes the following:

(5) The problem of checking whether a pWDPT is subsumption-equivalent to one from a globally tractable class (under mild restrictions) is decidable in double-exponential time (more specifically, in $\text{NExpTime}^{\text{NP}}$).
(6) The partial and maximal evaluation problems for the pWDPTs that are subsumption-equivalent to one from a globally tractable class are fixed-parameter tractable (again taking the size of the pWDPTas parameter).

Approximations of pWDPTs. When a CQ q is not equivalent to one in a desired class $\mathcal{Q}$, it might be convenient to compute a $\mathcal{Q}$-*approximation* of q. This is a CQ $q' \in \mathcal{Q}$ that is maximal (with respect to query containment $\subseteq$) among all queries in $\mathcal{Q}$ that are contained in q. Intuitively, q' is sound with respect to q (since $q' \subseteq q$) and provides the best under-approximation of q among all queries in $\mathcal{Q}$ that are sound for q. Approximations of CQs are well understood [6]; e.g., $\mathsf{HW}(k)$-approximations of CQs always exist and can be computed in exponential time. These results allow us to explain the role of approximations. In general, the evaluation of a CQ q on a database $\mathcal{D}$ is of the order $|\mathcal{D}|^{O(|q|)}$, which is very expensive for a large dataset $\mathcal{D}$ even if q is small. On the other hand, the previous properties imply that computing and running an approximation of a CQ q on a database $\mathcal{D}$ takes time $|\mathcal{D}| \cdot 2^{pol(|q|)}$, for some polynomial *pol*. This is much faster than $|\mathcal{D}|^{O(|q|)}$ on large databases. Thus, if the quality of the approximation is good, we may prefer to run this faster query instead of q.

Our techniques allow us to develop a thorough theory of approximations for pWDPTs. Again, we define approximations via subsumption instead of containment. Furthermore, we look for approximations by pWDPTs of the globally tractable classes. Our main result in this context is the following:

(7) Approximations in globally tractable classes of pWDPTs (under mild restrictions) always exist, can be computed in double-exponential time, and have at most single-exponential size.

Unions of pWDPTs. We finally study unions of pWDPTs (UpWDPTs) as a natural extension of pWDPTs. For the variants of query evaluation considered here, all results on pWDPTs easily carry over to UpWDPTs. In contrast, for reformulation and approximation by tractable classes of UpWDPTs, we shall reveal a huge difference between pWDPTs and UpWDPTs. By establishing a close connection between UpWDPTs and unions of CQs, we can apply the theory of approximations of CQs to pWDPTs. This will allow us to prove significantly better complexity bounds for the problems studied in the context of reformulation and approximation. In particular:

(8) Checking if a UpWDPT ρ is equivalent to a union ρ' of globally tractable pWDPTs, and checking if such a ρ' is an approximation of ρ, can be solved in single-exponential time (more precisely, in Π_3^P).

This is in stark contrast to single pWDPTs, where we obtain double-exponential upper bounds for the analogous problems.

Relevance for Practice. As shown in Reference [37], the queries in the {AND,OPTIONAL}-fragment of SPARQL form the backbone of the SPARQL queries posed in practice, and a big part of such real-world {AND,OPTIONAL}-SPARQL queries can be represented as WDPTs. As with respect to projection, close to 15% of the SPARQL queries posed in practice make use of such a feature [11]. In addition, the combination of projection with AND and OPTIONAL operators is present in other popular query languages for semistructured data; in particular, in the Cypher language used by the graph database Neo4J [34] and in the standard query language for MongoDB [33]. All these facts naturally call for a deeper understanding of when pWDPTs can be evaluated efficiently. Our work not only provides such an understanding, but also develops algorithms for efficient evaluation of pWDPTs. We believe that such algorithms are rather simple and have the potential to inspire practical evaluation methods.

The conditions we develop for tractability are also meaningful from a practical point of view. In fact, we can think of the strictest such condition that corresponds to the combination of local tractability and bounded interface. The former states that the hypertreewidth of the CQs that appear in the nodes of a pWDPT is bounded (ideally by a small value). It is known that, in practice, CQs are often of small hypertreewidth [11, 21], so one would expect a similar property to hold for the ones that label nodes in pWDPTs. This has been recently confirmed for the case of the pWDPTs that appear in SPARQL practice: the vast majority of such queries are acyclic, and most of the rest are of (hyper)treewidth two [11]. Interface values, on the other hand, are also small for real-world SPARQL queries; in fact, close to 90% of such queries have an interface value of one [11].

Interestingly, also some of our negative results about the complexity of pWDPT evaluation have a practical meaning. For instance, in practice it is common to find {AND,OPTIONAL}-SPARQL queries with only a few occurrences of the OPTIONAL operator. We show that in restricted cases (e.g., if either local tractability or bounded interface fails), evaluation for pWDPTs is intractable even if restricted to pWDPTs that consist of two nodes only. Notice that such pWDPTs correspond to {AND,OPTIONAL}-SPARQL queries with a single occurrence of the OPTIONAL operator.

The static analysis tasks we study in the article, namely, containment, reformulation, and approximation, are directly motivated by the practical problems of optimization and efficient evaluation of pWDPTs. While the algorithms we develop for such problems can hardly be considered practical, we believe that some of our complexity results open the possibility for better algorithms to be developed. For instance, we establish that the reformulation problem for UpWDPTs can be solved in single-exponential time, and also approximations can be computed in single-exponential time. This is a reasonable bound for such static analysis tasks, as the input (the pWDPT) is usually small.

Previous Work. This article extends and enhances the work in Reference [7] and parts of Reference [30]. By combining these two papers, it provides a thorough analysis of the complexity of pWDPT evaluation, both in terms of combined and parameterized complexity, and of some of its static analysis tasks, e.g., subsumption and approximation. Moreover, several results from these papers have been extended or generalized, such as Theorems 5.5, 5.6, and Corollaries 3.14, 3.18; and some results are completely new such as Theorems 3.13, 3.17, and Proposition 4.2. Moreover, many additional proof details and explanations have been added to the article.

Organization. In Section 2, we recall the basics on CQs and pWDPTs. The problem of finding tractable classes of pWDPTs is studied in Section 3. In Section 4, we look at pWDPT evaluation from a parameterized complexity-theoretic point of view. In Section 5, we consider containment and subsumption. Section 6 contains our investigation of reformulations and approximations. Finally, in Section 7, we study unions of pWDPTs. A conclusion and outlook to future work are given in Section 8. Due to space constraints, several of our proofs are provided in the online appendix.

2 PRELIMINARIES

Let $\mathbf{U}$ and $\mathbf{X}$ be disjoint countably infinite sets of constants and variables, respectively. Assume that σ is a relational schema. A *relational atom* over σ is an expression of the form $R(\vec{v})$, where R is a relation symbol in σ of arity $n > 0$ and $\vec{v}$ is an n-tuple over $\mathbf{U} \cup \mathbf{X}$. For an atom $\tau = R(\vec{v})$, let $\text{var}(\tau)$ denote the set of variables occurring in τ. We extend this notion to sets $\mathcal{R}$ of atoms $\{\tau_1, \ldots, \tau_m\}$ as $\text{var}(\mathcal{R}) = \bigcup_{i=1}^{m} \text{var}(\tau_i)$. Similarly, we use $\text{const}(\mathcal{R})$ to refer to the *constants* occurring in $\mathcal{R}$. A *database* $\mathcal{D}$ over σ is a finite set of relational atoms without variables over σ.

Conjunctive Queries. A *conjunctive query* (CQ) q over σ is a rule of the form

$$\text{Ans}(\vec{x}) \;\leftarrow\; R_1(\vec{v}_1), \ldots, R_m(\vec{v}_m), \tag{2}$$

where each $R_i(\vec{v}_i)$ $(1 \leq i \leq m)$ is a relational atom over σ and $\vec{x}$ is a tuple of distinct variables among the ones that appear in the $\vec{v}_i$'s. We often write CQsas $q(\vec{x})$ to denote that $\vec{x}$ is the tuple of the *free* variables of q.

The semantics of CQs is defined in terms of *homomorphisms*. Let $\mathcal{D}$ be a database over σ. A homomorphism from a CQ $q(\vec{x})$ of the form Equation (2) to $\mathcal{D}$ is a partial mapping $h : \mathbf{X} \to \mathbf{U}$ defined precisely on the variables mentioned in $q(\vec{x})$ such that $R_i(h(\vec{v}_i)) \in \mathcal{D}$, for each $1 \leq i \leq m$.[1] We denote by $h_{\vec{x}}$ the restriction of h to the variables in $\vec{x}$. The *evaluation* $q(\mathcal{D})$ of $q(\vec{x})$ over $\mathcal{D}$ is the set of all mappings $h_{\vec{x}}$ such that h is a homomorphism from $q(\vec{x})$ to $\mathcal{D}$.

Subsumption. For a partial mapping h, we write $\text{dom}(h)$ to denote the set of variables on which h is defined. For comparing partial mappings, the notion of *subsumption* is useful. Formally, for partial mappings $h, h' : \mathbf{X} \to \mathbf{U}$, we say that h is *subsumed* by h', denoted $h \sqsubseteq h'$, if $\text{dom}(h) \subseteq \text{dom}(h')$ and $h(x) = h'(x)$, for each $x \in \text{dom}(h)$. If $h \sqsubseteq h'$ but not $h' \sqsubseteq h$, then we write $h \sqsubset h'$.

Graphs. Throughout this article, we assume all graphs to be simple and undirected. Consider a graph $G = (V, E)$. We write $V(G)$ to refer to the set V of nodes, and $E(G)$ for the set E of edges. We often write $v \in G$ to refer to a node $v \in V(G)$, but, when necessary, may write $V(G)$ and $E(G)$ explicitly. A graph $G' = (V', E')$ is a subgraph of G if $V' \subseteq V$ and $E' \subseteq E$. For a set $V' \subseteq V$ of nodes, the *induced subgraph* of G, denoted by $G[V']$, is the graph $(V', \{\{u, v\} \in E \mid u, v \in V'\})$. A tree is a connected, acyclic graph. A subtree is a connected, acyclic subgraph. A *rooted tree* T is a tree with a distinguished node marked as its root. Given two nodes $t, \hat{t} \in V(T)$, we say that $\hat{t}$ is an ancestor of t if $\hat{t}$ lies on the path from t to the root, and $\hat{t}$ is the parent of t (t is a child of $\hat{t}$) if $\hat{t}$ is an ancestor of t and $\{t, \hat{t}\} \in E(T)$.

Pattern Trees. Intuitively, a pattern tree allows one to specify patterns over the data that should be retrieved, if available, but do not force the query to fail to give an answer otherwise. We concentrate here on the class of WDPTs extended by projection (pWDPTs), which has received considerable attention in the semantic web literature. As shown by Letelier et al. [31], pWDPTs provide an intuitive representation of well-designed {AND,OPTIONAL}-SPARQL with projection [36]. They have proved useful in analyzing query evaluation and static query analysis of SPARQL [31, 38]. Intuitively, the nodes of a pWDPT represent projection-free CQs (called "basic graph patterns" in the semantic web context) while the tree structure of a pWDPT represents the nesting of optional matching.

Definition 2.1 (The Class of pWDPTs). A *projected well-designed pattern tree* (pWDPT) over a relational schema σ is a tuple $(T, \lambda, \vec{x})$ such that the following holds:

(1) T is a tree rooted in a distinguished node r, and λ maps each node t in T to a set of relational atoms over σ;

[1] As usual, we write $h(v_1, \ldots, v_n)$ for $(h(v_1), \ldots, h(v_n))$, and define $h(u) = u$ for each constant $u \in \mathbf{U}$.

(2) for every variable y that appears in T, the set of nodes of T where y appears is connected;
(3) $\vec{x}$ is a tuple of distinct variables occurring in T, which are the *free variables* of the pWDPT.

We say that $(T, \lambda, \vec{x})$ is *projection-free* if $\vec{x}$ contains all variables occurring in T. Projection-free pWDPTs are called simply WDPTs. We sometimes write (T, λ) instead of $(T, \lambda, \vec{x})$.

Pairs (T, λ) that satisfy condition Equation (1) correspond to the extension of pattern trees studied in the semantic web context [31] to arbitrary schemas. Condition Equation (2) defines well-designedness [36].

Assume $p = (T, \lambda, \vec{x})$ is a pWDPT over σ. We define $\text{atoms}(p) = \bigcup_{t \in T} \lambda(t)$. The definitions of $\text{var}(\cdot)$ and $\text{const}(\cdot)$ extend naturally to pWDPTs as $\text{var}(p) = \text{var}(\text{atoms}(p))$, and $\text{const}(p) = \text{const}(\text{atoms}(p))$. We define $\text{dom}(p) = \text{var}(p) \cup \text{const}(p)$. In some cases, to increase readability, it is convenient to overload this notation and write $\text{atoms}(T)$ and $\text{var}(T)$ instead of $\text{atoms}(p)$ and $\text{var}(p)$, respectively. In such cases, the pWDPT p corresponding to the tree structure T will always be clear from the context. All these notions naturally extend to subtrees of p.

For a subtree T' of T, we use $p_{T'}$ to refer to the pattern tree $(T', \lambda_{|V(T')}, \text{fvar}(T'))$, where $\lambda_{|V(T')}$ is the restriction of λ to the nodes in T'. Also, we use $\text{fvar}(p_{T'})$, or $\text{fvar}(T')$ if p is clear from the context, to denote the free variables $\text{var}(p_{T'}) \cap \vec{x}$ in $p_{T'}$. We also define

$$q_{T'} \quad \text{to be the CQ} \quad \text{Ans}(\vec{y}) \leftarrow R_1(\vec{v}_1), \ldots, R_m(\vec{v}_m),$$

where $\{R_1(\vec{v}_1), \ldots, R_m(\vec{v}_m)\} = \text{atoms}(T')$, and $\vec{y} = \text{var}(T')$. If T' consists of a single node $t \in T$, then we write q_t instead of $q_{T'}$.

The *size* of a node $t \in T$ is the size of (a reasonable encoding of) the CQq_t. We write $|p|$ to denote the size of p, i.e., the sum of the sizes of nodes $t \in T$.

Minimal Subtrees. Given a pWDPT $p = (T, \lambda, \vec{x})$ and a set Y of variables, one is often looking for the *minimal subtree* T' of T rooted in r (the root node of T) such that $Y \subseteq \text{var}(p_{T'})$. It is easy to see (and was shown e.g., by Letelier et al. [31]), that if $Y \subseteq \text{var}(p)$, then T' exists and is uniquely defined: just observe that being well-designed implies that for each variable $y \in \text{var}(p)$, among all the nodes $t_i \in T$ with $y \in \text{var}(\lambda(t_i))$ there is one with minimal distance from r that is an ancestor to all the others. In summary:

Proposition 2.2 ([31]). *Let $p = (T, \lambda, \vec{x})$ be a pWDPT, r the root of T, and Y a set of variables.*

(1) The existence of a subtree T' of T containing r with $Y \subseteq \text{var}(p_{T'})$ can be decided in polynomial time.
(2) If any such subtree exists, then the minimal such subtree is uniquely defined.
(3) This minimal subtree can be computed in polynomial time.

We use $\text{minSubtree}(p, Y)$ to refer to this unique minimal subtree, in case it exists.

Semantics of pWDPTs. We define the semantics of pWDPTs by naturally extending their interpretation under RDF triples and SPARQL triple patterns [31, 38]. The intuition behind the semantics of a pWDPT $(T, \lambda, \vec{x})$ is as follows. Each subtree T' of T rooted in r (where r is the root node of T) describes a pattern, namely the CQ $q_{T'}$. A mapping h satisfies (T, λ) over a database $\mathcal{D}$, if it is "maximal" among the mappings that satisfy the patterns defined by the subtrees of T. This means that h satisfies the pattern defined by some subtree T' of T, and there is no way to "extend" h to satisfy the pattern of a bigger subtree T'' of T. The evaluation of a pWDPT $(T, \lambda, \vec{x})$ over $\mathcal{D}$ is the projection over $\vec{x}$ of the mappings h that satisfy (T, λ) over $\mathcal{D}$.

Definition 2.3 (Semantics of pWDPTs). Let $p = (T, \lambda, \vec{x})$ be a pWDPT, r the root node of T, and $\mathcal{D}$ a database over σ.

— A *homomorphism* from p to $\mathcal{D}$ is a partial mapping $h : \mathbf{X} \to \mathbf{U}$, for which there is a subtree T' of T rooted in r such that $h \in q_{T'}(\mathcal{D})$.
— The homomorphism h is *maximal* if there is no homomorphism h' from p to $\mathcal{D}$ such that $h \sqsubset h'$.

The *evaluation* of a pWDPT $p = (T, \lambda, \vec{x})$ over $\mathcal{D}$, denoted $p(\mathcal{D})$, is the set of all mappings of the form $h_{\vec{x}}$, such that h is a maximal homomorphism from p to $\mathcal{D}$.

Notice that pWDPTs properly extend CQs. In fact, assume $q(\vec{x})$ is a CQ of the form $\text{Ans}(\vec{x}) \leftarrow R_1(\vec{v}_1), \ldots, R_m(\vec{v}_m)$. Then $q(\vec{x})$ is equivalent to the pWDPT $p = (T, \lambda, \vec{x})$, where T consists of a single node r, and $\lambda(r) = \{R_1(\vec{v}_1), \ldots, R_m(\vec{v}_m)\}$. That is, $q(\mathcal{D}) = p(\mathcal{D})$, for every database $\mathcal{D}$. We typically do not distinguish between a CQ and the single-node pWDPT that represents it. On the other hand, as illustrated in Example 1.1, pWDPTs express interesting properties that cannot be expressed as CQs.

RDF Well-Designed Pattern Trees. By the nature of RDF triples and SPARQL triple patterns, pWDPTs are defined in such a context over a schema that consists of a single ternary relation. Although all lower bounds obtained in our article can be proven to hold even for RDF pWDPTs, for the sake of readability we will sometimes use atoms with higher arities.

Parameterized Complexity. Let Σ be a finite alphabet. A *parameterization* of Σ^* is a polynomial-time computable mapping $\kappa : \Sigma^* \to \mathbb{N}$. A *parameterized problem* over Σ is a pair (L, κ), where $L \subseteq \Sigma^*$ and κ is a parameterization of Σ^*. We refer to $w \in \Sigma^*$ as the instances of a problem and to the integers $\kappa(w)$ as the parameters. The following well-known problems will play an important role in our parameterized-complexity analysis:

p-Clique	
Input:	A graph G and $k \in \mathbb{N}$.
Parameter:	k
Question:	Is there a clique of size k in G?

p-Dominating Set	
Input:	A graph G and $k \in \mathbb{N}$.
Parameter:	k
Question:	Does G have a dominating set of size k?

In the literature on parameterized complexity, PARAMETER: k is a common shorthand notation for PARAMETERIZATION: $\kappa(G, k) \mapsto k$, which we follow for readability.

A parameterized problem $E = (L, \kappa)$ belongs to the class FPT of fixed-parameter tractable problems, if there exists an algorithm $\mathcal{A}$ deciding L, a polynomial $pol : \mathbb{N} \to \mathbb{N}$, and a computable function $f : \mathbb{N} \to \mathbb{N}$, such that the running time of $\mathcal{A}$ on input $w \in \Sigma^*$ is at most $f(\kappa(w)) \cdot pol(|w|)$.

Parameterized complexity theory also provides notions of intractability. Toward one of these notions, we first recall the definition of fpt-reductions. Let $E = (L, \kappa)$ and $E' = (L', \kappa')$ be parameterized problems over the alphabets Σ and Σ', respectively. An *fpt-reduction* from E to E' is a mapping $R : \Sigma^* \to (\Sigma')^*$ such that (1) for all $w \in \Sigma^*$ we have $w \in L$ iff $R(w) \in L'$, (2) there is a computable function f and a polynomial pol such that $R(w)$ can be computed in time $f(\kappa(w)) \cdot pol(|w|)$, and (3) there is a computable function $g : \mathbb{N} \to \mathbb{N}$ such that $\kappa'(R(w)) \leq g(\kappa(w))$ for all $w \in \Sigma^*$.

One notion of intractability for parameterized problems is provided by the classes W[i] (for $i \geq 1$) of the W-hierarchy. Since we are here only interested in the classes W[1] and W[2], we omit a discussion of the W-hierarchy and only recall the following facts: a parameterized problem E is in W[1] if there is an fpt-reduction of E to p-Clique. Similarly, E is in W[2] if there is an fpt-reduction of E to p-Dominating Set. Also, E is W[1]-hard or W[2]-hard if there exists an fpt-reduction of p-Clique or p-Dominating Set, respectively. It is strongly believed that problems that are hard for W[1] or W[2] are not in FPT. For details, see Flum and Grohe [20].

3 EFFICIENT EVALUATION OF PWDPTS

In this section, we study the complexity of the evaluation problem for different classes C of pWDPTs. This problem is formally defined as follows:

Eval(C)	
Input:	A pWDPT$p \in C$, a partial mapping $h\colon \mathbf{X} \to \mathbf{U}$, and a database $\mathcal{D}$.
Question:	Is $h \in p(\mathcal{D})$?

The complexity of Eval(C) has been studied for classes C of all pWDPTs and (projection-free) WDPTs.

Theorem 3.1. Eval (pWDPTs) *is* Σ_2^P*-complete* [31] *and* Eval(WDPTs) *is* coNP*-complete* [36].

The intractability of the evaluation problem leads to the natural question as to which classes of pWDPTs can be evaluated in polynomial time. Recall that evaluation of pWDPTs is defined in terms of CQ evaluation, which is also an intractable problem in general. Therefore, our goal of identifying tractable classes of pWDPTs naturally calls for a restriction of the classes of CQ patterns allowed in them. This idea has already been successfully applied for obtaining tractable classes of WDPTs [31]. With further extensions, it was later used to characterize all tractable classes of WDPTs [40]. For pWDPTs, restricting the classes of CQ patterns requires new conditions to provide tractability, which we develop in this section. But first we review some of the classes of CQs that can be evaluated efficiently.

3.1 Tractable Evaluation for CQs

The evaluation problem for a class $\mathcal{Q}$ of CQs, denoted CQ-Eval($\mathcal{Q}$), is defined analogously to the case of pWDPTs. That is, CQ-Eval($\mathcal{Q}$) is the problem of checking whether $h \in q(\mathcal{D})$, given a database $\mathcal{D}$, a CQ $q(\vec{x}) \in \mathcal{Q}$, and a mapping $h : \vec{x} \to \mathbf{U}$.

CQ-Eval($\mathcal{Q}$) is NP-complete when $\mathcal{Q}$ is the class of all CQs. Due to intensive research in the last two decades, we have by now a very good understanding of which classes of CQs admit tractable evaluation. In this work, we concentrate on two fundamental tractable classes of CQs: the class of CQs of bounded *treewidth* [14] and of bounded (generalized) *hypertreewidth* [23].

CQs of Bounded Treewidth. A tractable class of CQs can be obtained by restricting the *treewidth* of the *hypergraph* of queries [14]. A hypergraph H is a pair $(V(H), E(H))$, where $V(H)$ is a finite set of nodes and $E(H)$ is a finite set of *hyperedges*, i.e., subsets of $V(H)$. If H is clear from the context, we simply write (V, E).

A *tree decomposition* of a hypergraph $H = (V(H), E(H))$ is a pair (S, ν), where $S = (V(S), E(S))$ is a tree and $\nu : V(S) \to 2^{V(H)}$ is a mapping satisfying the following:

(1) For each $u \in V(H)$ the nodes in $\{s \in V(S) \mid u \in \nu(s)\}$ form a subtree of S, and
(2) each hyperedge of $E(H)$ is contained in at least one of the sets $\nu(s)$, for $s \in V(S)$.

The *width* of (S, ν) is $\max_{s \in V(S)} |\nu(s)| - 1$. The *treewidth* of H is the minimum width of its tree decompositions. Intuitively, the treewidth of H measures its *tree-likeness*. In particular, an undirected graph H is acyclic if and only if its treewidth is one. On the other hand, if H contains a clique of size k, then its treewidth is $(k - 1)$.

Let q be a CQ $\text{Ans}(\vec{x}) \leftarrow R_1(\vec{v}_1), \ldots, R_m(\vec{v}_m)$. Its underlying hypergraph H_q is the pair (V, E), where V is the set of variables mentioned in q and E consists precisely of the sets of variables in the atoms $R_i(\vec{v}_i)$, for $1 \leq i \leq m$. For example, for the CQ $\text{Ans}() \leftarrow R(x,y,z), R(x,v,v), E(v,z)$, the hyperedges are $\{x,y,z\}$, $\{x,v\}$, and $\{v,z\}$. The treewidth of a CQ q is the treewidth of H_q. We then define $\mathsf{TW}(k)$ as the class of CQs of treewidth at most k, for $k \geq 1$.

THEOREM 3.2 [14, 16]. CQ-EVAL($\mathsf{TW}(k)$) *can be solved in polynomial time, for each* $k \geq 1$.

CQs of Bounded Generalized Hypertreewidth. The notion of treewidth is too restrictive when the arity of the schemas is not fixed in advance. To overcome this limitation, Gottlob et al. [23] proposed syntactic restrictions of the class of CQs based on decompositions of their *hypergraphs*. The analogue of treewidth in this context is the notion of generalized hypertreewidth, which also leads to tractability of query evaluation.

A *generalized hypertree decomposition* of a hypergraph $H = (V(H), E(H))$ is a triple (S, ν, ξ), where S is a tree, ν is a map from $V(S)$ to $2^{V(H)}$, and ξ is a map from $V(S)$ to $2^{E(H)}$, such that

(1) (S, ν) is a tree decomposition of H;
(2) $\nu(s) \subseteq \bigcup \xi(s)$ holds for every $s \in V(S)$. In other words, the set $\nu(s)$ is *covered* by the edges in $\xi(s)$.

The *width* of (S, ν, ξ) is defined as $\max_{s \in V(S)} |\xi(s)|$. The *generalized hypertreewidth* of a hypergraph is the minimum width over all its generalized hypertree decompositions.

The generalized hypertreewidth of a CQq is the generalized hypertreewidth of H_q. We denote by $\mathsf{HW}(k)$ the class of all CQs with generalized hypertreewidth at most k. Notably, $\mathsf{HW}(1)$ corresponds to the well-studied class of *acyclic* CQs [43]. Moreover, bounded treewidth is subsumed by bounded generalized hypertreewidth: $\mathsf{TW}(k) \subseteq \mathsf{HW}(k+1)$, for every $k \geq 1$ [1]. In contrast, $\mathsf{HW}(1)$ (i.e., the class of acyclic CQs) is not subsumed by any of the classes $\mathsf{TW}(k)$.

Evaluation of CQs of bounded generalized hypertreewidth is not only polynomial but can be solved in the parallelizable complexity class LOGCFL, which lies in between NL and AC.^{1} Formally, this corresponds to the class of languages that can be reduced in logarithmic space to a context-free language.

THEOREM 3.3 [23]. *The problem* CQ-EVAL($\mathsf{HW}(k)$) *is complete for* LOGCFL *under logspace reductions, for every* $k \geq 1$.

3.2 Tractable Evaluation of pWDPTs

We now return to the main question of this section: When is the evaluation of pWDPTs tractable? A condition that has been shown to help with identifying relevant tractable fragments of WDPTs is local tractability [31].

Local Tractability. The idea is to restrict the CQ defined by each node in a pWDPT to belong to a certain class of CQs. Formally, let $\mathcal{Q}$ be a class of CQs.

Definition 3.4 (Locally in $\mathcal{Q}$). A pWDPT $(T, \lambda, \vec{x})$ is *locally in* $\mathcal{Q}$, if for each node $t \in T$ such that $\lambda(t) = \{R_1(\vec{v}_1), \ldots, R_m(\vec{v}_m)\}$ the Boolean CQ $\text{Ans}() \leftarrow R_1(\vec{v}_1), \ldots, R_m(\vec{v}_m)$ is in $\mathcal{Q}$. We write $\ell\text{-}\mathcal{Q}$ for the set of pWDPTs that are locally in $\mathcal{Q}$.

For tractability to carry over from classes of CQsto classes of pWDPTs, it is often necessary to consider classes of CQs that are closed under replacing all occurrences of a variable by the same constant. We call such classes of CQs*robust*. Formally, a class $\mathcal{Q}$ of CQsis *robust* if every $q \in \mathcal{Q}$ satisfies the following property: for every variable x in q and an arbitrary constant c (not necessarily from q), the query derived from q by replacing all occurrences of x by c is also in $\mathcal{Q}$. This is both a natural and very weak restriction which, to the best of our knowledge, is satisfied by all reasonable tractable classes of CQs. In particular, it is satisfied by the CQs of bounded treewidth as well as the CQs of bounded generalized hypertreewidth.

We use the term "local tractability" to denote the property that a pWDPT is locally in $\mathcal{Q}$ for some tractable, robust class $\mathcal{Q}$ of CQs. It is known that local tractability leads to tractability of evaluation for (projection-free) WDPTs.

Theorem 3.5 [31]. *Let $\mathcal{Q}$ be a robust class of CQs. If* CQ-Eval($\mathcal{Q}$) *is in* Ptime, *then* Eval(ℓ-$\mathcal{Q}$ $\cap$ WDPTs) *is also in* Ptime.

In the presence of projection local tractability does not guarantee tractability, even when $\mathcal{Q}$ is the class of acyclic CQs [31]. We strengthen this result.

Theorem 3.6. *Both* Eval(ℓ-TW(k)) *and* Eval(ℓ-HW(k)) *are* NP*-complete for every $k \geq 1$. They remain* NP*-hard even when restricted to pWDPTs consisting of two nodes.*

Proof. Membership was already shown by Letelier et al. [31]. Hardness is shown for the case $k = 1$ by reduction of Dominating Set. Let an instance of Dominating Set be given by a graph $G = (V, E)$ and integer $d \in \mathbb{N}$, and assume $|V| = n$. We construct a pWDPT $p = (T, \lambda, \vec{x})$, a database $\mathcal{D}$, and a partial mapping h as follows:

- T consists of a root node r with a single child t,
- $\lambda(r) = \{s(u_1), \ldots, s(u_d), map(x_0)\}$,
- $\lambda(t) = \{neq(u_0, u_i), nonedge(u_0, u_i) \mid 1 \leq i \leq d\} \cup \{map(x_1)\}$,
- all the u_i's, x_0, and x_1 are variables, with x_0 and x_1 being the free variables $\vec{x}$,
- $\mathcal{D} = \{s(v_i) \mid v_i \in V\} \cup \{neq(v_i, v_j) \mid v_i, v_j \in V \text{ with } v_i \neq v_j\} \cup$ $\{nonedge(v_i, v_j) \mid v_i, v_j \in V \text{ and } \{v_i, v_j\} \notin E\} \cup \{map(1)\}$, where by slight abuse of notation we identify constants in $\mathcal{D}$ with nodes in V, and
- the mapping h satisfies $h(x_0) = 1$ and is undefined on all other variables.

Clearly, p, $\mathcal{D}$, and h can be constructed in polynomial time from G. Furthermore, it is easy to verify that p belongs to ℓ-TW(1) and ℓ-HW(1).

We claim that G contains a dominating set of size at most d iff $h \in p(\mathcal{D})$. This is a consequence of the one-to-one correspondence between mappings $h' : \{u_1, \ldots, u_d\} \to \{v_1, \ldots, v_n\}$ such that $h \cup h'$ is a maximal homomorphism from p to $\mathcal{D}$ and dominating sets $DS \subseteq V$ of G. That is, for every dominating set DS there exists such a mapping h' with $DS = \{h'(u_i) \mid 1 \leq i \leq d\}$, and for every such mapping h', the set $\{h'(u_i) \mid 1 \leq i \leq d\}$ is a dominating set. To see this, consider a dominating set DS. The existence of a corresponding h' is obvious. Toward a contradiction of its maximality, assume that there is an extension h'' of $h' \cup h$ that is a homomorphism from p into $\mathcal{D}$ and consider $h''(u_0)$. Clearly, $h''(u_0) \in V$. We derive a contradiction by distinguishing two cases: (1) If $h''(u_0) \in DS$, then $neq(h''(u_0), h''(u_i)) \notin \mathcal{D}$ for some $1 \leq i \leq d$. This contradicts the assumption that h'' is a homomorphism from p (and, hence, in particular, from $\lambda(t)$) into $\mathcal{D}$. (2) If $h''(u_0) \notin DS$, then there is at least one $v_j \in DS$ such that $\{h''(u_0), v_j\} \in E$. But then $nonedge(h''(u_0), h''(u_i)) \notin \mathcal{D}$ for the u_i such that $h''(u_i) = h'(u_i) = v_j$, which is again a contradiction.

Next, consider a maximal mapping $h' \cup h$. If, for $DS = \{h'(u_i) \mid 1 \leq i \leq d\}$, there exists a node $v_j \in V \setminus DS$ that shares no edge with at least one node in DS, it can be easily checked that the extension h'' of $h' \cup h$ with $h''(u_0) = v_j$ and $h''(x_1) = 1$ contradicts the maximality of $h \cup h'$. □

This raises the question of which further restrictions are needed to achieve tractability. Here we identify a natural such restriction, called bounded interface.

Bounded Interface. Intuitively, the idea is to restrict the number of variables shared by any node in the pWDPT with all other nodes in the pWDPT. Formally, for a pWDPT $p = (T, \lambda, \vec{x})$ and a node $t \in T$, we define the *interface $\mathcal{I}_t$ of t in p* as $\mathcal{I}_t = \text{var}(\lambda(t)) \cap \bigcup_{t' \in T \setminus \{t\}} \text{var}(\lambda(t'))$.

Definition 3.7 (Bounded Interface). Let $c \geq 1$. A pWDPT $(T, \lambda, \vec{x})$ has *c-bounded interface* if $|\mathcal{I}_t| \leq c$ for each $t \in T$. We denote by $\mathsf{BI}(c)$ the set of pWDPTs of c-bounded interface.

When not referring to a particular c, we may use the term bounded interface instead.

Example 3.8. Recall the pWDPT p from Figure 1, and let r denote the root node of p, let t_1 be the "left" child of r, and t_2 the "right" child of r. Then $p \in \ell\text{-}\mathsf{TW}(1)$ and $p \in \mathsf{BI}(2)$. In fact, since each node contains exactly two variables, the treewidth of each node is trivially one. Concerning the number of shared variables, observe that x occurs in both in r and t_1, while y occurs in r and t_2. Thus, $\mathcal{I}_r = \{x, y\}$, $\mathcal{I}_{t_1} = \{x\}$, $\mathcal{I}_{t_2} = \{y\}$, and thus p has two-bounded interface.

For pWDPTs, due to the well-designedness condition, it is actually the case that $\mathcal{I}_t$ contains exactly those variables that t shares with its direct neighbors, i.e., its parent and its children. Hence, saying that a pWDPT has c-bounded interface is equivalent to saying that every node shares at most c variables with all of its neighbors.

Our main result of the section states that local tractability and bounded interface yield tractability of pWDPT evaluation. Even more, recall that the evaluation problem for the CQclasses $\mathsf{TW}(k)$ and $\mathsf{HW}(k)$, for $k \geq 1$, is not only tractable but lies in the parallelizable class LogCFL. In fact, it turns out that in the cases where the CQ evaluation problem is in LogCFL, local tractability and bounded interface are sufficient to carry over LogCFL-membership.

Theorem 3.9. *Let $c \geq 1$ be a positive integer and $\mathcal{Q}$ a robust class of* CQs.

(1) If CQ-Eval*($\mathcal{Q}$) is in* Ptime*, then* Eval*(ℓ-$\mathcal{Q} \cap \mathsf{BI}(c)$) is also in* Ptime.
(2) If CQ-Eval*($\mathcal{Q}$) is in* LogCFL*, then* Eval*(ℓ-$\mathcal{Q} \cap \mathsf{BI}(c)$) is also in* LogCFL.

Thus, Eval*(ℓ-$\mathsf{TW}(k) \cap \mathsf{BI}(c)$) and* Eval*($\ell$-$\mathsf{HW}(k) \cap \mathsf{BI}(c)$) are in* LogCFL *for each $k, c \geq 1$.*

Recall that CQs can be considered as pWDPTs consisting of the root node only. Hence, $\mathsf{TW}(k) \subseteq \ell\text{-}\mathsf{TW}(k) \cap \mathsf{BI}(c)$ and $\mathsf{HW}(k) \subseteq \ell\text{-}\mathsf{HW}(k) \cap \mathsf{BI}(c)$ hold for each $c \geq 1$. Therefore, by Theorem 3.9, ℓ-$\mathsf{TW}(k) \cap \mathsf{BI}(c)$ and ℓ-$\mathsf{HW}(k) \cap \mathsf{BI}(c)$ define relevant extensions of $\mathsf{TW}(k)$ and $\mathsf{HW}(k)$, respectively, that do not increase the complexity of evaluation. It follows from Reference [22] that both Eval(ℓ-$\mathsf{TW}(k) \cap \mathsf{BI}(c)$) and Eval($\ell$-$\mathsf{HW}(k) \cap \mathsf{BI}(c)$) are LogCFL-hard under logspace reductions.

One possibility to show (1) is by reduction to the evaluation problem for acyclic CQs. However, below, we first give a proof for (2). From this, (1) follows almost immediately (a short discussion on this is provided in Section A.1 of the online appendix). The proof of the LogCFL-membership uses the characterization of LogCFL via nondeterministic auxiliary pushdown automata, which we recall briefly. A *nondeterministic auxiliary pushdown automaton (NAuxPDA)* consists of a nondeterministic Turing machine with a single, read-only input tape, several worktapes (read/write), and an additional stack (pushdown store). For the space consumption of an NAuxPDA, only the space occupied on the worktapes matters—the size used on the stack does not count for space bounds. LogCFL coincides with the class of languages accepted by NAuxPDAs in logarithmic space and polynomial time [41, 42].

Proof sketch of Theorem 3.9. We concentrate on item (2). Let $p = (T, \lambda, \vec{x})$ be a pWDPT in ℓ-$\mathcal{Q} \cap \mathsf{BI}(c)$, $\mathcal{D}$ a database, and h a mapping. In addition, let p' be the WDPT $p' = (T, \lambda, \text{var}(T))$, i.e., p' is the same pattern tree as p but without projection. Recall that $h \in p(\mathcal{D})$ if and only if there is some extension h' of h with $\text{dom}(h') \cap \vec{x} = \text{dom}(h)$ and $h' \in p'(\mathcal{D})$. This, in turn, is the case if there exists some subtree T' of T containing the distinguished root node of T such that $h' \in q_{T'}(\mathcal{D})$ and there is no subtree T'' (containing T') and extension h'' of h' such that $h'' \in q_{T''}(\mathcal{D})$.

A straightforward, nondeterministic algorithm to check whether $h \in p(\mathcal{D})$ is thus to guess the subtree T' and the mapping h', and then to check that $h' \in q_{T'}(\mathcal{D})$ and that it satisfies the maximality condition stated above. Of course, guessing T' and h' at once is not possible using only logarithmic space. However, due to p being well-designed and having a bounded interface, it is possible to guess (and check) T' and h' "piecewise." We describe a logarithmic-space algorithm for a NAuxPDA implementing this idea. A discussion of its correctness is provided in Section A.1 of the online appendix.

The algorithm gets p, $\mathcal{D}$, and h as input. In addition, it uses the following variables:

— *curNode*, which stores the currently visited node of the pWDPT,
— a counter f storing a value from $\{0, \ldots, |\operatorname{dom}(h)|\}$, which corresponds to the number of different free variables encountered so far,
— a mapping $\gamma : \mathbf{X} \to \mathbf{U}$ with $|\operatorname{dom}(\gamma)| \leq c$ (where γ is the mapping on the interface variables of the current node), and
— a Boolean flag δ.

To fit into logarithmic space, *curNode* only stores a pointer to the current node. Also, the assignments in γ are stored by maintaining pairs of pointers: the first pointer references the variable, and the second points to the value. For readability, we will not make this explicit in the following.

The main scheme of the algorithm is a depth-first, left-to-right traversal of T. Since this is feasible even in LogSpace [15], and an implementation on a NAuxPDA is straightforward, we omit an algorithm for this. We only note that the variable δ controls the traversal of the tree: if set to true, the traversal continues to the children of the current node. Otherwise, the traversal tracks back to the parent of the current node. Since the code for the actual tree traversal is omitted, in the code and descriptions presented here, δ is only set, but never read. Nevertheless, we leave the assignments to δ in the code to indicate the next step in the traversal.

In the initialization phase of the algorithm, the value of *curNode* is set to the root node of the pattern tree, the value of the variable f, which is meant to count the number of free variables "seen" during the traversal of the tree, is set to 0, and δ is set to *true*. The steps laid out in Algorithm 1 are executed whenever a node is accessed in a top-down step. It is described in more detail below. Whenever a node is accessed in a bottom-up step, the only actions that need to be performed are to restore the mapping on the interface variables of the current node from the stack (i.e., γ is set to the value on top of the stack), and to set δ to *true* to make sure that the traversal of the tree continues. Finally, the algorithm may terminate in two ways: either during the execution of the steps presented in Algorithm 1 while returning NO, or as part of the traversal, when returning to the root node and there is no further unvisited child of the root node. In the latter case, to finalize, the algorithm compares f with $|\operatorname{dom}(h)|$. If they match, the output is YES, otherwise the output is NO. We discuss the rationale behind this after the discussion of Algorithm 1.

As mentioned above, whenever a node is visited for the first time (i.e., in a top-down step), the steps described in Algorithm 1 are executed (observe that they are also applied to the root node). First, a copy of the mapping on the interface variables of the parent is saved to the stack. Next (lines 2–4), from the "working copy" of γ, all mappings on variables not appearing in the current node are removed.

A crucial step is line 5, where a nondeterministic guess is made whether the current node is part of the subtree T', or not. Lines 6–18 deal with the case that *curNode* belongs to the subtree, while lines 19–21 deal with *curNode* not being part of the subtree.

If *curNode* is guessed to belong to the subtree, then first of all it is checked whether all free variables occurring in the current node are actually contained in the domain of h (line 7). If this is not the case, then clearly the current node cannot be part of T', and the guess was wrong. Next (lines 8–9), the counter f is increased by the number of "new" free variables in the current node, i.e., free variables in the current node that do not occur in its parent. The next step is to guess a mapping on the interface variables of the current node (lines 10–16). Of course, this mapping must agree with the mapping on the interface variables in the parent (line 11) as well as with h (lines 12–13). Thus, only the values for the variables $x \in \text{var}(\lambda(curNode)) \setminus (\text{var}(\lambda(m)) \cup \text{dom}(h))$ are guessed (lines 14–16), where m is the parent of *curNode*. Afterward, this guess is verified by checking if the partial mapping given by γ on the interface variables and by h can be extended to a mapping on all variables in $\text{var}(\lambda(curNode))$ that maps all atoms in $\lambda(curNode)$ into $\mathcal{D}$ (line 17). Thus, if the execution reaches line 18, *curNode* is part of T', and thus the computation should descend to the children of *curNode*, indicated by setting δ to *true*.

If the guess was that *curNode* does not belong to T', then it is checked that the mapping γ on the interface variables of the parent cannot be extended to the current node (line 20). Thus, if line 21 is reached, we may indeed assume that *curNode* does not belong to T', therefore descending to its children is not necessary.

Thus, once the traversal stops at the root node because there is no further child to visit, the algorithm has found some maximal mapping consistent with h that maps some subtree T' of T into $\mathcal{D}$. The only property that has not been checked up to this point is if this maximal mapping actually contains all of $\text{dom}(h)$, or only parts of it. Note that the counter f contains the number of free variables in those nodes that are guessed to belong to T'. Since the algorithm stops immediately and returns NO whenever it encounters a free variable not in $\text{dom}(h)$ (Algorithm 1, line 7), after finishing the traversal, f has only counted variables from $\text{dom}(h)$. Thus $f = |\text{dom}(h)|$ if and only if the mapping found by the algorithm covers all of $\text{dom}(h)$.

Finally, observe that the tests in lines 17 and 20 are in LogCFL by assumption and the fact that LogCFL is closed under complement. □

ALGORITHM 1: EnterNodeTopDown

```
1:  PUSH($\gamma$)                                              ▷ Push mapping on parent interface to stack
2:  for all $(x, v)$ in $\gamma$ do
3:      if $x \notin \text{var}(\lambda(curNode))$ then
4:          remove $x \mapsto v$ from $\gamma$
5:  Nondeterministically choose $in \in \{true, false\}$
6:  if $in = true$ then
7:      if $\text{fvar}(\lambda(curNode)) \nsubseteq \text{dom}(h)$ then EXIT(NO)
8:      for all $x \in \text{var}(\lambda(curNode))$ do
9:          if $x \in \text{dom}(h)$ and $x \notin \text{dom}(\gamma)$ then $f := f + 1$
10:     for all $x \in \mathcal{I}_{curNode}$ do
11:         if $x \notin \text{dom}(\gamma)$ then                        ▷ Does not occur in the parent
12:             if $x \in \text{dom}(h)$ then                         ▷ In $h$, thus predefined value
13:                 add $x \mapsto h(x)$ to $\gamma$
14:             else                                             ▷ Neither in parent nor in $h$
15:                 Nondeterministically choose $v \in \text{dom}(\mathcal{D})$
16:                 add $x \mapsto v$ to $\gamma$
17:     if $\gamma \cup h$ cannot be extended to subsume a solution to $q_{curNode}$ then EXIT(NO)
18:     $\delta := true$                                          ▷ Continue traversal to children
19: else
20:     if $\gamma \cup h$ can be extended to subsume a solution to $q_{curNode}$ then EXIT(NO)
21:     $\delta := false$                            ▷ Do not proceed to children, return to parent
```

3.3 Partial Evaluation of pWDPTs

Given the nature of pWDPTs, it is also interesting to check whether a mapping h is a *partial* answer to a pWDPT p over $\mathcal{D}$ [36], i.e., whether h can be extended to some answer h' to p over $\mathcal{D}$. This gives rise to the partial evaluation problem for a class C of pWDPTs.

Partial-Eval(C)	
Input:	A pWDPT $p \in C$, a partial mapping $h\colon \mathbf{X} \to \mathbf{U}$, and a database $\mathcal{D}$.
Question:	Is there an $h' \in p(\mathcal{D})$ such that $h \sqsubseteq h'$?

Partial evaluation is tractable for the class of (projection-free) WDPTs [36]. In contrast, if projection is allowed, then partial evaluation is intractable even under local tractability:

Proposition 3.10 [31]. *The problem* Partial-Eval(ℓ-TW(k)) *is* NP*-complete for every* $k \geq 1$.

Recall from Theorem 3.9 that the conjunction of local tractability and bounded interface leads to efficient (exact) evaluation of pWDPTs. It is easy to modify the proof of Theorem 3.9 (property (2)) to show that also Partial-Eval(ℓ-TW(k) $\cap$ BI(c)) and Partial-Eval(ℓ-HW(k) $\cap$ BI(c)) are in LogCFL. However, partial evaluation is seemingly easier than exact evaluation. Hence, the question naturally arises if tractability of partial evaluation of pWDPTs can be ensured by a weaker condition. Indeed, we give a positive answer to this question below by introducing the notion of global tractability.

Global Tractability. Intuitively, for some class $\mathcal{Q}$ of CQs, this condition states that the CQs defined by the different subtrees of a pWDPT belong to $\mathcal{Q}$.

Definition 3.11 (Globally in $\mathcal{Q}$). A pWDPT $(T, \lambda, \vec{x})$ with root node r is *globally in* $\mathcal{Q}$, if for each subtree T' of T rooted in r with atoms(T') $= \{R_1(\vec{v}_1), \ldots, R_m(\vec{v}_m)\}$ the Boolean CQ Ans() $\leftarrow R_1(\vec{v}_1), \ldots, R_m(\vec{v}_m)$ is in $\mathcal{Q}$. We write g-$\mathcal{Q}$ for the set of pWDPTs that are globally in $\mathcal{Q}$.

Similarly to local tractability, by "global tractability" we refer to the property of a pWDPT being globally in $\mathcal{Q}$ for a tractable, robust class $\mathcal{Q}$ of CQs.

The following proposition formally states that for the classes of CQs of bounded treewidth and generalized hypertreewidth, global tractability is a strictly weaker condition than the conjunction of local tractability and bounded interface.

Proposition 3.12.

(1) *For each* $k, c \geq 1$, ℓ-TW(k) $\cap$ BI(c) $\subseteq g$-TW($k+c$), *and* ℓ-HW(k) $\cap$ BI(c) $\subseteq g$-HW($k+c$).
(2) *For every* $k \geq 1$, *there is a family* C_k *of pWDPTs in* g-TW(k) (*respectively,* g-HW(k)) *such that* $C_k \nsubseteq$ BI(c), *for any* $c \geq 1$.

Proof.

(1) Consider an arbitrary pWDPT $p = (T, \lambda, \vec{x}) \in C$, for C either ℓ-TW(k) $\cap$ BI(c) or ℓ-HW(k) $\cap$ BI(c). Then, for every node $t \in T$, there exists a (tree- or generalized hypertree) decomposition D_t of width $\leq k$ for the CQ defined by $\lambda(t)$. From D_t (with $D_t = (S, \nu)$ and $D_t = (S, \nu, \xi)$, respectively) we construct a decomposition D'_t by extending $\nu(s)$ to $\nu'(s) = \nu(s) \cup \mathcal{I}_t$ for every $s \in S$. Thus, the width of the decomposition increases at most by c.

Given such a decomposition for each node $t \in T$ and some subtree T' of T, we can construct a decomposition for the complete set of atoms occurring in the nodes of T' by simply combining the decompositions D'_{t_1} and D'_{t_2} of two neighboring nodes t_1 and t_2 in T' by adding an edge between an arbitrarily chosen node s_1 in D'_{t_1} and an arbitrarily chosen node s_2 in D'_{t_2}. Since both $\nu'(s_1)$ and $\nu'(s_2)$ contain all the interface variables, the combined decomposition clearly satisfies the connectedness condition. As a result, we get a decomposition of $q_{T'}$ with width at most $k + c$, which proves the claim.

(2) Let $k \geq 1$ and $c \geq 1$. Using unary predicates un_1, un_2 and a binary predicate bin, it is easy to construct a pattern tree $p = (T, \lambda, \vec{x})$ consisting of two nodes, such that p has a global treewidth of k, but not c-bounded interface (a similar proof holds for generalized hypertreewidth). For example, let T consist of a root r and a child t, with
$-\lambda(r) = \{bin(z_i, z_j) \mid i, j \in \{1, \ldots, k\}, i \neq j\} \cup \{un_1(y_i) \mid i \in \{1, \ldots, c+1\}\}$ and
$-\lambda(t) = \{un_2(y_i) \mid i \in \{1, \ldots, c+1\}\} \cup \{un_1(x)\}$,
where x and all z_i and y_i are variables, and $\vec{x}$ contains x and an arbitrary subset of the remaining variables. □

We now formally prove that global tractability leads to tractability of the partial evaluation problem for pWDPTs.

Theorem 3.13. *Let $\mathcal{Q}$ be a robust class of* CQs *for which* CQ-Eval*($\mathcal{Q}$) is in* Ptime. *Then* Partial-Eval*(g-$\mathcal{Q}$) is also in* Ptime.

Proof. Let $p = (T, \lambda, \vec{x})$ be a pWDPT, let $\mathcal{D}$ be a database, and let h be a mapping. Also, let T' be minSubtree$(p, \text{dom}(h))$ and atoms$(T') = \{R_1(\vec{v}_1), \ldots, R_m(\vec{v}_m)\}$. Recall that T' can be computed in polynomial time (Proposition 2.2). Observe that to test whether there exists some $h' \in p(\mathcal{D})$ with $h \sqsubseteq h'$, it suffices to test whether there exists a mapping $h'' : \text{var}(T') \setminus \text{dom}(h) \rightarrow \text{dom}(\mathcal{D})$ such that $R_i([h \cup h''](\vec{v}_i)) \in \mathcal{D}$ for all $1 \leq i \leq m$. Thus, let $q(\vec{v})$ be the CQAns$(\text{dom}(h)) \leftarrow R_1(\vec{v}_1), \ldots, R_m(\vec{v}_m)$. Then $h \in q(\mathcal{D})$ if and only if h is a partial solution to p over $\mathcal{D}$. Also, checking whether $h \in q(\mathcal{D})$ can be decided in polynomial time. In fact, this is equivalent to asking whether the empty mapping is a solution to the CQq' defined as Ans$() \leftarrow R_1(h(\vec{v}_1)), \ldots, R_m(h(\vec{v}_m))$, which is in Ptime by assumption. □

Similar to the case of local tractability and bounded interface, also, in this case, membership in LogCFL carries over.

Corollary 3.14. *Let $\mathcal{Q}$ be a robust class of* CQs *for which* CQ-Eval*($\mathcal{Q}$) is in* LogCFL. *Then* Partial-Eval*(g-$\mathcal{Q}$) is also in* LogCFL. *In particular,* Partial-Eval*(g-*TW*(k))* and Partial-Eval*(g-*HW*(k))* are in LogCFL *for every $k \geq 1$.*

Proof. Clearly, the reduction described in the proof of Theorem 3.13 can be implemented by a LogSpace-transducer. Also, deciding whether the empty mapping is a solution to the CQ q' in the proof of Theorem 3.13 is in LogCFL by assumption. □

LogCFL-hardness for $\mathcal{Q} = \text{TW}(k)$ and $\mathcal{Q} = \text{HW}(k)$ follows again from the LogCFL-hardness of CQs in $\mathcal{Q}$ and the observation that the evaluation problem and the partial evaluation problem coincide for CQs.

It remains to answer the question of whether global tractability also suffices to ensure tractability of (exact) evaluation for pWDPTs. We show that this is not the case.

Proposition 3.15. *Both* Eval*(g-*TW*(k))* and Eval*(g-*HW*(k))* are NP*-complete for every $k \geq 1$. They remain* NP*-hard when restricted to pWDPTs consisting of two nodes.*

Proof. Membership can be shown by a guess-and-check algorithm that, given p, $\mathcal{D}$, and h, guesses a subtree p' of p together with a mapping h' on the existential variables in p', and then checks in polynomial time whether $h' \cup h$ is a maximal homomorphism. A description of such an algorithm is provided in Section A.2 of the online appendix. For the hardness, recall the pWDPT p defined in the proof of Theorem 3.6. It is easy to see that $p \in g\text{-TW}(1)$ and $p \in g\text{-HW}(1)$. □

3.4 Semantics Based on Maximal Mappings

The semantics of (projection-free) WDPTs is only based on *maximal* mappings, i.e., mappings that are not subsumed by any other mapping in the answer. This is no longer the case in the presence of projection as demonstrated in Example 1.3.

Recent work on query answering for SPARQL under entailment regimes has established the need for a semantics for pWDPTs that is solely based on maximal mappings [2]. Such semantics also turned out to be essential in a recent study on so-called *weakly-monotone* SPARQL queries [5]. This semantics is formalized as follows. Assume $\mathcal{D}$ is a database and p is a pWDPT over σ. The *evaluation of* p *over* $\mathcal{D}$ *under maximal mappings*, denoted $p_m(\mathcal{D})$, contains the restriction of $p(\mathcal{D})$ to those mappings $h \in p(\mathcal{D})$ that are maximal with respect to $\sqsubseteq$.

Example 3.16. Let p be the pWDPT from Figure 1 and assume the free variables to be $\vec{x} = \{y, z\}$. Consider the database $\mathcal{D}$ from Example 1.2. Then $p(\mathcal{D}) = \{h_1, h_2\}$ with $h_1(y) = h_2(y) =$ "*Caribou*" and $h_2(z) =$ "8" while $p_m(\mathcal{D}) = \{h_2\}$.

This naturally leads to the following decision problem:

MAX-EVAL(C)	
Input:	A pWDPT$p \in C$, a partial mapping $h \colon \mathbf{X} \to \mathbf{U}$, and a database $\mathcal{D}$.
Question:	Is $h \in p_m(\mathcal{D})$?

Ahmetaj et al. [2] show that MAX-EVAL(C) is intractable when C is the class of all pWDPTs (in particular, this problem is complete for DP, i.e., the class of languages defined as the intersection of a language in NP and one in coNP). Analogously to the case of partial evaluation presented in Theorem 3.13, it is sufficient to impose global tractability to obtain tractability also in this case:

THEOREM 3.17. *Let* $\mathcal{Q}$ *be a robust class of* CQs *for which* CQ-EVAL*(*$\mathcal{Q}$*) is in* PTIME. *Then* MAX-EVAL*(*g-$\mathcal{Q}$*) is also in* PTIME.

PROOF. Observe that, given a pWDPT $p = (T, \lambda, \vec{x})$, a database $\mathcal{D}$, and a mapping h, we have $h \in p_m(\mathcal{D})$ if and only if

(1) $h \sqsubseteq h'$ for some $h' \in p(\mathcal{D})$ and
(2) there does not exist a mapping h' with $h \sqsubset h'$ such that $h' \in p(\mathcal{D})$.

That is, the problem can be solved by reducing it to a polynomial number of instances of the partial evaluation problem. Indeed, for (1) we only need to test if h is a partial solution, and for (2) it suffices to test whether $\bar{h}$ is no partial solution for all mappings of the form $\bar{h} = h \cup \{x \mapsto c\}$, where $x \in \vec{x} \setminus \text{dom}(h)$ and $c \in \text{dom}(\mathcal{D})$. □

It follows immediately from the above algorithm and the fact that LOGCFL is closed under complement, that whenever CQ-EVAL($\mathcal{Q}$) is in LOGCFL, so is MAX-EVAL(g-$\mathcal{Q}$):

COROLLARY 3.18. *Let* $\mathcal{Q}$ *be a robust class of* CQs *for which* CQ-EVAL*(*$\mathcal{Q}$*) is in* LOGCFL. *Then* MAX-EVAL*(*g-$\mathcal{Q}$*) is also in* LOGCFL. *In particular,* MAX-EVAL*(*g-TW(k)*) and* MAX-EVAL*(*g-HW(k)*) are in* LOGCFL *for every* $k \geq 1$.

Analogously to PARTIAL-EVAL(C), local tractability is not sufficient to ensure tractability of MAX-EVAL(C):

PROPOSITION 3.19. *For every* $k \geq 1$ *the problems* MAX-EVAL*(*ℓ-TW(k)*) and* MAX-EVAL*(*ℓ-HW(k)*) are* DP*-complete.*

PROOF SKETCH. The DP-membership follows immediately from the general case studied by Ahmetaj et al. [2]. The DP-hardness is shown in Section A.3 of the online appendix. □

4 PARAMETERIZED COMPLEXITY OF PWDPT EVALUATION

In the previous section, we introduced some sufficient criteria for the different evaluation problems on pWDPTs to be tractable, where we considered a problem to be tractable if it can be decided in polynomial time. One main idea behind the definitions of our tractable classes was to extend tractable restrictions from CQs to pWDPTs.

However, for CQsit is also reasonable to consider less restrictive notions of tractability, in particular fixed-parameter tractability. In general, the problem p-CQ-Eval($\mathcal{Q}$), which is the problem CQ-Eval($\mathcal{Q}$) parameterized by the size of the query, is W[1]-complete for the set $\mathcal{Q}$ of all CQs [35]. In addition, the deep work of Marx [32] provided a complete understanding of which classes of CQs admit fixed-parameter tractable evaluation.

There is one relevant scenario, however, in which the notions of tractability and fixed-parameter tractability for CQ evaluation are equivalent (at least under widely held complexity assumptions); namely, when the arity of schemas is fixed in advance. In fact, a surprising result by Grohe [26] states that, under the widely accepted complexity assumption that FPT$\neq$ W[1] (cf. Reference [20]), the notion of treewidth captures, in a precise sense, the space of tractable (and also fixed-parameter tractable) CQs over schemas of bounded arity. Formally, in this scenario of the evaluation problem a class $\mathcal{Q}$ of CQs is tractable if and only if it is fixed-parameter tractable if and only if $\mathcal{Q}$ is of bounded treewidth *modulo equivalence*. The latter means that for some $k \geq 1$, every CQ in $\mathcal{Q}$ is equivalent to a CQ in $\mathsf{TW}(k)$ [26].

In contrast, by considering a slight extension of the conditions introduced in Section 3, we will show in this section that the latter equality does not hold for pWDPTs. That is, there are classes C of pWDPTs for which the evaluation problem is NP-complete, but for which the parameterized evaluation problem is fixed-parameter tractable (even over fixed arity schemas). The condition, which we will refer to as "semi-bounded interface," allows us to draw a more detailed picture of the complexity of the evaluation problem for pWDPTs.

Semi-Bounded Interface. Intuitively, the notion of c-semi-bounded interface restricts the number of variables shared between any pair of nodes in the tree. Formally, for a pWDPT $p = (T, \lambda, \vec{x})$, the interface $\mathcal{I}_{t,t'}$ between nodes $t, t' \in T$ is $\mathcal{I}_{t,t'} = \text{var}(\lambda(t)) \cap \text{var}(\lambda(t'))$.

Definition 4.1 (c-semi-bounded interface). Let $c \geq 1$. A pWDPT $(T, \lambda, \vec{x})$ has *c-semi-bounded interface* if $|\mathcal{I}_{t,t'}| \leq c$ for all pairs t, t' of nodes in T. We denote by $\mathsf{SBI}(c)$ the class of pWDPTs of c-semi-bounded interface.

Observe that because of the pattern tree being well-designed, this definition is equivalent to requesting that for all pairs of nodes $t, t' \in T$ that are connected by an edge, the number of shared variables is bounded.

In contrast to the class $\mathsf{BI}(c)$, no property analogous to Proposition 3.12 holds for $\mathsf{SBI}(c)$; that is, ℓ-$\mathsf{TW}(k) \cap \mathsf{SBI}(c)$ (or ℓ-$\mathsf{HW}(k) \cap \mathsf{SBI}(c)$, respectively) does not imply g-$\mathsf{TW}(k')$ (or g-$\mathsf{HW}(k')$, respectively) for any $k' \geq 1$. For instance, consider the family $(p_n)_{n\geq 2}$ of pWDPTs, where p_n contains a root node and $\binom{n}{2}$ children $t_{i,j}$ with $1 \leq i < j \leq n$. For a unary predicate un and a binary predicate bin, we set $\lambda(r) = \{un(x_1), \ldots, un(x_n)\}$ and $\lambda(t_{i,j}) = \{bin(x_i, x_j)\}$. Then $p_n \in \ell$-$\mathsf{TW}(1) \cap \mathsf{SBI}(2)$ and $p_n \in \ell$-$\mathsf{HW}(1) \cap \mathsf{SBI}(2)$ for every $n \geq 1$. But for every $k' \geq 1$, we can choose n so that $p_n \notin g$-$\mathsf{TW}(k')$ and $p_n \notin g$-$\mathsf{HW}(k')$. This is because $\bigcup_{1\leq i<j\leq n} \lambda(t_{i,j})$ contains a clique of size n. Thus, the only relationships between these classes arise from the fact that g-$\mathsf{TW}(k) \subseteq \ell$-$\mathsf{TW}(k)$, since, unlike generalized hypertreewidth, the treewidth of a graph is preserved by its subgraphs.

In terms of the classical complexity of the evaluation problem, introducing the class $\mathsf{SBI}(c)$ does not improve the situation with respect to the tractability of the problem.

Proposition 4.2. *Both* Eval(g-TW$(k) \cap$ SBI(c)) *and* Eval(ℓ-TW$(k) \cap$ SBI(c)) *are* NP-*complete for every $k \geq 1$ and $c \geq 2$. The same holds for* HW(k) *instead of* TW(k).

The complexity is thus exactly the same as for Eval(ℓ-TW(k)) and Eval(g-TW(k)) (and HW(k)), respectively. However, for the parameterized variant of the evaluation problem, we get a much more diverse picture. We will study the following problem:

p-Eval(C)	
Input:	A pWDPT $p \in C$, a partial mapping $h\colon \mathbf{X} \to \mathbf{U}$, and a database $\mathcal{D}$
Parameter:	$\lvert p \rvert$
Question:	Is $h \in p(\mathcal{D})$?

We will show that for all four classes ℓ-TW$(k) \cap$ BI(c), ℓ-TW$(k) \cap$ SBI(c), g-TW$(k) \cap$ SBI(c), and g-TW(k), the problem p-Eval(C) exhibits a different (parameterized) complexity. For $C = \ell$-TW$(k) \cap$ BI(c), we have shown in Section 3 that the problem is in Ptime (even in LogCFL). For $C = g$-TW$(k) \cap$ SBI(c), on the other hand, the problem is actually fixed-parameter tractable:

Theorem 4.3. *The problems* p-Eval(g-TW$(k) \cap$ SBI(c)) *and* p-Eval(g-HW$(k) \cap$ SBI(c)) *are in* FPT *for every $k, c \geq 1$.*

Proof. We discuss the main ideas of the FPT algorithm for p-Eval(g-TW$(k) \cap$ SBI(c)). The case of p-Eval(g-HW$(k) \cap$ SBI(c)) is analogous. Given some pWDPT $p = (T, \lambda, \vec{x}) \in g$-TW$(k) \cap$ SBI(c) with root r, a database $\mathcal{D}$, and a mapping $h : X \to \text{dom}(\mathcal{D})$, where $X \subseteq \vec{x}$, it is easy to test for some subtree p' of p rooted in r with $\text{fvar}(p') = \text{dom}(h)$ whether there exists an extension h' of h with $h' : \text{var}(p') \to \text{dom}(\mathcal{D})$ and $h'(\tau) \in \mathcal{D}$ for all atoms $\tau \in \text{atoms}(p')$. This is because $p \in g$-TW(k). However, in addition, we must test whether at least one of these extensions h' of h is also maximal.

One possibility to tackle this issue is to introduce one relational atom for each node t of T' (where T' is the tree structure of p') that tells us which mappings on the interface variables $\mathcal{I}_t$ can be extended to a mapping on the complete node. This is feasible in the case of bounded interface, since the size of these relations is bounded by a polynomial. But given that we now deal with c-semi-bounded pWDPTs, this is no longer possible, since the size of $\mathcal{I}_t$ is no longer bounded. However, for testing the maximality of mappings, it is sufficient to check that they cannot be extended to the nodes $t_i \in T \setminus T'$ whose parent $\hat{t}_i$ is in T'. For these nodes t_i, we thus want to have one relational atom that tells us which mappings on the interface $\mathcal{I}_{\hat{t}_i, t_i}$ do *not* extend to t_i.

Unfortunately, it turns out that just introducing atoms $R_i(\mathcal{I}_{\hat{t}_i, t_i})$ is not possible, since this may increase the treewidth of the resulting CQ too much. Instead, we need to work with potentially smaller atoms. Intuitively, we make sure that a mapping cannot be extended to some node t_i by choosing a *connected component* (to be made precise below) of atoms in t_i to which the mapping cannot be extended. Moreover, the number of connected components is bounded by the size of the query. Hence, iterating over all combinations of these components is feasible in FPT. Thus, the idea is instead of having one atom $R_i(\mathcal{I}_{\hat{t}_i, t_i})$ for each node t_i, to have an atom $R_i(\vec{v}_i)$, where $\vec{v}_i$ contains all variables (arranged in an arbitrary order) of some connected component of $\lambda(t_i)$.

The tuples $\vec{v}_i$, which play a crucial role in the FPT algorithm, are defined as follows. Let t be a non-root node in a pWDPT $(T, \lambda, \vec{x})$ and $\hat{t}$ the parent of t. Let us denote by G_t the *Gaifman graph* of the variables in $\lambda(t) \setminus \mathcal{I}_{\hat{t}, t}$; i.e., G_t is the undirected graph that has as nodes the variables in $\lambda(t) \setminus \mathcal{I}_{\hat{t}, t}$ and as edges the pairs of variables that appear together in some atom of $\lambda(t)$. A *critical subset of* $\mathcal{I}_{\hat{t}, t}$ is a set $S \subseteq \mathcal{I}_{\hat{t}, t}$ such that

(1) either there is an atom in $\lambda(t)$ whose variables are precisely S, or
(2) there is a connected component G of G_t such that

$$S = \{y \in \mathcal{I}_{\hat{t},t} \mid \text{there is an atom in } \lambda(t) \text{ that contains both } y \text{ and a variable in } G\}.$$

We write $\mathcal{CS}(t)$ for the set of critical subsets of $\mathcal{I}_{\hat{t},t}$. The atoms $R_i(\vec{v}_i)$ mentioned above will then be obtained by taking $\vec{v}_i$ as the tuple of variables (arranged in arbitrary order) of some critical subset $S_i \in \mathcal{CS}(t_i)$.

The importance of the critical subsets stems from the fact that given some mapping $\hat{h}$ on an interface $\mathcal{I}_{\hat{t},t}$, the critical subsets suffice to decide whether $\hat{h}$ can be extended to t or not. Toward this goal, we introduce the following notions. As before, let t be a non-root node in a pWDPT $(T, \lambda, \vec{x})$ and $\hat{t}$ the parent of t. For a set $Y \subseteq \mathcal{I}_{\hat{t},t}$ of variables, the set *stop*(Y) of *non-extendable mappings* is the set of all mappings $\hat{h} : Y \to \text{dom}(\mathcal{D})$ such that there exists no mapping $\hat{h}' : \text{var}(\lambda(t)) \to \text{dom}(\mathcal{D})$ satisfying $\hat{h} \sqsubseteq \hat{h}'$ and $R(\hat{h}'(\vec{w})) \in \mathcal{D}$ for every $R(\vec{w}) \in \lambda(t)$. Intuitively, for a subset Y of the interface variables $\mathcal{I}_{\hat{t},t}$, the set *stop*(Y) contains all mappings on Y that cannot be extended so as to map all atoms in $\lambda(t)$ into $\mathcal{D}$. For the proof of Theorem 4.3, we are interested in those sets *stop*(S) where $S \in \mathcal{CS}(t)$ is a critical subset of $\mathcal{I}_{\hat{t},t}$.

We now have all ingredients necessary to describe the FPT algorithm. Assume we are given a pWDPT $p = (T, \lambda, \vec{x})$, a database $\mathcal{D}$, and a mapping h, and we want to decide whether $h \in p(\mathcal{D})$. The main structure of the procedure is sketched in Algorithm 2. Before describing the algorithm, we first have to define what $B(T')$ (line 2) means. For a pWDPT $(T, \lambda, \vec{x})$ and a subtree T' of T, the set $B(T')$ contains all nodes $t \in T \setminus T'$ such that the parent of t is in T'. Intuitively, $B(T')$ contains the "children" of T' (or nodes immediately "below" T', thus $B(T')$).

The idea of the algorithm is to compute a potentially exponential number of CQsq (exponential only in the size of the pWDPT p) and for each of the CQsa database $\mathcal{D}'$ such that $h \in p(\mathcal{D})$ if and only if $q(\mathcal{D}') \neq \emptyset$ for at least one of these queries. Recall that to get $h \in p(\mathcal{D})$, there must be some maximal subtree T' of T containing the root node of T and an extension h' of h such that $h' \in q_{T'}(\mathcal{D})$. Thus the algorithm iterates over all potential such subtrees T' of T (line 1). For each of these subtrees it would now be easy to check whether such an $h' \in q_{T'}(\mathcal{D})$ exists: compute $q_{T'}$, replace in $q_{T'}$ all variables $x \in \text{dom}(h)$ by $h(x)$, let the remaining variables be existential variables, and check if the resulting Boolean CQ evaluates to true over $\mathcal{D}$. However, just checking $h' \in q_{T'}(\mathcal{D})$ is not enough, since also the maximality must be checked. This is done by exploiting the critical subsets of $B(T')$.

As a result, GETCQ (T', css) (line 4) returns the CQq defined as follows. Let

$$\text{atoms}(T') = \{R_1(\vec{w}_1), \ldots, R_m(\vec{w}_m)\},$$

and assume that $css = \{S_1, \ldots, S_n\}$, i.e., css contains one critical subsets S_i for every node $t_i \in B(T')$. Then q is of the form $\text{Ans}() \leftarrow R_1(h(\vec{w}_1)), \ldots, R_m(h(\vec{w}_m)), R'_1(h(\vec{s}_1)), \ldots, R'_n(h(\vec{s}_n))$, where $R'_1, \ldots, R'_n$ are new relation symbols not occurring anywhere in p or $\mathcal{D}$, and each $\vec{s}_i$ is some tuple containing exactly the variables from S_i in some arbitrary order (for $1 \leq i \leq n$).

The matching database $\mathcal{D}'$ returned by GETDB $(css, \mathcal{D})$ (line 5) is defined as follows:

$$\mathcal{D}' \ := \ \mathcal{D} \cup \{R'_i(\hat{h}'(\vec{s}_i)) \mid \hat{h}' \in stop(S_i), S_i \in css\},$$

where each $\vec{s}_i$ is again the tuple containing the variables from S_i. Hence, for each critical subset S_i, we put exactly those tuples into $\mathcal{D}'$ that cannot be extended to mappings on $\lambda(t_i)$.

The idea behind these definitions is as follows. Assume that $q(\mathcal{D}') \neq \emptyset$ for some q and $\mathcal{D}'$. Then each $h' \in q(\mathcal{D}')$ maps all atoms in atoms(T') into $\mathcal{D}$ and there exists no extension h'' of h' that, for any $t_i \in B(T')$, maps $\lambda(t_i)$ into $\mathcal{D}$. The latter property is due to the fact that h' is consistent with some mapping from *stop*(S) for at least one critical subset $S \in \mathcal{CS}(t_i)$ for each such node t_i.

Concerning its runtime, note that the algorithm potentially tests an exponential number of pairs of CQsq and databases $\mathcal{D}'$. However, the number is bounded by $O(2^{|p|})$ (recall that $|p|$ is the parameter). Finally, it can be shown that the CQq is of bounded treewidth or generalized hypertreewidth, given that $p \in g\text{-TW}(k) \cap \text{SBI}(c)$ or $p \in g\text{-HW}(k) \cap \text{SBI}(c)$ for some $k, c \geq 1$, respectively. A thorough discussion of this property can be found in Section B.2 of the appendix. □

ALGORITHM 2: EvalFPT(p, $\mathcal{D}$, h)

```
1: for all subtrees $T'$ of $T$ rooted in $r$ such that $\text{fvar}(T') = \text{dom}(h)$ do    ▷ $r$: root node of $T$
2:     Let $B(T')$ be of the form $\{t_1, \ldots, t_n\}$
3:     for all $css \in \{\{S_1, \ldots, S_n\} \mid S_i \in \mathcal{CS}(t_i)$ for $1 \leq i \leq n\}$ do
4:         $q$ = getCQ($T'$, $css$)
5:         $\mathcal{D}'$ = getDB($css$, $\mathcal{D}$)
6:         if $q(\mathcal{D}') \neq \emptyset$ then exit(YES)
7: exit(NO)
```

Theorem 4.3 thus gives the discrepancy to the behavior of CQsmentioned above: While the problem Eval(g-TW(k) ∩ SBI(c)) is NP-complete for $k \geq 1$ and $c \geq 2$, and thus intractable, the parameterized variant is in FPT and thus feasible. In Propositions 4.4 and 4.5 below, we show that the case of local tractability plus semi-bounded interface and the case of global tractability display yet another parameterized complexity behavior.

Proposition 4.4. *The problems* p-Eval(ℓ-TW(k) ∩ SBI(c)) *and* p-Eval(ℓ-HW(k) ∩ SBI(c)) *are* W[1]*-complete for every $k \geq 1$ and $c \geq 2$.*

Proof. Membership is shown by reduction to the W[1]-complete problem Pos-Eval, which is the evaluation problem for FO-queries, restricted as follows: given a FO-query built from relational atoms using $\exists, \wedge, \vee$, a database, and a mapping, is the mapping a solution to the query[2]; with the size of the FO-query as parameter [35]. Let an instance of p-Eval(ℓ-TW(k) ∩ SBI(c)) or p-Eval(ℓ-HW(k) ∩ SBI(c)) be given by a pWDPT $p = (T, \lambda, \vec{x})$ with root node r, a database instance $\mathcal{D}$, and a partial mapping h from (a subset of) the variables in $\vec{x}$ to $\text{dom}(\mathcal{D})$. It is convenient to introduce some notation. Let $\mathcal{T}'$ be the set of all subtrees p' of p containing r such that $\text{fvar}(p') = \text{dom}(h)$ (i.e., the set of possible "witnesses" for $h \in p(\mathcal{D})$). Also, like in the proof of Theorem 4.3, for a subtree $p' = (T', \lambda, \vec{x}')$ of p, let $B(T')$ contain exactly the nodes $t \in T \setminus T'$ such that the parent of t is in T'. Finally, for every node t in $B(T')$, let $\hat{t}$ denote the parent of t, let $R_{p',t}$ be a fresh relation symbol not occurring in p of arity $|\mathcal{I}_{\hat{t},t}|$, and let $\vec{w}_t$ contain the variables from $\mathcal{I}_{\hat{t},t}$ in some order.

The idea of the reduction is now similar to the one in the proof of Theorem 4.3: create a positive query which is a big disjunction over all possible CQs $q_{T'}$ corresponding to subtrees $p' = (T', \lambda, \vec{x}')$ in $\mathcal{T}'$. Thus, for each such p', let

$$At_{p'} = \text{atoms}(T') \cup \{R_{p',t}(\vec{w}_t) \mid t \in B(T')\}.$$

The purpose of the atoms $R_{p',t}(\vec{w}_t)$ is to guarantee that only maximal answers are picked. They will only map to such values on the interface variables $\mathcal{I}_{\hat{t},t}$ that cannot be extended to mappings on t. Observe that unlike in the proof of Theorem 4.3, using a single atom for each node in $B(T')$ with all interface variables is feasible. This is because the resulting query is not required to have bounded treewidth.

[2]Like for CQs, for convenience we consider the answers to the FO-queries as mappings rather than tuples.

For defining the values added to $\mathcal{D}$, recall from the proof of Theorem 4.3 that $stop(\mathcal{I}_{\hat{t},t})$ contains those mappings on $\mathcal{I}_{\hat{t},t}$ that cannot be extended so as to map all atoms in $\lambda(t)$ into $\mathcal{D}$. We define a positive query Q and a database instance $\mathcal{D}'$, while h will be corresponding mapping:

$$\mathcal{D}' = \mathcal{D} \cup \bigcup_{\substack{p' \in \mathcal{T}' \\ p' = (T', \lambda, \vec{x}')}} \{R_{p',t}(h'(\vec{w}_t)) \mid h' \in stop(\mathcal{I}_{\hat{t},t}), t \in B(T')\},$$

$$Q = \bigvee_{p' \in \mathcal{T}'} \exists(\text{var}(p') \setminus \text{dom}(h)) \bigwedge_{\tau \in At_{p'}} \tau.$$

Clearly, the above reduction is an fpt-reduction: the size of Q is in $O(2^{|p|})$, and the size of $\mathcal{D}'$ is polynomial in the size of $\mathcal{D}$ since $|\mathcal{I}_{\hat{t},t}| < c$ for every $t \in T$. Finally, it follows immediately that $h \in p(\mathcal{D})$ if and only if $h \in Q(\mathcal{D}')$, which proves membership.

Hardness is shown by an fpt-reduction of p-Clique. Let $G = (V, E)$ be a graph and $d \in \mathbb{N}$. We define a pWDPT $p = (T, \lambda, \vec{x})$, a mapping h, and a database $\mathcal{D}$ as follows:

— T consists of a root r with children $t_{i,j}$ for $1 \leq i < j \leq d$, with:

$$\begin{aligned} \lambda(r) &= \{node(y_1), \ldots, node(y_d), aux(x_0)\}, \\ \lambda(t_{i,j}) &= \{edge(y_i, y_j), aux(x_{i,j}) \mid 1 \leq i < j \leq d\}, \\ \vec{x} &= \{x_0\} \cup \{x_{i,j} \mid 1 \leq i < j \leq d\}; \end{aligned}$$

— $\mathcal{D} = \{edge(u, v) \mid \{u, v\} \in E\} \cup \{node(v) \mid v \in V\} \cup \{aux(1)\}$, and
— $h = \{x_0 \mapsto 1\} \cup \{x_{i,j} \mapsto 1 \mid 1 \leq i < j \leq d\}$.

Notice that $p \in \ell\text{-TW}(1) \cap \text{SBI}(2)$ and that the reduction is an fpt-reduction. It is also easy to see that G has a clique of size d iff $h \in p(\mathcal{D})$. In fact, observe that for every mapping h' that maps $\lambda(r)$ into $\mathcal{D}$, the values $h'(y_1), \ldots, h'(y_d)$ encode a selection of d (not necessarily distinct) nodes from V. Also, for each child $t_{i,j}$, the mapping h' can only be extended to map all of $\lambda(t_{i,j})$ into $\mathcal{D}$ iff there is an edge between $h'(y_i)$ and $h'(y_j)$ (note that G does not contain self loops). Thus the selection of nodes encoded by $h'(y_1), \ldots, h'(y_d)$ is a clique iff all children of r can be mapped into $\mathcal{D}$. □

Proposition 4.5. *Both* p-Eval(g-TW(k)) *and* p-Eval(g-HW(k)) *are* W[2]*-hard for every* $k \geq 1$.

Proof. Recall the reduction for the lower bound in Theorem 3.6. This also describes an fpt-reduction of p-Dominating Set, and that the resulting pWDPT is in g-TW(1) and g-HW(1). □

5 CONTAINMENT AND SUBSUMPTION

Query containment and query equivalence are among the most fundamental problems in static query analysis of any query language, i.e., given two queries q_1 and q_2, one wants to test if, for every database $\mathcal{D}$, the condition $q_1(\mathcal{D}) \subseteq q_2(\mathcal{D})$ or $q_1(\mathcal{D}) = q_2(\mathcal{D})$, respectively, holds. If this is the case, we write $q_1 \subseteq q_2$ or $q_1 \equiv q_2$, respectively. For CQs, these problems are NP-complete in general [13] and LogCFL-complete for classes TW(k) and HW(k) [14, 23].

Pichler and Skritek [38] carried out a detailed study of containment and equivalence of RDF pWDPTs and showed that, in sharp contrast to CQs, both problems are undecidable. Now the question remains if the restriction to tractable fragments of pWDPT evaluation can help. An inspection of the undecidability proofs shows that this is not the case.

Theorem 5.1 (Implicit in Pichler and Skritek [38]). *The containment and equivalence problems of pWDPTs are undecidable. The undecidability holds even if both pWDPTs are from* ℓ-TW(k) $\cap$ BI(c) *for arbitrary* $k \geq 1$ *and appropriately chosen constant* c.

Arenas and Pérez [4] observed that containment of pWDPTs may display an unintuitive behavior. Consequently, they proposed *subsumption* as a variant of containment: a pWDPT p_1 is subsumed by p_2 (written as $p_1 \sqsubseteq p_2$) if, for every database $\mathcal{D}$, every answer $h \in p_1(\mathcal{D})$ is subsumed by an answer $h' \in p_2(\mathcal{D})$.

Analogously, we define *subsumption-equivalence* (denoted as $p_1 \equiv_s p_2$) if both $p_1 \sqsubseteq p_2$ and $p_2 \sqsubseteq p_1$ hold. We thus study the following problems, for C_1, C_2 classes of pWDPTs:

Subs (C_1, C_2)	
Input:	pWDPTs $p_1 \in C_1$ and $p_2 \in C_2$.
Question:	Does $p_1 \sqsubseteq p_2$ hold?

Subs-Equiv (C_1, C_2)	
Input :	pWDPTs $p_1 \in C_1$ and $p_2 \in C_2$.
Question :	Does $p_1 \equiv_s p_2$ hold?

Letelier et al. [31] established Π_2^P-completeness of Subs(C_1, C_2) where both C_1 and C_2 are the class of all pWDPTs; Π_2^P-hardness holds even for the class of (projection-free) WDPTs.

We now recall the useful characterization of subsumption between two pWDPTs the Π_2^P-membership is based on, since we will use it in several proofs throughout the remainder of this article. To do this, we first define a suitable notion of homomorphism between pWDPTs. For two pWDPTs p_1 and p_2, a homomorphism $\mu : p_1 \to p_2$ is a mapping from $\mathrm{var}(p_1)$ to $\mathrm{dom}(p_2)$ such that $R(\mu(\vec{v})) \in \mathrm{atoms}(p_2)$ for all $R(\vec{v}) \in \mathrm{atoms}(p_1)$. In the next proposition, for the sake of readability, for a pWDPT $p_i = (T_i, \lambda_i, \vec{x}_i)$ and a subtree T_i' of T_i we write p_i' instead of $(p_i)_{T_i'}$ for the restriction of p_i to the nodes in T_i'.

Proposition 5.2 [31]. *Let $p_1 = (T_1, \lambda_1, \vec{x}_1)$ and $p_2 = (T_2, \lambda_2, \vec{x}_2)$ be two pWDPTs and r_1, r_2 their roots. Then $p_1 \sqsubseteq p_2$ if and only if, for every subtree T_1' of T_1 rooted in r_1, there exists a subtree T_2' of T_2 rooted in r_2 such that*

(1) $\mathrm{fvar}(p_1') \subseteq \mathrm{fvar}(p_2')$*, and*
(2) there exists a homomorphism $\mu : p_2' \to p_1'$ with $\mu(x) = x$ for all $x \in \mathrm{fvar}(p_1')$.

In contrast to Subs (C_1, C_2), the problem Subs-Equiv (C_1, C_2) has not been studied so far. However, Ahmetaj et al. [2] considered a closely related problem based on the maximal mappings semantics from Section 3.4—the so-called MaxEquiv (C_1, C_2)-problem: given two pWDPTs $p \in C_1$, $p' \in C_2$, does $p_m(\mathcal{D}) = p'_m(\mathcal{D})$ hold for every database $\mathcal{D}$? In other words, we check if two pWDPTs p and p' have the same *maximal* solutions over every $\mathcal{D}$. If this is the case, we write $p \equiv_{\max} p'$. This problem was shown to be Π_2^P-complete [2]. An inspection of the proof shows that Π_2^P-hardness holds even if one of the classes C_i is restricted to ℓ-TW$(2) \cap$ BI(2). Below, we show that Subs-Equiv (C_1, C_2) and MaxEquiv (C_1, C_2) are equivalent problems. In this way, we immediately inherit Π_2^P-completeness results also for Subs-Equiv (C_1, C_2):

Proposition 5.3. *For any pWDPTs p and p', $p \equiv_s p'$ if and only if $p \equiv_{\max} p'$.*

Proof. Suppose $p \equiv_s p'$. Let $\mathcal{D}$ be an arbitrary database and let $h \in p_m(\mathcal{D}) \subseteq p(\mathcal{D})$. Then there exists an $h' \in p'(\mathcal{D})$ such that $h \sqsubseteq h'$. Without loss of generality, we may assume that $h' \in p'_m(\mathcal{D})$. By $p \equiv_s p'$, there is h'' in $p(\mathcal{D})$ such that $h' \sqsubseteq h''$. Hence, $h \sqsubseteq h''$. By the maximality of h, we conclude that $h = h' = h''$. Thus, $h \in p'_m(\mathcal{D})$. The case for $h \in p'_m(\mathcal{D})$ is symmetric.

Suppose $p \equiv_{\max} p'$. Let $\mathcal{D}$ be an arbitrary database and let $h \in p(\mathcal{D})$. Clearly there exists $h' \in p_m(\mathcal{D})$ with $h \sqsubseteq h'$. By $p \equiv_{\max} p'$ we also have $h' \in p'(\mathcal{D})$. Hence, for each $h \in p(\mathcal{D})$ there exists an $h' \in p'(\mathcal{D})$ with $h \sqsubseteq h'$, i.e., $p \sqsubseteq p'$. The proof that also $p' \sqsubseteq p$ holds is symmetric. □

We then immediately obtain the following from our previous remarks on the complexity of the MaxEquiv (C_1, C_2)-problem:

Corollary 5.4. *The problem* Subs-Equiv(C_1, C_2) *is* Π_2^P*-complete, even if one of the classes* C_i *is restricted to* ℓ-HW$(k) \cap$ BI(c)(*or to* ℓ-TW$(k) \cap$ BI(c)) *with* $k = c = 2$*, while the other is the class of all pWDPTs.*

Now the natural questions are if the restriction of C_1 and/or C_2 to tractable classes of pWDPT evaluation leads to a lower complexity of Subs(C_1, C_2), and if the restriction of both C_1 and C_2 leads to a lower complexity of Subs-Equiv(C_1, C_2). We provide answers below in Theorems 5.5 and 5.6.

Theorem 5.5.

(1) Subs (C_1, C_2) *is in* coNP *whenever* C_1 *is the class of arbitrary pWDPTs and* C_2 *is any class such that* Partial-Eval(C_2) *is in* Ptime*. In particular, this is the case when* $C_2 = g$-TW(k) *or* $C_2 = g$-HW(k) *for any* $k \geq 1$.

(2) Subs (C_1, C_2) *is* coNP*-hard when* $C_1 = \ell$-HW$(k) \cap$ BI(c) *or* $C_1 = \ell$-TW$(k) \cap$ BI(c)*, for any* $k \geq 1$ *and* $c \geq 0$*, and* C_2 *is the class of* CQs *of bounded treewidth or bounded hypertreewidth.*

Proof. The first item follows from the observation that the co-problem, i.e., the problem of deciding, given two pWDPTs $p_1 = (T_1, \lambda_1, \vec{x}_1) \in C_1$ and $p_2 = (T_2, \lambda_2, \vec{x}_2) \in C_2$, whether $p_1 \not\sqsubseteq p_2$, is in NP. To see this, we first recall the standard notion of a frozen database: for a set S of atoms, the frozen database fr(S) is the database obtained from S by simultaneously replacing each variable x with a fresh constant c_x. Then $p_1 \not\sqsubseteq p_2$ can be tested as follows: First, guess a suitable subtree T' of T_1, and, second, check that the mapping $h_{T'}$, with $h_{T'}(x) = \text{fr}(x)$ for all $x \in \text{dom}(h_{T'}) = \text{fvar}(T')$, is *not* a partial answer to p_2 over fr(atoms(T')). The correctness of this approach is an immediate consequence of the following relationship: for two pWDPTs p_1, p_2 with $p_1 = (T, \lambda, \vec{x})$, subsumption $p_1 \sqsubseteq p_2$ holds if and only if, for every subtree T' of T containing the root of T, the mapping $h_{T'}$ is a partial answer to p_2 over fr(atoms(T')). (This relationship follows immediately from Proposition 5.2; a slight variant thereof was used by Ahmetaj et al. [2].) To see that this test is actually in NP, observe that the witness guessed in the first step is of polynomial size, while the check in the second step is in Ptime because of the assumption that Partial-Eval(C_2) in Ptime.

The second item is shown by reduction of the validity problem for propositional formulas in 3-DNF. Let an arbitrary instance of the problem be given by a formula $\phi = \bigvee_{i=1}^{N}(l_{i,1} \wedge l_{i,2} \wedge l_{i,3})$, where each $l_{i,j}$ $(1 \leq i \leq N, 1 \leq j \leq 3)$ is a literal over the variables $X = \{x_1, \ldots, x_n\}$. For each variable $x_i \in X$, let γ_i denote the number of times x_i occurs in ϕ. We define two pWDPTs $p_1 = (T_1, \lambda_1, \vec{z})$ and $p_2 = (T_2, \lambda_2, \vec{z})$ such that

— T_1 consists of a root r with n children t_i, for $1 \leq i \leq n$,
— T_2 consists of a single node r',
— $\lambda_1(r) = \{v_i(0,0,1) \mid 1 \leq i \leq n\} \cup \{and(0,0,0,0), and(0,0,1,0), and(0,1,0,0), and(1,0,0,0), and(0,1,1,0), and(1,0,1,0), and(1,1,0,0), and(1,1,1,1)\} \cup \{or(0,0,0), or(0,1,1), or(1,0,1), or(1,1,1)\} \cup \{true(1)\} \cup \{map(u)\}$,
— $\lambda_1(t_i) = \{v_i(z_{i,j}, 1, 0) \mid 1 \leq j \leq \gamma_i\}$ for $1 \leq i \leq n$,
— $\lambda_2(r') = \{map(u)\} \cup \{v_i(z_{i,j}, x_{i,j}, \bar{x}_{i,j}) \mid 1 \leq i \leq n, 1 \leq j \leq \gamma_i\} \cup \{and(l^*_{i,1}, l^*_{i,2}, l^*_{i,3}, y_i) \mid 1 \leq i \leq N\} \cup \{or(y_1, y_2, y'_2)\} \cup \{or(y'_{i-1}, y_i, y'_i) \mid 3 \leq i \leq N\} \cup \{true(y'_N)\}$,
— all terms except 0 and 1 are variables and for $1 \leq i \leq N$, $1 \leq j \leq 3$ we have $l^*_{i,j} = x_{\alpha,\beta}$ if $l_{i,j} = x_\alpha$ is the β^{th} occurrence of x_α in ϕ, and $l^*_{i,j} = \bar{x}_{\alpha,\beta}$ if $l_{i,j} = \neg x_\alpha$ is the β^{th} occurrence of x_α in ϕ, and
— the free variables are $\vec{z} = \{u, z_{1,1}, \ldots, z_{1,\gamma_1}, \ldots, z_{n,1}, \ldots, z_{n,\gamma_n}\}$.

It is now easy to see that in both pattern trees, all interfaces are empty. Observe that while both p_1 and p_2 are clearly in ℓ-HW$(1) \cap$ BI(0), they are not in ℓ-TW$(1) \cap$ BI(0). This is, however, due to the presentation using atoms of arity bigger than 2 for the sake of readability. By standard reification techniques, one can obtain an RDF pWDPT in ℓ-TW$(1) \cap$ BI(0). Even more, p_2 consists of a single node and thus represents a CQ. We further observe that the proof could be adapted to work even if we disallowed constants, i.e., if also 0 and 1 were variables. However, the size of the interface of p_1 would then be $c = 2$.

In the following, we only provide a rough sketch of the correctness of the reduction. The full proof is given in Section C.1 of the online appendix. Intuitively, each subtree T_1' of T_1 represents a truth assignment I over X: for $1 \leq i \leq n$, we have $I(x_i) = true$ iff $t_i \in T_1'$. By Proposition 5.2, $p_1 \sqsubseteq p_2$ holds if and only if, for each subtree T_1' of T_1 containing r, there exists a homomorphism μ from $\lambda_2(r')$ to atoms(T_1') that is the identity on fvar(T_1'). This is the case if and only if I satisfies ϕ: observe that for all $1 \leq i \leq n$ we have that $\mu(z_{i,j}) = z_{i,j}$ (and, therefore $\mu(x_{i,j}) = 1$) for all $1 \leq j \leq \gamma_i$ if $t_i \in T_1'$ and $\mu(z_{i,j}) = 0$ (and, therefore $\mu(x_{i,j}) = 0$) otherwise. In this way, the bindings of the variables $x_{i,j}$ under μ encode exactly the intended truth assignment of the propositional variable x_i. Next, observe that the atoms *and*() in $\lambda_2(r')$ encode for each disjunct (represented by the first three arguments of the *and*()-atom) whether it evaluates to *true* or *false* under I (represented by the last argument). Thus $\mu(y_i) = 1$ for $1 \leq i \leq N$ only gives a valid homomorphism if the i^{th} disjunct evaluates to *true* under I. Finally, the *or*() atoms check if at least one disjunct evaluates to *true* by encoding the Boolean "or" (the first two arguments are the input, the last one the result), i.e., setting $\mu(y_N') = 1$ is only allowed if this is the case. However, $\mu(y_N') = 1$ is the only possibility to map *true*(y_N') into T_1, since $\lambda_1(r)$ only contains *true*(1). Thus, the required homomorphism μ exists if and only if at least one disjunct of ϕ evaluates to *true* under I. □

Next, we consider Subs-Equiv in more detail and give a similar complexity classification as in Theorem 5.5 (see Section C.1 of the online appendix for proof details).

Theorem 5.6.

(1) Subs-Equiv(C_1, C_2) *is in* coNP *whenever* C_1 *and* C_2 *are classes such that* Partial-Eval(C_1) *and* Partial-Eval(C_2) *is in* Ptime. *In particular, this is the case when* $C_1, C_2 = g$-TW(k) *or* $C_1, C_2 = g$-HW(k) *for any* $k \geq 1$.

(2) Subs (C_1, C_2) *is* coNP-*hard if* $C_1, C_2 = \ell$-HW$(k) \cap$ BI(c), *or if* $C_1, C_2 = \ell$-TW$(k) \cap$ BI(c), *for any* $k, c \geq 2$.

Theorems 5.5 and 5.6, together with the Π_2^P-completeness results of Letelier et al. [31] and Corollary 5.4, leave a small gap: What if both C_1 and C_2 are locally tractable classes? We close this gap below. The proof is in Section C.1 of the online appendix.

Proposition 5.7. *The problems* Subs(C_1, C_2) *and* Subs-Equiv(C_1, C_2) *remain* Π_2^P*-complete even if both* C_1 *and* C_2 *are restricted to* ℓ-HW$(k) \cap$ SBI(c) *or to* ℓ-TW$(k) \cap$ SBI(c), *for* $k \geq 2$ *and* $c \geq 3$.

6 REFORMULATIONS AND APPROXIMATIONS IN TRACTABLE CLASSES OF PWDPTS

In Sections 3 and 4, we developed conditions that ensure tractability for several variants of the pWDPT evaluation problem. In this section, we study the *semantic space* defined by these conditions; i.e., the space of pWDPTs that are equivalent to a pWDPT in a class defined via (hyper)tree decompositions, respectively.

First we have to fix the right notion of *equivalence*. By Theorem 5.1, strict equivalence ("≡") is undecidable even for the most restricted fragments of pWDPTs considered here. Hence, we have to be contented with a relaxed notion of equivalence—*subsumption-equivalence* ($\equiv_s$) introduced in Section 5.

But then we also have to choose the appropriate variant of pWDPT evaluation: subsumption-equivalence preserves *partial* and *maximal* solutions. Hence, we shall focus on the PARTIAL-EVAL(C) and MAX-EVAL(C) problems here. (It should be noted that the problems MAX-EVAL(C), PARTIAL-EVAL(C), and EVAL(C) coincide for CQs, i.e., pWDPTs consisting of the root node only.)

Finally, we determine the right syntactical restriction on pWDPTs to ensure tractability of these problems. By Corollary 3.14 and Theorem 3.17, the restriction to g-$\mathsf{TW}(k)$ or g-$\mathsf{HW}(k)$ for constant k is sufficient.

At this point, a discussion on two properties of generalized hypertreewidth is in order. The first one is the complexity of the recognition problem, i.e., for fixed integer k, given a hypergraph H, decide whether the generalized hypertreewidth of H is at most k. This problem is NP-complete for every $k \geq 2$ [19, 24]. As a result, in the literature, instead of *generalized* hypertreewidth, usually a slightly restricted notion of *hypertreewidth* is used. Similar to the generalized hypertreewidth, it is defined as the minimal width of any *hypertree decompositions* of a hypergraph. The definition of a hypertree decomposition in turn is almost identical to that of generalized hypertree decompositions, only that beside properties (1) and (2), it has to satisfy a third condition:

(3) $\bigcup_{e \in \nu(s)} e \cap \bigcup_{v \in S_s} \xi(v) \subseteq \xi(s)$ holds for every $s \in S$, where S_s refers to the subtree of S rooted at s.

The recognition problem for hypertreewidth is solvable in polynomial time [23]. Concerning the relationship between hypertreewidth $htw(H)$ and generalized hypertreewidth $ghtw(H)$ of an arbitrary hypergraph H, the inequalities $ghtw(H) \leq htw(H) \leq 3 \cdot ghtw(H) + 1$ hold [1].

The second property to discuss is a significant difference between treewidth and both generalized hypertreewidth and hypertreewidth. While $\mathsf{TW}(k)$ is closed under taking *arbitrary* subqueries, $\mathsf{HW}(k)$ is not. However, it will turn out convenient to choose our fragment of CQs in such a way that it enjoys this property. Instead of the class $\mathsf{HW}(k)$, in the following we will therefore consider the class $\mathsf{HW}'(k)$ consisting of all CQsq such that the *hypertreewidth* of each subquery q' of q (including q) is at most k. This restricted notion of hypertreewidth is known as β-hypertreewidth [25], in analogy with β-acyclicity studied by Fagin [18]. While the recognition problem for β-acyclicity is tractable [18], the complexity of the recognition problem for β-hypertreewidth is not known. It is obviously in coNP; but it is not known whether the problem is tractable or not. Thus, for most problems studied below, the results obtained for $\mathsf{TW}(k)$ and for $\mathsf{HW}'(k)$ will be the same. However, there will also be some results (in particular upper bounds) where an additional NP-oracle is needed in case of the class g-$\mathsf{HW}'(k)$. Note that if we dropped condition (3) from the definition of β-hypertreewidth, then we would even need a Σ_2^P-oracle instead of the NP-oracle. At any rate, to not complicate the presentation unnecessarily, in the following we concentrate on the classes g-$\mathsf{TW}(k)$ and g-$\mathsf{HW}'(k)$. The semantic space defined by these classes is defined below.

Definition 6.1 (Classes $\mathcal{M}(C)$). Let C be a class of pWDPTs and $k \geq 1$. We denote by $\mathcal{M}(C)$ the class of pWDPTs p for which there is a pWDPT $p' \in C$ such that $p \equiv_s p'$.

We show that, for $C = g$-$\mathsf{TW}(k)$ and $C = g$-$\mathsf{HW}'(k)$, these classes are decidable. We then apply this result to show that the partial and maximal evaluation problems for pWDPTs in $\mathcal{M}(g$-$\mathsf{TW}(k))$ and $\mathcal{M}(g$-$\mathsf{HW}'(k))$ are fixed-parameter tractable (when taking the size of the pWDPT as the parameter). This is an improvement with respect to the corresponding evaluation problems for arbitrary pWDPTs and even for CQs. For the latter, no fixed-parameter tractable algorithm is believed to exist. Finally, we study the notion of g-$\mathsf{TW}(k)$- and g-$\mathsf{HW}'(k)$-approximation for pWDPTs.

6.1 Decidability of $\mathcal{M}(g\text{-TW}(k))$ and $\mathcal{M}(g\text{-HW}'(k))$ Modulo Equivalence

We start by stating the decidability of membership in $\mathcal{M}(g\text{-TW}(k))$ and $\mathcal{M}(g\text{-HW}'(k))$:

Theorem 6.2. *Let $k \geq 1$. There is an* NExpTime$^{\text{NP}}$ *algorithm that, given a pWDPT p, decides whether p is in $\mathcal{M}(g\text{-HW}'(k))$, and, if this is the case, constructs a pWDPT $p' \in g\text{-HW}'(k)$ such that $p \equiv_s p'$ and the size of p' is at most exponential in the size of p. The NP-oracle is not needed for $g\text{-TW}(k)$ instead of $g\text{-HW}'(k)$.*

The correctness of this algorithm, which we will present below, follows from the next lemma that provides an exponential bound on the size of candidate pWDPTs that we have to check for equivalence.

Lemma 6.3. *Let $k \geq 1$. Let p and p_1 be pWDPTs such that $p_1 \sqsubseteq p$ and $p_1 \in g\text{-TW}(k)$ or $p_1 \in g\text{-HW}'(k)$. Then there is $p_2 \in g\text{-TW}(k)$ or $p_2 \in g\text{-HW}'(k)$, respectively, such that*

(1) $p_1 \sqsubseteq p_2 \sqsubseteq p$,
(2) the number of nodes of p_2 is at most twice the number of free variables of p, and
(3) the number of atoms of p_2 is at most exponential in the size of p.

Proof. Let $p = (T, \lambda, \vec{x})$ and $p_1 = (T_1, \lambda_1, \vec{x}_1)$ be pWDPTs as above. We construct $p_2 = (T_2, \lambda_2, \vec{x}_1)$ with the desired properties by transforming and reducing p_1 in four steps (for a node t, in the following we will always use $\hat{t}$ to denote its parent).

(a) *Remove "dead leaves."* Let $N \subseteq V(T_1)$ be the set of nodes in T_1 that introduce at least one free variable, i.e., $N = \{t \in T_1 \mid \text{fvar}(\lambda_1(t)) \setminus \text{var}(\lambda_1(\hat{t})) \neq \emptyset\}$. Transform T_1 into T_1' by deleting all nodes that are not on a path from the root to some node in N, and let λ_1' denote the restriction of λ_1 to the nodes in T_1'.

(b) *Collapse branches.* While T_1' contains some node t with a single child t' and satisfying $\text{fvar}(\lambda_1'(t)) \setminus \text{fvar}(\lambda_1'(\hat{t})) = \emptyset$, merge t with t', i.e., set $\lambda_1'(t) = \lambda_1'(t) \cup \lambda_1'(t')$, make all children t_i of t' to children of t, and remove t' from T_1'. Refer to the pattern tree resulting from exhaustive application of this transformation as $p'' = (T_1'', \lambda_1'', \vec{x}_1)$.

(c) *Remove "dead atoms."* Let r_1'' and r be the root nodes of T_1'' and T, respectively. We have $p'' \sqsubseteq p$. Thus, by Proposition 5.2 for each subtree S'' of T_1'' containing r_1'', there is a subtree S of T containing r with $\text{fvar}(S'') \subseteq \text{fvar}(S)$ and a homomorphism $\mu : p_S \rightarrow p''_{S''}$ such that $\mu(x) = x$ for all $x \in \text{fvar}(S'')$. For each subtree S'' of T_1'' containing r_1'', fix exactly one such homomorphism. Create $(T_1''', \lambda_1''', \vec{x}_1)$ from $(T_1'', \lambda_1'', \vec{x}_1)$ by removing all atoms from labels $\lambda''(t)$ of nodes $t \in T_1''$ which never occur in the image of any of these homomorphisms.

(d) *Restore well-designedness.* Traverse T_1''' in a top-down manner and exhaustively carry out the following transformation: if a node t contains a variable z (in some atom α, say) that occurs in some descendant of t but not in the child t' along the path to this descendant, then add atom α to $\lambda(t')$. The resulting pWDPT is p_2.

First of all, it is easy to see that $p_1 \sqsubseteq p_2 \sqsubseteq p$. Clearly, steps (a) and (b) preserve strict equivalence with p_1, i.e., we have $p_1 \equiv p''$. Step (c) may destroy well-designedness but step (d) restores it. Steps (c) and (d) may break equivalence, but the subsumption relationships are preserved, i.e., $p_1 \sqsubseteq p_2$ holds since for each subtree S of p_1, the set of atoms in the corresponding subtree of p_2 is a subset of the atoms in S. This is due to the fact that, in step (d), we only copy atoms "downward" along branches, which does not change the set of atoms in any subtree. Also, $p_2 \sqsubseteq p$ since we keep the images of the required homomorphisms in p_2.

We now show that the number m of nodes in p_2 is bounded by $2 \cdot |\vec{x}_1|$. Together with $\vec{x}_1 \subseteq \vec{x}$, this establishes property (2). Indeed, after step (b), the tree T_1'' only contains nodes that either introduce new free variables (i.e., nodes t such that $\text{fvar}(\lambda''(t)) \setminus \text{var}(\lambda''(\hat{t})) \neq \emptyset$) or have at least two children. Thus T_1'' has at most $|\vec{x}_1|$ leaves, restricting the overall number of nodes in T_1'' to $m \leq 2 \cdot |\vec{x}_1|$ and, clearly, $V(T_2) \subseteq V(T_1'')$.

Third, we need to show that the number of atoms of p_2 is at most exponential in the size of p. The number of subtrees S'' of T_2'', and thus the number of homomorphisms considered in step (c), is bounded by 2^m, while the image of each homomorphism contains at most n atoms, where n is the number of atoms in p. The number k of atoms in T_1''' is thus bounded by $k \leq 2^m \cdot n$. Finally, in step (d) the number of atoms added is again bounded by the number of variables in T_1''' times the number of nodes of T_1'''. As a result the number of atoms in p_2 is bounded by $k + k \cdot m$.

Finally, membership of p_2 in g-$\mathsf{TW}(k)$, respectively, g-$\mathsf{HW}'(k)$, follows. This is because g-$\mathsf{TW}(k)$ and g-$\mathsf{HW}'(k)$ are closed under taking subsets, and for each subtree of p_2, the set of atoms in such a subtree is contained in some subtree of p_1. □

We can now prove Theorem 6.2.

Proof of Theorem 6.2. Let $p \in \mathcal{M}(g\text{-}\mathsf{HW}'(k))$ be a pWDPT, i.e., there exists a pWDPT $p_1 \in g$-$\mathsf{HW}'(k)$ such that $p \equiv_s p_1$. Thus $p_1 \sqsubseteq p$, and, therefore, there exists a pWDPT $p_2 \in \mathsf{HW}'(k)$ satisfying Lemma 6.3. Clearly, properties (2) and (3) of the Lemma imply that the overall size of p_2 is at most exponential in the size of p. Also, by (1), $p \sqsubseteq p_1 \sqsubseteq p_2 \sqsubseteq p$, and thus $p_2 \equiv_s p$. By properties (2) and (3), a simple NExpTime$^{\text{NP}}$ algorithm can work as follows. Given a pWDPT p, (a) guess a pWDPT p' (a linear number of nodes and at most an exponential number of atoms for each node), (b) check whether $p' \in \mathsf{HW}'(k)$, and (c) check whether $p' \equiv_s p$. Checking condition (b) requires an NP-oracle. Condition (c) is satisfied if certain (exponentially many) homomorphisms exist [31]. Such homomorphisms can be guessed alongside p' itself in step (a) and do not increase the complexity.

The proof for g-$\mathsf{TW}(k)$ is analogous, but checking condition (b) no longer needs an NP-oracle, thus leading to the lower complexity. □

While the upper bound in Theorem 6.2 might not be optimal, we can prove that the problem is at least on the second level of the polynomial hierarchy:

Proposition 6.4. *Let $k > 1$ and assume that $C = g$-$\mathsf{TW}(k)$ or $C = g$-$\mathsf{HW}'(k)$. Checking whether a pWDPT p belongs to $\mathcal{M}(C)$ is Π_2^P-hard.*

The proof of this proposition is based on the following lemma.

Lemma 6.5. *For every $k > 1$ and instance of QSAT$_{\forall,2}$ in 3CNF such that the hypergraph of $\phi(\vec{x}, \vec{y})$ (where the nodes are the variables and the hyperedges are the clauses) is connected, there are polynomial-time constructible pWDPTs p_1 and p_2 with $p_2 \in g$-$\mathsf{HW}'(k) \cap g$-$\mathsf{TW}(k)$, satisfying*

(1) $p_1 \sqsubseteq p_2$;
(2) $p_1 \equiv_s p_2$ if and only if Φ is valid;
(3) $p_1 \equiv_s p$ for some $p \in g$-$\mathsf{HW}'(k)$ or $p \in g$-$\mathsf{TW}(k)$ if and only if $p_1 \equiv_s p_2$.

Proof. Let Φ be a $\forall\exists$QBF instance $\Phi = \forall\vec{x}\exists\vec{y}\phi(\vec{x}, \vec{y})$ in 3CNF where $\phi(\vec{x}, \vec{y}) = \bigwedge_{i=1}^{N} C_i$ and, for all $1 \leq i \leq N$, we have that $C_i = (l_{i,1} \vee l_{i,2} \vee l_{i,3})$ with $l_{i,j}$ being a literal over $\vec{x} \cup \vec{y}$. Assume $\vec{x} = x_1, \ldots, x_n$ and $\vec{y} = y_1, \ldots, y_m$. For the construction of the pWDPTs, we define the following sets of atoms:

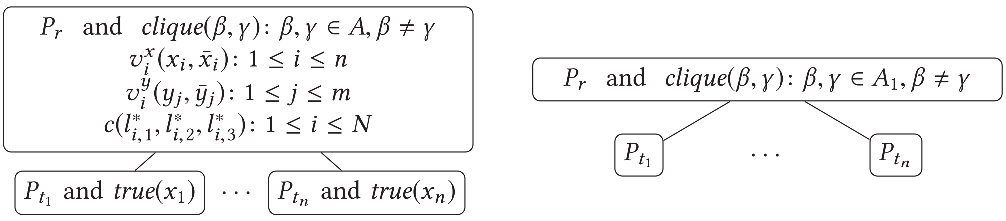
P_r and $clique(\beta,\gamma)\colon \beta,\gamma \in A, \beta \neq \gamma$
$v_i^x(x_i,\bar{x}_i)\colon 1 \leq i \leq n$
$v_i^y(y_j,\bar{y}_j)\colon 1 \leq j \leq m$
$c(l_{i,1}^*, l_{i,2}^*, l_{i,3}^*)\colon 1 \leq i \leq N$
P_{t_1} and $true(x_1)$
$\cdots$
P_{t_n} and $true(x_n)$
P_r and $clique(\beta,\gamma)\colon \beta,\gamma \in A_1, \beta \neq \gamma$
P_{t_1}
$\cdots$
P_{t_n}

Fig. 2. Pattern trees p_1 (left) and p_2 (right) from the proof of Lemma 6.5.

$$\begin{aligned}
P_{xy} &= \{v_i^x(0,1) \mid 1 \leq i \leq n\} \cup \{v_i^y(0,1), v_i^y(1,0) \mid 1 \leq i \leq m\},\\
P_d &= \{clique(1,1), clique(0,0), false(0), true(1)\} \cup \{fix_i(\alpha_i) \mid 2 \leq i \leq k\},\\
P_c &= \{c(0,0,1), c(0,1,0), c(1,0,0), c(0,1,1), c(1,0,1), c(1,1,0), c(1,1,1)\},\\
P_r &= \{map(z)\} \cup P_{xy} \cup P_d \cup P_c,\\
P_{t_i} &= \{v_i^x(1,0), map_i(z_i)\} \text{ for } 1 \leq i \leq n.
\end{aligned}$$

In addition, we consider the variable sets $A_1 = \{1,0\} \cup \{\alpha_i \mid 2 \leq i \leq k\}$ and $A = A_1 \cup \{x_i, \bar{x}_i \mid 1 \leq i \leq n\} \cup \{y_i, \bar{y}_i \mid 1 \leq i \leq m\}$. (Note that also 0 and 1 are variables. This is motivated by the fact that we will reuse the construction in a context where constants are forbidden, cf. Section 6.2. For example, see the proof of Proposition 6.9 in Section D.2 of the online appendix. For readability, we keep the names 0 and 1, since these variables represent the truth values *true* and *false*.)

With these sets, we define the pWDPTs $p_1 = (T_1, \lambda_1, \vec{z})$, with $\vec{z} = \{z\} \cup \{z_i \mid 1 \leq i \leq n\}$, and $p_2 = (T_2, \lambda_2, \vec{z})$ as depicted in Figure 2. Observe that all terms occurring in p_1 are variables and that p_1 has n leaves $t_1, \ldots, t_n$. Finally, we have $l_{i,j}^* = x_\delta$ if $l_{i,j} = x_\delta$ and $l_{i,j}^* = \bar{x}_\delta$ if $l_{i,j} = \neg x_\delta$, and analogously $l_{i,j}^* = y_\delta$ if $l_{i,j} = y_\delta$ and $l_{i,j}^* = \bar{y}_\delta$ if $l_{i,j} = \neg y_\delta$.

The construction of p_1 and p_2 is clearly feasible in polynomial time.

Properties (1), (2), and (3) are proved in Section D.1 of the online appendix. The proof of the first two properties is somewhat standard. $p_1 \sqsubseteq p_2$ can be easily checked, while showing $p_2 \sqsubseteq p_1$ iff Φ is valid works along the same lines as the Π_2^P-hardness proof for subsumption [31]: observe that x_i must be mapped to 0 if t_i is not part of the subtree, and mapped to 1 otherwise. Below we briefly comment on the third property. First of all, we have both $p_2 \in g\text{-HW}'(k)$ and $p_2 \in g\text{-TW}(k)$. It thus suffices to show that $p_1 \equiv_s p_2$ whenever $p_1 \equiv_s p$ for some $p \in g\text{-HW}'(k) \cup g\text{-TW}(k)$. Assume that there exists some $p \in g\text{-HW}'(k) \cup g\text{-TW}(k)$ with $p_1 \equiv_s p$. Since $p_1 \sqsubseteq p_2$ always holds, we need to show $p_2 \sqsubseteq p_1$. Thus consider an arbitrary subtree p_2' of p_2 containing the root node. We claim that for the uniquely defined subtree p_1' of p_1 with $\text{fvar}(p_1') = \text{fvar}(p_2')$, there exists the required homomorphism $\mu_1 : p_1' \to p_2'$ (as established in Proposition 5.2). Its existence can be shown by the following sequence of arguments. Since the set of atoms in p_2' is a subset of the atoms in p_1', μ_1 is an endomorphism $p_1' \to p_1'$ whose image contains only atoms from p_2. Because $p_1 \sqsubseteq p$, there exists a subtree p' of p containing the root with $\text{fvar}(p_1') \subseteq \text{fvar}(p')$ and a homomorphism $\mu' : p' \to p_1'$. Since also $p \sqsubseteq p_1$, there exists a subtree p_1'' of p_1 containing the root with $\text{fvar}(p') \subseteq \text{fvar}(p_1'')$ and a homomorphism $\mu : p_1'' \to p'$. Thus, clearly $\mu'(\mu(\cdot))$ is a homomorphism $p_1'' \to p_1'$, and its restriction to the variables in p_1' is an endomorphism on p_1'. The fact that it is also a homomorphism from $p_1' \to p_2'$ follows from the fact that every endomorphism on p_1 must either be an isomorphism on all variables, or map all $x_i, \bar{x}_i, y_j, \bar{y}_j$ onto 0,1. This is basically due to the assumption that the formula ϕ is connected and there is no $c(\cdot,\cdot,\cdot)$ atom containing variables from both sets, $\{0,1\}$ and $\{x_i, \bar{x}_i, y_j, \bar{y}_j\}$. However, $\mu'(\mu(\cdot))$ being an isomorphism contradicts $p \in g\text{-HW}'(k) \cup g\text{-TW}(k)$, since clearly $p_1 \notin g\text{-HW}'(k) \cup g\text{-TW}(k)$, which proves the case. □

Proof of Proposition 6.4. The proof is by reduction of QSAT$_{\forall,2}$ in 3CNF and an immediate consequence of Lemma 6.5: observe that for p_1 and Φ from Lemma 6.5, the relationship "$p_1 \in \mathcal{M}(g\text{-HW}'(k))$ (respectively, $p_1 \in \mathcal{M}(g\text{-TW}(k))$) if and only if Φ is valid" follows immediately from properties (2) and (3). □

Notice that this establishes a difference to the analogous problem of checking whether a CQ is equivalent to one in a tractable class: for each $k \geq 1$, checking whether a CQ q is equivalent to some CQ q' in $\text{TW}(k)$ is in NP [16].

Evaluation of pWDPTs in $\mathcal{M}(g\text{-TW}(k))$ and $\mathcal{M}(g\text{-HW}'(k))$. An important corollary of Theorem 6.2 is that the partial and maximal evaluation problems for pWDPTs in both $\mathcal{M}(g\text{-TW}(k))$ and $\mathcal{M}(g\text{-HW}'(k))$ are fixed-parameter tractable. Analogously to the problem p-Eval(C), let p-Partial-Eval(C) and p-Max-Eval(C) be the parameterized variants of the problems Partial-Eval(C) and Max-Eval(C), parameterized by the size of the pWDPT.

Corollary 6.6. *Let $k \geq 1$. Then* p-Partial-Eval*($\mathcal{M}(g\text{-TW}(k))$) and* p-Max-Eval*($\mathcal{M}(g\text{-TW}(k))$) are fixed-parameter tractable. The result also holds for $g\text{-HW}'(k)$ instead of $g\text{-TW}(k)$.*

6.2 $g\text{-TW}(k)/g\text{-HW}'(k)$-Approximations of pWDPTs

In general, whenever a query q is not equivalent to one in some desirable class Q, it might be useful to compute an *approximation* of q in Q [6]. Recall that this is a query $q' \in Q$ that is *maximally contained* in q with respect to all queries in Q. In other words, $q' \subseteq q$, and there is no $q'' \in Q$ such that $q' \subset q'' \subseteq q$. For the reasons given at the beginning of Section 5, we define approximations in the pWDPT context not in terms of containment, but subsumption. Throughout this section, we assume that pWDPTs *do not contain constants*. The reason is that the notion of approximations with constants is problematic and not even well understood for CQs [6].

We now define approximations for pWDPTs. Recall that we write $p \sqsubset p'$ to denote that $p \sqsubseteq p'$ but $p \not\equiv_s p'$:

Definition 6.7 (C-approximations). Let C be a class of pWDPTs and $k \geq 1$. Assume p and p' are pWDPTs such that $p' \in C$. Then p' is a *C-approximation* of p if $p' \sqsubseteq p$ and there is no $p'' \in C$ such that $p' \sqsubset p'' \sqsubseteq p$.

Existence of Approximations. An important question in this context is if approximations always exist [6, 10]. The techniques developed in Lemma 6.3 allow us to prove that this is indeed the case in the pWDPT scenario.

Theorem 6.8. *Let $k \geq 1$. For both $g\text{-TW}(k)$ and $g\text{-HW}'(k)$, there is a double-exponential time algorithm that, given a pWDPT p, constructs an exponential-size $g\text{-TW}(k)$-approximation (or $g\text{-HW}'(k)$-approximation, respectively) p' of p.*

Proof. By Lemma 6.3, if there is any approximation p_1 of p, then, because of $p_1 \sqsubseteq p$, there also exists an approximation of size at most $2^{pol(|p|)}$ for some polynomial pol. We thus only need to check pWDPTs up to this size bound. Among those we identify all pWDPTs $\bar{p}$ with $\bar{p} \sqsubseteq p$ and choose one that is maximal with respect to $\sqsubseteq$. Clearly, all pWDPTs p' of size at most $2^{pol(|p|)}$ can be enumerated in double-exponential time. Thus, both correctness and the double-exponential time bound follow immediately. □

Complexity. To better understand the complexity of computing approximations, we study the following decision problem:

$\mathcal{C}$-Approximation	
Input:	Two pWDPTs p, p' such that $p' \in \mathcal{C}$.
Question:	Is p' a $\mathcal{C}$-approximation of p?

The next proposition establishes some upper and lower bounds for the problem.

Proposition 6.9.

(1) *For each $k \geq 1$, g-$\mathsf{TW}(k)$-Approximation is in coNExpTime and g-$\mathsf{HW}'(k)$-Approximation is in coNExpTime$^{\mathrm{NP}}$.*
(2) *For each $k > 1$, g-$\mathsf{TW}(k)$-Approximation and g-$\mathsf{HW}'(k)$-Approximation are Π_2^P-hard.*
(3) *For each $k > 1$, if the input of the problem includes the promise that $p' \sqsubseteq p$, then g-$\mathsf{TW}(k)$-Approximation and g-$\mathsf{HW}'(k)$-Approximation are Σ_2^P-hard.*

Proof. We only prove property (1). Full proofs of properties (2) and (3) are provided in Section D.2 of the online appendix. We first show that g-$\mathsf{HW}'(k)$-Approximation is in coNExpTime$^{\mathrm{NP}}$. Given pWDPTs p_1 and p_2 with $p_2 \in g$-$\mathsf{HW}'(k)$, to check whether p_2 is a g-$\mathsf{HW}'(k)$-approximation of p_1, one has to ensure that $p_2 \sqsubseteq p_1$ and there does not exist $p_3 \in g$-$\mathsf{HW}'(k)$ such that $p_2 \sqsubset p_3 \sqsubseteq p_1$.

The first property can be decided in Π_2^P. Moreover, by Lemma 6.3, we conclude that if there exists some pWDPT p_3 satisfying the second property, then there also exists one of size at most exponential in the size of p_1. Thus, to look for a counterexample to p_2 being an approximation, it suffices to check pWDPTs of exponential size. In fact, by Lemma 6.3, there exists a pWDPT $p_3 = (T_3, \lambda_3, \vec{x}_3)$ such that the number of nodes in T_3 is at most twice the number of free variables in p_1. Thus, the exponential blowup can only happen in the size of the labels λ_3, but not in the tree-size. More specifically, the number of subtrees of T_3 is bounded by $4^{|p_1|}$. This property allows us to check in NExpTime$^{\mathrm{NP}}$ whether there exists a witness for p_2 not being an approximation—thus providing a coNExpTime$^{\mathrm{NP}}$ algorithm for the problem of deciding whether p_2 is an approximation.

The crucial part in the algorithm to decide if p_2 is *not* an approximation of p_1 is the guess of an exponential-size witness, which consists of the following parts (assume that r_2 is the root of p_2):

(1) the pWDPT p_3 (let r_3 denote its root),
(2) for every subtree p_3' of p_3 rooted in r_3 a homomorphism $\mathrm{minSubtree}(p_1, \mathrm{fvar}(p_3')) \to p_3'$,
(3) for every subtree p_2' of p_2 rooted in r_2 a homomorphism $\mathrm{minSubtree}(p_3, \mathrm{fvar}(p_2')) \to p_2'$,
(4) one subtree p_3'' of p_3 together with the set $\mathcal{H}_2$ of all mappings $\mathrm{var}(p_2'') \to \mathrm{dom}(p_3'')$ where $p_2'' = \mathrm{minSubtree}(p_2, \mathrm{fvar}(p_3''))$.

After guessing a witness, the algorithm performs the following checks:

— check that $p_3 \in g$-$\mathsf{HW}'(k)$,
— check that the mappings guessed in steps (2) and (3) are indeed the required homomorphisms (this shows that $p_3 \sqsubseteq p_1$ and $p_2 \sqsubseteq p_3$, respectively),
— check that no mapping in $\mathcal{H}_2$ is a homomorphism $p_2'' \to p_3''$ (this shows that $p_3 \not\sqsubseteq p_2$).

All steps can be done in exponential time with an NP-oracle for deciding whether $p_3 \in g$-$\mathsf{HW}'(k)$.

It thus remains to discuss the size of the witness. We get the following size bounds for each of the four parts of the witness. The size of p_3 is bounded by $O(4^{|p_1|} \cdot |p_1|)$. The number of subtrees of p_3 is in $O(4^{|p_1|})$, and the size of a homomorphism in (2) is in $O(|p_1|)$, giving for (2) the bound of $O(4^{|p_1|})$. The number of subtrees of p_2 is in $O(2^{|p_2|})$, and the size of a homomorphism in (3) is in $O(2^{|p_1|})$, resulting for (3) in the bound of $O(2^{|p_2|} \cdot 2^{|p_1|})$. For (4), we get a bound of

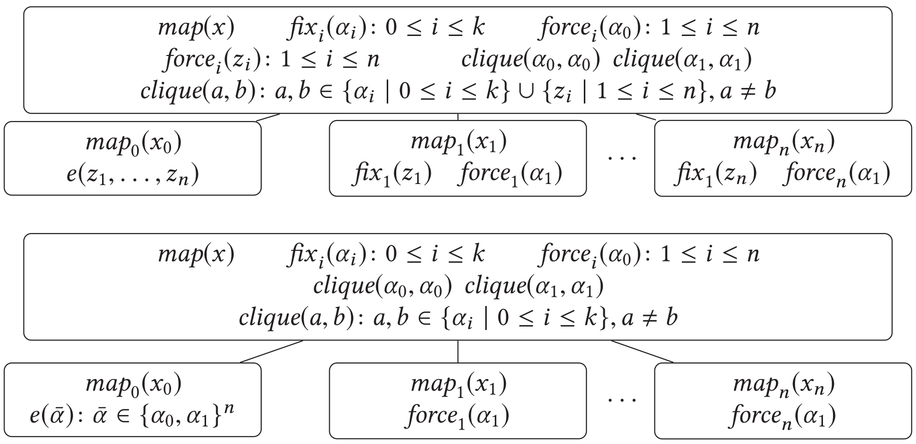
$map(x)$ $fix_i(\alpha_i)\colon 0 \le i \le k$ $force_i(\alpha_0)\colon 1 \le i \le n$
$force_i(z_i)\colon 1 \le i \le n$ $clique(\alpha_0, \alpha_0)$ $clique(\alpha_1, \alpha_1)$
$clique(a, b)\colon a, b \in \{\alpha_i \mid 0 \le i \le k\} \cup \{z_i \mid 1 \le i \le n\}, a \neq b$
$map_0(x_0)$ $e(z_1, \ldots, z_n)$
$map_1(x_1)$ $fix_1(z_1)$ $force_1(\alpha_1)$
$\ldots$
$map_n(x_n)$ $fix_1(z_n)$ $force_n(\alpha_1)$
$map(x)$ $fix_i(\alpha_i)\colon 0 \le i \le k$ $force_i(\alpha_0)\colon 1 \le i \le n$
$clique(\alpha_0, \alpha_0)$ $clique(\alpha_1, \alpha_1)$
$clique(a, b)\colon a, b \in \{\alpha_i \mid 0 \le i \le k\}, a \neq b$
$map_0(x_0)$ $e(\bar{\alpha})\colon \bar{\alpha} \in \{\alpha_0, \alpha_1\}^n$
$map_1(x_1)$ $force_1(\alpha_1)$
$\ldots$
$map_n(x_n)$ $force_n(\alpha_1)$

Fig. 3. Exponential blow-up from p_1 (top) to p_2 (bottom) (Theorem 6.10).

$O(4^{|p_1|} \cdot |p_1| \cdot 2^{2 \cdot |p_1| \cdot |p_2|} \cdot |p_2|)$, which is the size of a subtree of p_3 times the number of mappings times the size of such a mapping.

Thus, the overall size of the witness is clearly exponential in the input size. This concludes the proof for g-HW$'(k)$. The case for g-TW(k) is almost identical. The only difference is that deciding $p_3 \in g$-TW(k) does not need an NP-oracle. □

Again, this shows that our problem is harder than the analogous problem for CQs. For each $k \geq 1$, the problem of checking if a CQ q_1 is a TW(k)-approximation of a CQ q_2 is DP-complete. If, in addition, the input includes the promise that $q_1 \subseteq q_2$, then it becomes coNP-complete [6].

Size of Approximations. We have seen above that approximations always exist even though Lemma 6.3 only allowed us to give an exponential upper bound on their size. The proof of that lemma centered around the properties of subsumption—without making use of the specific properties of approximations. One may thus ask if exponential size is indeed attainable by approximations in a *non-trivial* way. By non-trivial, we mean that an approximation p_2 of a pWDPT p_1 is exponentially bigger than p_1 and every pWDPT p_3 with $p_3 \equiv_s p_2$ is at least as big as p_2. In other words, p_2 is necessarily that big and has not just been artificially blown up by adding redundant atoms. Below, we give an affirmative answer to this question. This establishes another sharp contrast with CQs, where every TW(k)-approximation is equivalent to one of polynomial size [6].

Theorem 6.10. *For every $k \geq 2$, there exists a sequence of pairs of pWDPTs $(p_1^{(n)}, p_2^{(n)})_{n \in \mathbb{N}}$, such that $p_2^{(n)}$ is a* g-TW(k)/g-HW$'(k)$*-approximation of $p_1^{(n)}$, and $p_2^{(n)}$ is* exponentially bigger *than $p_1^{(n)}$. More precisely, we have $|p_1^{(n)}| = O(n^2)$ and $|p_2^{(n)}| = \Omega(2^n)$ and, for every pWDPT $p_3^{(n)} \in$* g-TW(k)/g-HW$'(k)$ *with $p_2^{(n)} \sqsubseteq p_3^{(n)} \sqsubseteq p_1^{(n)}$, we have $|p_3^{(n)}| \geq |p_2^{(n)}|$.*

Proof. Let $k \geq 1$ and $n \geq 1$. Consider the pWDPTs $p_1^{(n)}$ and $p_2^{(n)}$ as defined in Figure 3 with free variables $\vec{x} = \{x, x_0, \ldots, x_n\}$. For the sake of readability, we omit superscript (n) from now on. It is easy to verify that p_2 is exponentially bigger than p_1 and that $p_2 \sqsubseteq p_1$ holds. Also, clearly $p_1 \notin g$-TW(k) and $p_1 \notin g$-HW$'(k)$ due to the clique of size $k + 1 + n$ (formed by the *clique*-atoms) in the root node r of p_1 while $p_2 \in g$-TW(k) and $p_1 \in g$-HW$'(k)$ holds. Note that, to prove the theorem, it is not necessary that p_2 is indeed a g-TW(k)/g-HW$'(k)$-approximation of p_1. But of course, if it is not, then there exists a g-TW(k)/g-HW$'(k)$-approximation p_3 of p_1 with $p_2 \sqsubseteq p_3 \sqsubseteq p_1$. Hence, to prove the theorem, it suffices to show that for every pWDPT $p_3 \in g\text{-TW}(k) \cup g\text{-HW}'(k)$ with $p_2 \sqsubseteq p_3 \sqsubseteq p_1$, we have $|p_3| \geq |p_2|$. We do so by a number of claims showing that p_3 has at least as many nodes as p_2, and that it contains at least all atoms from p_2.

Thus, consider an arbitrary pWDPT $p_3 \in g\text{-TW}(k) \cup g\text{-HW}'(k)$ with $p_2 \sqsubseteq p_3 \sqsubseteq p_1$. First of all, we clearly have $\text{fvar}(p_1) = \text{fvar}(p_2) = \text{fvar}(p_3)$.

Claim 1. *p_3 contains (at least) the following subtrees:*

— *a subtree (call it S) whose only free variable is x, and*
— *for every $0 \leq i \leq n$, a subtree (call it S_i) extending S, whose only additional free variable is x_i.*

Proof of Claim 1. Consider the root r' of p_2 with $\text{fvar}(r') = \{x\}$. By Proposition 5.2 and $p_2 \sqsubseteq p_3$, there exists a subtree p_3' of p_3 with $\text{fvar}(r') \subseteq \text{fvar}(p_3')$ and a homomorphism $p_3' \to r'$. We claim that p_3' is the desired subtree S, i.e., $\{x\} = \text{fvar}(r') = \text{fvar}(p_3')$. Suppose to the contrary that $\text{fvar}(r') \subset \text{fvar}(p_3')$. Then, by $p_3 \sqsubseteq p_1$, there exists a subtree p_1' of p_1 with $\text{fvar}(p_3') \subseteq \text{fvar}(p_1')$ and homomorphism $p_1' \to p_3'$. In total, we thus have $\text{fvar}(r') \subset \text{fvar}(p_1')$ and $p_1' \to r'$. But this is a contradiction since the only subtree of p_1 with a homomorphism into r' is the root r of p_1 with $\text{fvar}(r') = \text{fvar}(r)$. The existence of the subtrees $S_0, \ldots, S_n$ is shown analogously. □

For the rest of the proof of Theorem 6.10, we assume without loss of generality that p_3 consists of a root node, $n+1$ leaf nodes t_i'', and an unspecified number of intermediate nodes. As shown above, S has exactly one free variable x, introduced in some node, say r''. We may thus “contract” subtree S to the single node r'' by dropping all nodes that are not ancestors of r'' and adding all atoms from the ancestors of r'' to r''.

Likewise, every subtree S_i with $0 \leq i \leq n$ has precisely one additional free variable compared to r''. This additional free variable is introduced in some node, say t_i''. Hence, each subtree S_i is a path from r'' to t_i'', since all nodes in S_i not on this path can again be dropped. Observe that we do not assume the paths to be disjoint even when omitting the root node (i.e., for $i \neq j$, it might happen that $(S_i \cap S_j) \setminus \{r''\} \neq \emptyset$). However, we have $t_i'' \notin S_j$ for all $i \neq j$, and we assume that the parent of each leaf has as least two descendants $t_i'' \neq t_j''$ (this can be easily achieved by “contracting” all nodes between the last node in S_i with this property and t_i'' into t_i''). Observe that dropping such nodes violates neither $p_3 \in g\text{-TW}(k) \cup g\text{-HW}'(k)$ nor $p_2 \sqsubseteq p_3 \sqsubseteq p_1$.

Claim 2. *Up to renaming of the existential variables, the node r'' of p_3 contains (at least) the atoms $\mathit{fix}_0(\alpha_0), \ldots, \mathit{fix}_k(\alpha_k)$ with pairwise distinct variables $\alpha_0, \ldots, \alpha_k$. Moreover, all atoms in p_1 with variables exclusively in $\{\alpha_0, \ldots, \alpha_k\}$ are also present in the corresponding nodes of p_3.*

Proof of Claim 2. By $p_3 \sqsubseteq p_1$, there exists a homomorphism $\mu : p_1 \to p_3$. Let $\mu(\alpha_0) = \beta_0, \ldots, \mu(\alpha_k) = \beta_k$ denote the function values of the variables α_i under μ. We claim that the variables β_i are pairwise distinct. Suppose to the contrary that $\beta_i = \beta_j$ holds for some pair $i \neq j$. Then p_3 contains atoms $\mathit{fix}_i(\beta_i)$ and $\mathit{fix}_j(\beta_i)$. However, we also assume $p_2 \sqsubseteq p_3$. Hence, there also exists a homomorphism $\mu' : p_3 \to p_2$. But the only atoms with relation symbols fix_i and fix_j in p_2 are $\mathit{fix}_i(\alpha_i)$ and $\mathit{fix}_j(\alpha_j)$ with $\alpha_i \neq \alpha_j$. Hence, we have $\mu'(\beta_i) = \alpha_i \neq \alpha_j = \mu'(\beta_i)$, which is a contradiction. Therefore, without loss of generality, we may identify $\beta_0, \ldots, \beta_k$ with $\alpha_0, \ldots, \alpha_k$.

Consequently, all atoms in p_1 with variables exclusively in $\{\alpha_0, \ldots, \alpha_k\}$ are also present in p_3. It remains to show that these atoms occur indeed in the corresponding nodes, i.e., in r'' or t_i'', respectively. To this end, we inspect the subtree of p_3 consisting of the root r'' only and then the subtrees S_i. Root r'' must contain all of these atoms in the root r of p_1 since otherwise the subsumption test for $p_3 \sqsubseteq p_1$ would fail for the subtree of p_3 containing the root only. Next, assume that any of the atoms $\mathit{force}_i(\alpha_1)$ is contained in any node except t_i''. This contradicts the assumption $p_2 \sqsubseteq p_3$: just consider the subtree p_2' of p_2 consisting of all nodes except t_i' (the node in p_2 corresponding to t_i''). Then the corresponding subtree of p_3 contains $\mathit{force}_i(\alpha_1)$. However, the only $\mathit{force}_i(\cdot)$ atom contained in p_2' is $\mathit{force}_i(\alpha_0)$, but we have just shown that $\mu(\alpha_1) \neq \mu(\alpha_0)$. Hence, each of these atoms $\mathit{force}_i(\alpha_1)$ is contained in the corresponding leaf node t_i'' of p_3. □

Claim 3. *Up to renaming of the existential variables, for every subtree p_2' of p_2 and subtree p_3' of p_3 with $\mathrm{fvar}(p_2') \subseteq \mathrm{fvar}(p_3')$ and homomorphism $\mu : p_3' \to p_2'$ that is the identity on $\mathrm{fvar}(p_2')$, we have that $\mu(\alpha_i) = \alpha_i$ for $0 \leq i \leq k$.*

Proof of Claim 3. Follows immediately from Claim 2 and the fact that the only atoms with relation symbols fix_i in p_2 are of the form $\mathit{fix}_i(\alpha_i)$ for $0 \leq i \leq k$. □

Claim 4. *For every $1 \leq i \leq n$, we have $\lambda_2(t_i') \subseteq \lambda_3(t_i'')$, i.e., the atoms labeling a leaf node t_i' of p_2 are also present in the corresponding leaf node t_i'' of p_3.*

Proof of Claim 4. For the atoms $\mathit{force}_i(\alpha_1)$, this follows immediately from Claim 2, since they also occur in p_1 and of course $\alpha_1 \in \{\alpha_0, \ldots, \alpha_k\}$. By $p_3 \sqsubseteq p_1$, $\lambda_3(S_i) = \bigcup_{t \in S_i} \lambda(t)$ must also contain the atom $\mathit{map}_i(x_i)$. Finally, by the assumption on the tree structure of p_3, the atom $\mathit{map}_i(x_i)$ is actually contained in t_i'' within S_i. □

Claim 5. *Let p_3' be an arbitrary subtree of p_3, and $\mu : p_1' \to p_3'$ the homomorphism from the corresponding subtree p_1' of p_1. Then, up to renaming of variables, $\mu(z_i) = \alpha_0$ if t_i'' is not contained in p_3', and $\mu(z_i) = \alpha_1$ otherwise.*

Proof of Claim 5. Consider an arbitrary $i \in \{1, \ldots, n\}$. First note that z_i is part of a big clique of *clique*-atoms in p_1. By Claim 2, $\lambda_3(r'')$ contains the clique of size $k + 1$ of *clique*-atoms with the variables $\{\alpha_0, \ldots, \alpha_k\}$ (up to renaming). Hence, since $p_3 \in g\text{-}\mathsf{TW}(k) \cup g\text{-}\mathsf{HW}'(k)$, $\mu(z_i) = \alpha_j$ holds for some $j \in \{0, \ldots, k\}$ (otherwise, p_3 would contain a clique of size bigger than $k + 1$, contradicting $p_3 \in g\text{-}\mathsf{TW}(k) \cup g\text{-}\mathsf{HW}'(k)$). Next, suppose $j \geq 2$. This contradicts the condition $p_2 \sqsubseteq p_3$ since, by Claim 3, every homomorphism $\mu' : (\lambda_3(S_i)) \to (\lambda_2(r') \cup \lambda_2(t_i'))$ restricted to $\{\alpha_0, \ldots, \alpha_k\}$ is the identity; but p_2 does not contain an atom $\mathit{clique}(\mu'(z_i), \mu(\alpha_j)) = \mathit{clique}(\alpha_j, \alpha_j)$ with $j \geq 2$.

Now assume that t_i'' is not in p_3', and toward a contradiction assume $\mu(z_i) = \alpha_1$. Again by $p_3 \sqsubseteq p_1$, this requires $\mathit{force}_i(\alpha_1)$ to be contained in p_3', which again contradicts $p_2 \sqsubseteq p_3$. Just consider the subtree p_2' of p_2 that contains r' and exactly those children t_j' such that t_j'' is contained in p_3'. Since $p_2 \sqsubseteq p_3$ there must be some homomorphism $\mu' : p_3' \to p_2'$. By Claim 3, we know that μ' is the identity on α_1, thus there must be $\mathit{force}_i(\alpha_1)$ in p_2', which is not the case. This gives the desired contradiction. Hence $\mu(z_i) = \alpha_0$ must hold for all z_i such that t_i'' is not contained in p_3'.

Next, assume that t_i'' is in p_3', and toward a contradiction that $\mu(z_i) = \alpha_0$. Then, analogously to the previous case, $\mathit{fix}_1(\alpha_0)$ must belong to p_3', which contradicts $p_2 \sqsubseteq p_3$ since $\mathit{fix}_1(\alpha_0)$ does not occur in p_2 (observe that $\mathit{force}_i(\alpha_1)$ is no problem in this case since it is contained in $\lambda_2(t_i')$). □

Claim 6. *$\lambda_2(t_0') \subseteq \lambda_3(t_0'')$.*

Proof of Claim 6. By $p_3 \sqsubseteq p_1$, the set $\lambda_3(t_0'')$ must contain an atom of the form $\mathit{map}_0(x_0)$. It remains to show that also every atom of the form $e(\alpha_{i_1}, \ldots, \alpha_{i_n})$ is contained in $\lambda_3(t_0'')$, where $i_j \in \{0, 1\}$ for every j. Thus let $e(\alpha_{i_1}, \ldots, \alpha_{i_n})$ be an arbitrary atom with $i_j \in \{0, 1\}$ for every j. Consider the subtree p_3' of p_3 that contains r'', and exactly those leaf nodes t_j'' such that $i_j = 1$ together with the path from r'' to t_j''. Let p_1' and $\mu : p_1' \to p_3'$ be the corresponding subtree and homomorphism according to $p_3 \sqsubseteq p_1$. Then, by Claim 5, $e(\mu(z_1), \ldots, \mu(z_n)) = e(\alpha_{i_1}, \ldots, \alpha_{i_n})$, which is thus contained in p_3'. Finally, by the same arguments as in the proof of Claim 4, it must actually be contained in $\lambda_3(t_0'')$, since otherwise we get a contradiction to $p_2 \sqsubseteq p_3$. □

We now conclude the proof of Theorem 6.10. By the assumption on the tree structure of p_3, it consists of at least as many nodes as p_2. By Claim 2, $\lambda_2(r') \subseteq \lambda_3(r'')$. By Claims 4 and 6, $\lambda_2(t_i') \subseteq \lambda_3(t_i'')$. Thus, p_3 is at least as big as p_2.

7 UNIONS OF PWDPTS

Closing pWDPTs under union constitutes one of the basic extensions of the language [36, 38]. Formally, a *union of pWDPTs* (UpWDPT) is an expression ρ of the form $\bigcup_{1 \le i \le n} p_i$, where each p_i is a pWDPT over σ. (Notice that we do not require different p_i's to have the same set of free variables.) The evaluation of ρ over database $\mathcal{D}$, denoted $\rho(\mathcal{D})$, is the set $\bigcup_{1 \le i \le n} p_i(\mathcal{D})$. Motivated by the language features of SPARQL 1.0, in our setting it might seem more natural to define projection on top of the union, instead of unions of pWDPTs. Observe however, that every query in one of these two formalisms can be easily transformed into an equivalent query in the other formalism by renaming conflicting variables apart. We thus opt for UpWDPT since they are also syntactically extensions of the pWDPTs considered so far.

As before, we write $\rho \sqsubseteq \rho'$, for UpWDPTs ρ and ρ', if for every database $\mathcal{D}$ and partial mapping $h \in \rho(\mathcal{D})$ there is an $h' \in \rho'(\mathcal{D})$ such that $h \sqsubseteq h'$. Similarly, we write $\rho \equiv_s \rho'$ whenever $\rho \sqsubseteq \rho'$ and $\rho' \sqsubseteq \rho$, and we write $\rho \sqsubset \rho'$ if $\rho \sqsubseteq \rho'$ but $\rho \not\equiv_s \rho'$.

To keep the notation simple, we overload the problem definitions of Eval($\mathcal{C}$), Partial-Eval($\mathcal{C}$), and Max-Eval($\mathcal{C}$) for classes $\mathcal{C}$ of UpWDPTs, instead of introducing new problem names. That is, these problems are defined as before, but instead of pWDPTs they take UpWDPTs as input. It is immediate that unions of pWDPTs from a tractable class $\mathcal{C}$ in terms of (variants of) evaluation preserve the good properties of $\mathcal{C}$. Formally, let $\bigcup$-$\mathcal{C}$ be the class of UpWDPTs that consist of unions of pWDPTs in $\mathcal{C}$. Then:

Theorem 7.1. *The following holds for each $k \ge 1$ assuming $\mathcal{Q} = \mathsf{TW}(k)$ or $\mathsf{HW}(k)$:*

(1) the problem Eval($\bigcup$-(ℓ-$\mathcal{Q} \cap \mathsf{BI}(c)$)) *is in* LogCFL *for each $c \ge 1$;*
(2) Partial-Eval($\bigcup$-(g-$\mathcal{Q}$)) *and* Max-Eval($\bigcup$-(g-$\mathcal{Q}$)) *are in* LogCFL.

In other words, the additional expressive power of UpWDPTs compared to pWDPTs has no effect on our variants of the evaluation problem. We look next again at the semantic space defined by these syntactic classes of UpWDPTs. It will turn out that the extension from pWDPTs to UpWDPTs makes a difference.

Reformulations in Tractable Classes. Analogously to Section 6, for a class $\mathcal{C}$ of UpWDPTs, the class $\mathcal{M}(\mathcal{C})$ contains all UpWDPTs $\equiv_s$-equivalent to queries in $\mathcal{C}$:

$$\mathcal{M}(\mathcal{C}) \;=\; \{\rho \mid \rho \equiv_s \rho', \text{for some } \rho' \text{ in } \mathcal{C}\}.$$

Following Section 6, we concentrate on UpWDPTs from $\bigcup$-$\mathcal{C}$, where $\mathcal{C}$ is either g-$\mathsf{TW}(k)$ or g-$\mathsf{HW}'(k)$. We prove below that for these two choices of $\mathcal{C}$, the class $\mathcal{M}(\bigcup$-$\mathcal{C})$ is not only decidable but allows for a nice characterization. To present this characterization, it is convenient to introduce some notation first.

Given a pWDPT $p = (T, \lambda, \vec{x})$ and a subtree T' of T containing the root of T, we need a slight variation of the CQ$q_{T'}$. Denote by $qp_{T'}$ the CQ $\mathrm{Ans}(\vec{x}') \leftarrow R_1(\vec{v}_1), \ldots, R_m(\vec{v}_m)$, where $\{R_1(\vec{v}_1), \ldots, R_m(\vec{v}_m)\} = \mathrm{atoms}(T')$ is the set of atoms in T' and $\vec{x}'$ is the set of variables that appear in $\vec{x}$ and in some $\vec{v}_i$, for $1 \le i \le m$. In other words, $qp_{T'}$ is exactly as $q_{T'}$, but the projection now is only over variables from the $R_i(\vec{v}_i)$'s that are free in p (or, by slight abuse of notation, $qp_{T'} = q_{T'}(\vec{x}')$). For a UpWDPT ρ, we then define

$$\rho_{\mathrm{CQ}} \;=\; \bigcup_{(T,\lambda,\vec{x}) \in \rho} \quad \bigcup_{T' \text{ a subtree of } T \text{ rooted in } r} qp_{T'}.$$

Example 7.2. Consider the RDF pWDPT p introduced in Example 1.1, and the projection onto the variables $\{y, z, z'\}$ introduced in Example 1.3. When replacing the triple patterns by binary atoms, we get ρ_{CQ} as the union of the following CQs:

— $\text{Ans}(y) \leftarrow \texttt{recorded\_by}(x, y), \texttt{published}(x, \text{“after\_2010”})$;
— $\text{Ans}(y, z) \leftarrow \texttt{recorded\_by}(x, y), \texttt{published}(x, \text{“after\_2010”}), \texttt{NME\_rating}(x, z)$;
— $\text{Ans}(y, z') \leftarrow \texttt{recorded\_by}(x, y), \texttt{published}(x, \text{“after\_2010”}), \texttt{formed\_in}(y, z')$;
— $\text{Ans}(y, z, z') \leftarrow \texttt{recorded\_by}(x, y), \texttt{published}(x, \text{“after\_2010”}), \texttt{NME\_rating}(x, z),$ $\texttt{formed\_in}(y, z')$.

Thus, ρ_{CQ} can be seen as a union of CQs (UCQ), but under a slightly more relaxed notion than the one often used in the literature: the CQs in ρ_{CQ} may contain different tuples of free variables, while UCQs are usually defined as unions of CQs all of which have the same tuple of free variables.

Proposition 7.3. *Let ρ be a UpWDPT. Then $\rho \equiv_s \rho_{\text{CQ}}$.*

Proof. For some database $\mathcal{D}$, consider $h \in \rho(\mathcal{D})$. By definition $h \in p_i(\mathcal{D})$ for some $p_i = (T, \lambda, \vec{x})$ in ρ. And so, $h = h'_{\vec{x}}$ for some maximal homomorphism h' from p_i to $\mathcal{D}$. Let T' be the subtree of T with $h' \in q_{T'}(\mathcal{D})$. Then clearly $h \in qp_{T'}(\mathcal{D})$, which shows $\rho \sqsubseteq \rho_{\text{CQ}}$.

For the converse direction, consider an arbitrary $h \in \rho_{\text{CQ}}(\mathcal{D})$. Then there exists some $p_i = (T, \lambda, \vec{x})$ in ρ such that $h \in qp_{T'}(\mathcal{D})$ for some subtree T' of T. Consider the corresponding query $q_{T'}$: there clearly exists some extension h' of h such that $h' \in q_{T'}(\mathcal{D})$. Now if h' is actually a maximal homomorphism, we have $h'_{\vec{x}} = h \in \rho(\mathcal{D})$, and we are done. Otherwise, if h' is not maximal, then there exists some $h'' \in q_{T''}(\mathcal{D})$ (where T'' is also a subtree of T) with $h' \sqsubset h''$. Hence $h''_{\vec{x}} \in \rho(\mathcal{D})$, and thus $\rho_{\text{CQ}} \sqsubseteq \rho$. □

With ρ_{CQ} we have a useful tool for our further analysis of the classes $\mathcal{M}(\bigcup\text{-}C)$ for $C = g\text{-TW}(k)$ and $C = g\text{-HW}'(k)$. In particular, this allows us to give the following characterization of $\mathcal{M}(\bigcup\text{-}C)$.

Proposition 7.4. *Let $k \geq 1$ and $\mathcal{Q} = \text{TW}(k)$ or $\text{HW}'(k)$. A UpWDPT ρ is in $\mathcal{M}(\bigcup\text{-}(g\text{-}\mathcal{Q}))$ if and only if ρ_{CQ} is $\equiv_s$-equivalent to a union of CQsin $\bigcup\text{-}\mathcal{Q}$.*

Proof. For the right-to-left direction, a union of CQsin $\bigcup\text{-}\mathcal{Q}$ is of course a UpWDPT in $\bigcup\text{-}(g\text{-}\mathcal{Q})$, since each CQis a single-node pWDPT in $g\text{-}\mathcal{Q}$. Thus $\rho_{\text{CQ}} \in \mathcal{M}(\bigcup\text{-}(g\text{-}\mathcal{Q}))$ by definition. Finally, because of $\rho \equiv_s \rho_{\text{CQ}}$ also $\rho \in \mathcal{M}(\bigcup\text{-}(g\text{-}\mathcal{Q}))$ holds by definition. For the other direction, assume that $\rho \equiv_s \rho^*$ for some UpWDPT $\rho^* \in \bigcup\text{-}(g\text{-}\mathcal{Q})$. Then $\rho^*_{\text{CQ}} \equiv_s \rho^* \equiv_s \rho \equiv_s \rho_{\text{CQ}}$, and thus $\rho^*_{\text{CQ}} \equiv_s \rho_{\text{CQ}}$. Since each pWDPT in ρ^* is in $g\text{-}\mathcal{Q}$, it follows that ρ^*_{CQ} is indeed a union of CQsin $\bigcup\text{-}\mathcal{Q}$, which proves the case. □

By applying Proposition 7.4, we obtain the following:

Theorem 7.5. *The following properties hold for each $k \geq 1$:*

(1) Given a UpWDPT ρ, checking whether ρ is in $\mathcal{M}(\bigcup\text{-}(g\text{-TW}(k)))$ is Π_2^P-complete, and whether it is in $\mathcal{M}(\bigcup\text{-}(g\text{-HW}'(k)))$ is Π_2^P-hard and in Π_3^P.
(2) For $C = g\text{-TW}(k)$ or $C = g\text{-HW}'(k)$, there is an ExpTime algorithm that, given ρ in $\mathcal{M}(\bigcup\text{-}C)$, constructs a union ρ' of (possibly exponentially many) pWDPTs in C such that the size of each pWDPT in ρ' is polynomial in the size of ρ, and $\rho \equiv_s \rho'$.

Proof. We start by showing the following claim: for any CQq and integer $k \geq 1$, deciding whether there exists a CQ$q' \in \text{TW}(k)$ such that $q \equiv q'$ is in NP, and in Σ_2^P for $q' \in \text{HW}'(k)$. To see that this is the case, consider the following algorithm:

(i) guess a subquery q' of q and a mapping $\mu : \text{var}(q) \rightarrow \text{var}(q')$, and
(ii) return yes if $\mu : q \rightarrow q'$ is a homomorphism and $q' \in \mathsf{TW}(k)$ or $\mathsf{HW}'(k)$, respectively; otherwise, return no.

That is, the algorithm looks for a subquery of q that is in $\mathsf{TW}(k)$ or $\mathsf{HW}'(k)$, respectively, and is a homomorphic image of q. For $\mathsf{TW}(k)$, the check is feasible in polynomial time, thus the overall algorithm is in NP. For $\mathsf{HW}'(k)$, the best known upper bound for checking whether $q' \in \mathsf{HW}'(k)$ is coNP, resulting in the higher complexity.

To see that the algorithm is also correct, we show that there exists some subquery of q in $\mathsf{TW}(k)$ equivalent to q if and only if q is equivalent to some CQ$q'' \in \mathsf{TW}(k)$ (for the sake of readability, we restrict ourselves to $\mathsf{TW}(k)$, the proof for $\mathsf{HW}'(k)$ is identical). Of course, the left-to-right direction is trivial. Thus assume that q is equivalent to some CQ$q'' \in \mathsf{TW}(k)$. If q'' is some subquery of q, then the case is again obvious. Thus assume that this is not the case. We show that then there exists a subquery q' of q that is equivalent to q and q'', and—since $\mathsf{TW}(k)$ is closed under taking images under arbitrary mappings—is also in $\mathsf{TW}(k)$. To see that this is indeed true, consider the homomorphisms $\mu' : q'' \rightarrow q$ and $\mu : q \rightarrow q''$ (which exist because $q \equiv q''$) and define $q' = \mu'(\mu(q))$. Then q' is by definition a subquery of q. Moreover, q' is equivalent to q.

Having settled this claim, we now show property (1). Let ρ^-_{CQ} be the union of CQsthat is obtained by removing from ρ_{CQ} every CQq that is subsumed by another CQq' in ρ_{CQ}. Since it can be easily shown that $q \sqsubseteq \rho_{\text{CQ}} \setminus \{q\}$ if and only if $q \sqsubseteq q'$ for some $q' \in \rho_{\text{CQ}}$ (with $q \neq q'$), there is no $q \in \rho^-_{\text{CQ}}$ such that $q \sqsubseteq \rho^-_{\text{CQ}} \setminus \{q\}$. By Proposition 7.4, for a class $\mathcal{Q}$ of CQs, we have that $\rho \in \mathcal{M}(\bigcup\text{-}(g\text{-}\mathcal{Q}))$ if and only if ρ_{CQ} is $\equiv_s$-equivalent to a union of CQsin $\bigcup\text{-}\mathcal{Q}$. We show in Section E.1 of the online appendix that this is the case if and only if each CQin ρ^-_{CQ} is equivalent to a CQin $\mathcal{Q}$. This gives us the following nondeterministic algorithm to check whether $\rho \notin \mathcal{M}(\bigcup\text{-}(g\text{-}\mathsf{TW}(k)))$:

— guess a CQ$q \in \rho_{\text{CQ}}$ (all of them are of polynomial size);
— check that (i) $q \not\sqsubseteq q'$ for every CQ $q \neq q' \in \rho_{\text{CQ}}$, and (ii) $q \not\equiv q'$ for every CQ$q' \in \mathsf{TW}(k)$.

As mentioned above, $q \equiv q'$ if and only if $q \equiv_s q'$. Clearly, checking (i) is in coNP. For (ii), recall the initial claim that the complement of the problem, i.e., deciding the existence of some CQ$q' \in \mathsf{TW}(k)$ satisfying $q \equiv q'$, is in NP. Thus, (ii) can be decided in coNP, giving the desired upper-bound. Deciding $\rho \notin \mathcal{M}(\bigcup\text{-}(g\text{-}\mathsf{HW}'(k)))$ works analogously, the only difference is that the aforementioned claim provides Σ^P_2-membership, thus (ii) can be decided in Π^P_2, again giving the desired upper bound.

The hardness follows from exactly the same reduction and arguments as used in the proof of Lemma 6.5. In fact, by applying exactly the arguments that were used to show that $p_2 \equiv_s p_1$ whenever there exists some pWDPT $p \in \mathcal{C}$ (for both $\mathcal{C} = g\text{-}\mathsf{TW}(k)$ and $\mathcal{C} = g\text{-}\mathsf{HW}'(k)$) with $p \equiv_s p_1$, one can also show that $p_2 \equiv_s p_1$ whenever there exists some UpWDPT $p' \in \bigcup\text{-}\mathcal{C}$ with $p' \equiv_s p_1$ (the subtrees of p are now simply the CQsin p').

Property (2) is shown similarly. In fact, the algorithm exploits the same ideas as the algorithm used to prove property (1). It is sketched in Algorithm 3.

Correctness and runtime follow immediately from the proof of property (1). □

Notice the stark contrast with the problem of checking whether a pWDPT p is in $\mathcal{M}(\mathcal{C})$ for $\mathcal{C} = g\text{-}\mathsf{TW}(k)$ or $\mathcal{C} = g\text{-}\mathsf{HW}'(k)$, for which we only obtained a NExpTime and NExpTime$^{\text{NP}}$ upper bound, respectively, in Theorem 6.2.

Evaluation for UpWDPTs in $\mathcal{M}(\bigcup\text{-}(g\text{-}\mathsf{TW}(k)))/\mathcal{M}(\bigcup\text{-}(g\text{-}\mathsf{HW}'(k)))$. Analogously to the partial and maximal evaluation problems, for a class $\mathcal{C}$ of UpWDPTs, we overload p-Partial-Eval($\mathcal{C}$) and p-Max-Eval($\mathcal{C}$) to be the parameterized variants of Partial-Eval($\mathcal{C}$) and Max-Eval($\mathcal{C}$),

ALGORITHM 3: For $\rho \in \mathcal{M}(\bigcup\text{-}C)$, return a UpWDPT $\rho' \in \bigcup\text{-}C$(for $C = g\text{-}\mathsf{TW}(k)$ or $g\text{-}\mathsf{HW}'(k)$)

```
1:  $\rho' := \emptyset$
2:  for all $q \in \rho_{\mathrm{CQ}}$ do
3:      Initialize $\bar{q}$ with null
4:      if there is no $q' \in \rho_{\mathrm{CQ}}$ with $q' \neq q$ and $q \sqsubset q'$ then
5:          for all mappings $\mu\colon \mathrm{var}(q) \to \mathrm{var}(q)$ do
6:              if $\mu(q) \equiv q$ and $\mu(q) \in \mathsf{TW}(k)$ then          ▷ respectively $\mu(q) \in \mathsf{HW}'(k)$
7:                  if $\bar{q}$ is null or $\bar{q} \sqsubseteq \mu(q)$ then
8:                      $\bar{q} := \mu(q)$
9:          if $\bar{q}$ is null then return NO
10:         else $\rho' := \rho' \cup \{\bar{q}\}$
11: return $\rho'$
```

respectively, where we choose the size of the UpWDPT as the parameter. Then, analogously to the case of Corollary 6.6, it follows from Theorem 7.5 that the maximal and partial evaluation problems for queries in $\mathcal{M}(\bigcup\text{-}(g\text{-}\mathsf{TW}(k)))$ and $\mathcal{M}(\bigcup\text{-}(g\text{-}\mathsf{HW}'(k)))$ are fixed-parameter tractable.

Corollary 7.6. *Let $k \geq 1$. The problems* p-Partial-Eval*($\mathcal{M}(\bigcup\text{-}(g\text{-}\mathsf{TW}(k)))$) and* p-Max-Eval*($\mathcal{M}(\bigcup\text{-}(g\text{-}\mathsf{TW}(k)))$) are fixed-parameter tractable. The result also holds for $g\text{-}\mathsf{HW}'(k)$ instead of $g\text{-}\mathsf{TW}(k)$.*

$\bigcup\text{-}g\text{-}\mathsf{TW}(k)/\bigcup\text{-}g\text{-}\mathsf{HW}'(k)$-approximations. As in Section 6.2, we study approximations for UpWDPTs without constants. Fix $k \geq 1$. Let ρ, ρ' be UpWDPTs such that $\rho' \in \bigcup\text{-}C$ for $C = g\text{-}\mathsf{TW}(k)$ or $C = g\text{-}\mathsf{HW}'(k)$. Analogously to Definition 6.7, we say that ρ' is a $\bigcup\text{-}C$-approximation of ρ if (1) $\rho' \sqsubseteq \rho$, and (2) there is no UpWDPT $\rho'' \in \bigcup\text{-}C$ such that $\rho' \sqsubset \rho'' \sqsubseteq \rho$.

For a UpWDPT ρ, let $\rho_{\mathrm{CQ-app}}$ denote the union of all $\mathsf{TW}(k)$-approximations (or $\mathsf{HW}'(k)$-approximations, respectively) of the CQs in ρ_{CQ}. We present below two results regarding $\rho_{\mathrm{CQ-app}}$ that allow us to develop a theory of approximations for UpWDPTs.

Proposition 7.7. *The set $\rho_{\mathrm{CQ-app}}$ always exists, can be computed in single-exponential time from ρ_{CQ}, and each CQ in $\rho_{\mathrm{CQ-app}}$ is of polynomial size.*

Proof. Each CQ has at least one, and at most exponentially many $\mathsf{TW}(k)$ or $\mathsf{HW}'(k)$-approximations, each one of which it is of polynomial size. Moreover, the set of all such approximations can be computed in exponential time. For a proof of these statements, see, e.g., Theorem 17 in Barceló et al. [8] for the case of $\mathsf{HW}(k)$; analogous proofs can be given for $\mathsf{TW}(k)$ and $\mathsf{HW}'(k)$. □

For sets S, S' of CQs we write $S \subseteq S'$ if for each CQ $q \in S$ there exists a CQ $q' \in S'$ such that $q \subseteq q'$.

Proposition 7.8. *The following statements hold:*

(1) $\rho_{\mathrm{CQ-app}} \subseteq \rho_{\mathrm{CQ}}$.
(2) *There is no set S of CQs in $\mathsf{TW}(k)$ (respectively, $\mathsf{HW}'(k)$) such that $\rho_{\mathrm{CQ-app}} \subsetneq S \subseteq \rho_{\mathrm{CQ}}$.*
(3) *For every set S of CQs in $\mathsf{TW}(k)$ (respectively, $\mathsf{HW}'(k)$) such that $S \subseteq \rho_{\mathrm{CQ}}$, it is also the case that $S \subseteq \rho_{\mathrm{CQ-app}}$. This means that, up to equivalence, $\rho_{\mathrm{CQ-app}}$ is the only set of CQs in $\mathsf{TW}(k)$ (respectively, $\mathsf{HW}'(k)$) that satisfies (1) and (2).*

Proof. For (1), each CQ in $\rho_{\mathrm{CQ-app}}$ is an approximation and thus contained in some CQ in ρ_{CQ}.

For (2), assume for the sake of contradiction that such an S exists. Since $\rho_{\mathrm{CQ-app}} \subsetneq S$, there exists a CQ $q \in S$ such that there is no $q' \in \rho_{\mathrm{CQ-app}}$ for which $q \subseteq q'$. Also, since $S \subseteq \rho_{\mathrm{CQ}}$, there exists a CQ $q^* \in \rho_{\mathrm{CQ}}$ such that $q \subseteq q^*$. But $q \in \mathsf{TW}(k)$ (respectively, $\mathsf{HW}'(k)$), and, therefore, there is a $\mathsf{TW}(k)$-approximation (respectively, $\mathsf{HW}'(k)$-approximation) q_1 of q^* such that $q \subseteq q_1 \subseteq q^*$ (see, e.g., the proof Theorem 17 in Barceló et al. [8] for the case of $\mathsf{HW}(k)$; analogous proofs can be given for $\mathsf{TW}(k)$ and $\mathsf{HW}'(k)$). It follows that there is no $q' \in \rho_{\mathrm{CQ-app}}$ for which $q_1 \subseteq q'$ holds. This contradicts the fact that $\rho_{\mathrm{CQ-app}}$ contains all $\mathsf{TW}(k)$-approximations (respectively, $\mathsf{HW}'(k)$-approximations) of q^*.

For (3), since $S \subseteq \rho_{\mathrm{CQ}}$, for each CQ $q \in S$ there exists a CQ $q' \in \rho_{\mathrm{CQ}}$ such that $q \subseteq q'$. But $q \in \mathsf{TW}(k)$ (respectively, $\mathsf{HW}'(k)$), and, therefore, there is a $\mathsf{TW}(k)$-approximation (respectively, $\mathsf{HW}'(k)$-approximation) q_1 of q' such that $q \subseteq q_1 \subseteq q'$. The CQ q_1 is in $\rho_{\mathrm{CQ-app}}$ by definition. It follows that $S \subseteq \rho_{\mathrm{CQ-app}}$. □

With these results, we can prove that $\bigcup$-C-approximations always exist and can be computed in exponential time, and that approximations are unique up to $\equiv_s$-equivalence and consist of (possibly exponentially many) pWDPTs of polynomial size (actually, these pWDPTs are even CQs):

THEOREM 7.9. *Let ρ be a UpWDPT, and let $C = g$-$\mathsf{TW}(k)$ or $C = g$-$\mathsf{HW}'(k)$. Then $\rho_{\mathrm{CQ-app}}$ is the unique $\bigcup$-C-approximation of ρ (up to $\equiv_s$-equivalence).*

PROOF. We show that $\rho_{\mathrm{CQ-app}}$ is a $\bigcup$-C-approximation of ρ. Assume to the contrary that this is not the case. Then there exists some union of pWDPTs ρ' such that $\rho_{\mathrm{CQ-app}} \sqsubset \rho' \sqsubseteq \rho$. Thus, consider ρ'_{CQ}. We have $\rho'_{\mathrm{CQ}} \equiv_s \rho'$. Thus also $\rho_{\mathrm{CQ-app}} \sqsubset \rho'_{\mathrm{CQ}} \sqsubseteq \rho \equiv_s \rho_{\mathrm{CQ}}$. However, this contradicts item (2) from Proposition 7.8. Therefore, such a ρ' cannot exist, which proves the claim. The uniqueness is shown similarly using item (3) from Proposition 7.8. □

Since the UpWDPT $\rho_{\mathrm{CQ-app}}$ can be computed in EXPTIME, we obtain:

COROLLARY 7.10. *Let $C = g$-$\mathsf{TW}(k)$ or $C = g$-$\mathsf{HW}'(k)$. There is an EXPTIME algorithm that, given a UpWDPT ρ, constructs a union ρ' of (possibly exponentially many) pWDPTs in C such that (1) each pWDPT in ρ' is of polynomial size and consists of a single node, and (2) ρ' is the unique $\bigcup$-C-approximation of ρ.*

For $C = g$-$\mathsf{TW}(k)$ or $C = g$-$\mathsf{HW}'(k)$, these techniques also allow us to find reasonable bounds for checking if ρ' is a $\bigcup$-C-approximation of ρ. This problem is called $\bigcup$-C-APPROXIMATION. Different from the question whether a pWDPT is in $\mathcal{M}(\bigcup$-$C)$, for this problem we have a tight complexity bound.

PROPOSITION 7.11. *Both $\bigcup$-(g-$\mathsf{TW}(k)$)-APPROXIMATION and $\bigcup$-(g-$\mathsf{HW}'(k)$)-APPROXIMATION are Π_2^P-complete for each $k \geq 1$.*

This is again in stark contrast to the problem of checking whether a pWDPT p' is a $\mathsf{TW}(k)$-approximation of p, for which we could only obtain a CONEXPTIME upper bound in Proposition 6.9.

8 CONCLUSION AND FUTURE WORK

We have extended the search for efficient query evaluation and query analysis from CQs to pWDPTs. It has turned out that additional restrictions are required to ensure tractability of query evaluation of pWDPTs. In Table 1, we give an overview of our results in terms of classical complexity. The five rows refer to the five problems EVAL(C), PARTIAL-EVAL(C), MAX-EVAL(C), SUBS (C_1, C_2), and SUBS-EQUIV (C_1, C_2), while the four columns refer to different choices of the classes C or C_1 and C_2. The results marked with references follow from previous works. Results marked with (ℓ-Q) in the general case refer to intractability results that already hold for the restricted case ℓ-Q. Results marked with (g-Q) in the case of ℓ-$Q \cap \mathsf{BI}(c)$ refer to tractability results that already hold for the more general case g-Q and, thus, need no separate proof. For the intractable

Table 1. Complexity of pWDPT Evaluation and Query Analysis (All Entries Denote Completeness) for $\mathcal{Q}$ = TW(k) and $\mathcal{Q}$ = HW(k)

	general	l-$\mathcal{Q}$	g-$\mathcal{Q}$	l-$\mathcal{Q} \cap$ BI(c)
Eval	Σ_2^P [31]	NP [31]	NP	LogCFL
p-Eval	NP(l-$\mathcal{Q}$)	NP [31]	LogCFL	LogCFL(g-$\mathcal{Q}$)
M-Eval	DP(l-$\mathcal{Q}$)	DP	LogCFL	LogCFL(g-$\mathcal{Q}$)
Subs	Π_2^P (l-$\mathcal{Q}$)	Π_2^P	coNP	coNP
Subs-Equiv	Π_2^P (l-$\mathcal{Q}$)	Π_2^P	coNP	coNP

Table 2. Reformulations in Tractable Classes of pWDPTs: Lower and Upper Bounds of the Complexity

	Lower Bound	Upper Bound
(g-HW′(k))-Membership	Π_2^P	NExpTime$^{\mathrm{NP}}$
(g-HW′(k))-Approximation	Π_2^P	coNExpTime$^{\mathrm{NP}}$
$\bigcup$-(g-HW′(k))-Membership	Π_2^P	Π_3^P
$\bigcup$-(g-HW′(k))-Approximation	Π_2^P	Π_2^P

The membership problem asks, given a (U)pWDPT, whether there exists an $\equiv_s$-equivalent one in g-HW′(k); approximation asks, given two (U)pWDPTs, Whether one is an g-HW′(k)-approximation of the other.

cases of the Eval($\mathcal{C}$) problem, we have also carried out a parameterized complexity analysis. In particular, p-Eval($\mathcal{C}$) is in FPT for $\mathcal{C} = g$-TW(k) $\cap$ SBI(c) or $\mathcal{C} = g$-HW(k) $\cap$ SBI(c), W[1]-complete for $\mathcal{C} = \ell$-TW(k) $\cap$ SBI(c) or $\mathcal{C} = \ell$-HW(k) $\cap$ SBI(c), and W[2]-hard for $\mathcal{C} = g$-TW(k) or $\mathcal{C} = g$-HW(k).

We have then applied our tractable classes of query evaluation to study static query analysis and to initiate a theory of approximation of pWDPTs. To this end, we have studied the classes $\mathcal{C}$ = g-TW(k) and $\mathcal{C}$ = g-HW′(k) and $\bigcup$-$\mathcal{C}$ of (unions) of "well-behaved" queries. Above all, we have managed to prove fixed-parameter tractability of query evaluation for (unions of) pWDPTs that are $\equiv_s$-equivalent to a query in $\mathcal{C}$ or $\bigcup$-$\mathcal{C}$, respectively. Further problems studied in this context are, given a (U)pWDPT, the existence of an $\equiv_s$-equivalent, "well-behaved" (U)pWDPT, and the question whether a given (U)pWDPT is an approximation of some other given (U)pWDPT. Preliminary complexity results for these tasks are displayed in Table 2. For the upper bounds, in the case of g-TW(k) instead of g-HW′(k), one NP-oracle is not needed and Π_3^P drops to Π_2^P.

Several lines of future work should be pursued. As far as query evaluation and query analysis are concerned, we have yet to identify a natural fragment of pWDPTs that guarantees tractable subsumption and subsumption-equivalence. Toward a theory of reformulations and approximations of pWDPTs, we have only made the first steps here. A better understanding of the nature of $\mathcal{C}$- and $\bigcup$-$\mathcal{C}$-approximations is needed to close the gaps in Table 2. For instance, we conjecture that there always exists some approximation of polynomial size and that the complexity of $\mathcal{C}$-Approximation drops to the polynomial hierarchy. The situation of pWDPTs is much more involved than for CQs, where the analogous problems come down to simple containment tests.

It would be very interesting as well to extend this analysis to the class of *weakly pWDPTs* introduced by Kaminski and Kostylev [29]. This is relevant since such a class of queries covers almost all the {AND,OPTIONAL}-SPARQL queries encountered in practice. We would also like to study efficient evaluation for more expressive fragments of the SPARQL language, such as the ones that include filters or path operators.

REFERENCES

[1] Isolde Adler, Georg Gottlob, and Martin Grohe. 2007. Hypertree width and related hypergraph invariants. *Eur. J. Comb.* 28, 8 (2007), 2167–2181.

[2] Shqiponja Ahmetaj, Wolfgang Fischl, Reinhard Pichler, Mantas Simkus, and Sebastian Skritek. 2015. Towards reconciling SPARQL and certain answers. In *Proceedings of the 24th International Conference on World Wide Web (WWW'15).* ACM, 23–33.

[3] Marcelo Arenas, Gonzalo I. Diaz, and Egor V. Kostylev. 2016. Reverse engineering SPARQL queries. In *Proceedings of the 25th International Conference on World Wide Web (WWW'16).* ACM, 239–249.

[4] Marcelo Arenas and Jorge Pérez. 2011. Querying semantic web data with SPARQL. In *Proceedings of the 30th ACM Symposium on Principles of Database Systems (PODS'11).* ACM, 305–316.

[5] Marcelo Arenas and Martín Ugarte. 2016. Designing a query language for RDF: Marrying open and closed worlds. In *Proceedings of the 35th ACM Symposium on Principles of Database Systems (PODS'16).* ACM, 225–236.

[6] Pablo Barceló, Leonid Libkin, and Miguel Romero. 2014. Efficient approximations of conjunctive queries. *SIAM J. Comput.* 43, 3 (2014), 1085–1130.

[7] Pablo Barceló, Reinhard Pichler, and Sebastian Skritek. 2015. Efficient evaluation and approximation of well-designed pattern trees. In *Proceedings of the 34th ACM Symposium on Principles of Database Systems (PODS'15).* ACM, 131–144.

[8] Pablo Barceló, Andreas Pieris, and Miguel Romero. 2017. Semantic optimization in tractable classes of conjunctive queries. *SIGMOD Record* 46, 2 (2017), 5–17.

[9] Pablo Barceló, Miguel Romero, and Moshe Y. Vardi. 2014. Does query evaluation tractability help query containment? In *Proceedings of the 33rd ACM Symposium on Principles of Database Systems (PODS'14).* ACM, 188–199.

[10] Pablo Barceló, Miguel Romero, and Moshe Y. Vardi. 2016. Semantic acyclicity on graph databases. *SIAM J. Comput.* 45, 4 (2016), 1339–1376.

[11] Angela Bonifati, Wim Martens, and Thomas Timm. 2017. An analytical study of large SPARQL query logs. *PVLDB* 11, 2 (2017), 149–161.

[12] Andrei A. Bulatov, Andrei A. Krokhin, and Benoit Larose. 2008. Dualities for constraint satisfaction problems. In *Complexity of Constraints - An Overview of Current Research Themes*, Lecture Notes in Computer Science, Vol. 5250. Springer, 93–124.

[13] Ashok K. Chandra and Philip M. Merlin. 1977. Optimal implementation of conjunctive queries in relational data bases. In *Proceedings of the 9th Annual ACM Symposium on Theory of Computing.* ACM, 77–90.

[14] Chandra Chekuri and Anand Rajaraman. 2000. Conjunctive query containment revisited. *Theor. Comput. Sci.* 239, 2 (2000), 211–229.

[15] Stephen A. Cook and Pierre McKenzie. 1987. Problems complete for deterministic logarithmic space. *J. Algorithms* 8, 3 (1987), 385–394.

[16] Víctor Dalmau, Phokion G. Kolaitis, and Moshe Y. Vardi. 2002. Constraint satisfaction, bounded treewidth, and finite-variable logics. In *Proceedings of the 8th International Conference on Principles and Practice of Constraint Programming (CP'02)*, Lecture Notes in Computer Science, Vol. 2470. Springer, 310–326.

[17] Rodney G. Downey and Michael R. Fellows. 1999. *Parameterized Complexity.* Springer, Berlin.

[18] Ronald Fagin. 1983. Degrees of acyclicity for hypergraphs and relational database schemes. *J. ACM* 30, 3 (1983), 514–550.

[19] Wolfgang Fischl, Georg Gottlob, and Reinhard Pichler. 2016. General and fractional hypertree decompositions: Hard and easy cases. *CoRR* abs/1611.01090 (2016).

[20] Jörg Flum and Martin Grohe. 2010. Parameterized Complexity Theory. *Texts in Theoretical Computer Science. An EATCS Series.* Springer, 1–495.

[21] Georg Gottlob, Gianluigi Greco, Nicola Leone, and Francesco Scarcello. 2016. Hypertree decompositions: Questions and answers. In *Proceedings of the 35th ACM Symposium on Principles of Database Systems (PODS'16).* PODS, 57–74.

[22] Georg Gottlob, Nicola Leone, and Francesco Scarcello. 2001. The complexity of acyclic conjunctive queries. *J. ACM* 48, 3 (2001), 431–498.

[23] Georg Gottlob, Nicola Leone, and Francesco Scarcello. 2002. Hypertree decompositions and tractable queries. *J. Comput. Syst. Sci.* 64, 3 (2002), 579–627.

[24] Georg Gottlob, Zoltán Miklós, and Thomas Schwentick. 2009. Generalized hypertree decompositions: NP-hardness and tractable variants. *J. ACM* 56, 6 (2009), 30:1–30:32.

[25] Georg Gottlob and Reinhard Pichler. 2004. Hypergraphs in model checking: Acyclicity and hypertree-width versus clique-width. *SIAM J. Comput.* 33, 2 (2004), 351–378.

[26] Martin Grohe. 2007. The complexity of homomorphism and constraint satisfaction problems seen from the other side. *J. ACM* 54, 1 (2007), 1:1–1:24.

[27] Martin Grohe and Dániel Marx. 2014. Constraint solving via fractional edge covers. *ACM Trans. Algorithms* 11, 1 (2014), 4:1–4:20.

[28] Martin Grohe, Thomas Schwentick, and Luc Segoufin. 2001. When is the evaluation of conjunctive queries tractable? In *Proceedings of the 33rd Annual ACM Symposium on Theory of Computing*. ACM, 657–666.

[29] Mark Kaminski and Egor V. Kostylev. 2016. Beyond well-designed SPARQL. In *Proceedings of the 19th International Conference on Database Theory (ICDT'16) (LIPIcs)*, Vol. 48. Schloss Dagstuhl—Leibniz-Zentrum fuer Informatik, 5:1–5:18.

[30] Markus Kröll, Reinhard Pichler, and Sebastian Skritek. 2016. On the complexity of enumerating the answers to well-designed pattern trees. In *Proceedings of the19th International Conference on Database Theory (ICDT'16) (LIPIcs)*, Vol. 48. 22:1–22:18.

[31] Andrés Letelier, Jorge Pérez, Reinhard Pichler, and Sebastian Skritek. 2013. Static analysis and optimization of semantic web queries. *ACM Trans. Database Syst.* 38, 4 (2013), 25:1–25:45.

[32] Dániel Marx. 2013. Tractable hypergraph properties for constraint satisfaction and conjunctive queries. *J. ACM* 60, 6 (2013), 42:1–42:51.

[33] MongoDB. 2017. MongoDB. Retrieved August 07, 2018 from https://www.mongodb.com/.

[34] Neo4j. 2017. Cypher - Graph Query Language. Retrieved August 07, 2018 from http://neo4j.com/developer/cypher-query-language/.

[35] Christos H. Papadimitriou and Mihalis Yannakakis. 1999. On the complexity of database queries. *J. Comput. Syst. Sci.* 58, 3 (1999), 407–427.

[36] Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. 2009. Semantics and complexity of SPARQL. *ACM Trans. Database Syst.* 34, 3 (2009), 16:1–16:45.

[37] François Picalausa and Stijn Vansummeren. 2011. What are real SPARQL queries like? In *Proceedings of the International Workshop on Semantic Web Information Management (SWIM'11)*. ACM, 7.

[38] Reinhard Pichler and Sebastian Skritek. 2014. Containment and equivalence of well-designed SPARQL. In *Proceedings of the 33rd ACM Symposium on Principles of Database Systems (PODS'14)*. ACM, 39–50.

[39] Eric Prud'hommeaux and Andy Seaborne. 2008. *SPARQL Query Language for RDF*. W3C Recommendation. W3C. Retrieved August 07, 2018 from http://www.w3.org/TR/rdf-sparql-query.

[40] Miguel Romero. 2018. The tractability frontier of well-designed SPARQL queries. In *Proceedings of the 37th ACM Symposium on Principles of Database Systems (PODS'18)*. ACM, 295–306.

[41] Ivan Hal Sudborough. 1977. Time and tape bounded auxiliary pushdown automata. In *Proceedings of the 6th Symposium on Mathematical Foundations of Computer Science (MFCS'77)*, Vol. 53. Springer, 493–503.

[42] Ivan Hal Sudborough. 1978. On the tape complexity of deterministic context-free languages. *J. ACM* 25, 3 (1978), 405–414.

[43] Mihalis Yannakakis. 1981. Algorithms for acyclic database schemes. In *Proceedings of the 7th International Conference on Very Large Data Bases VLDB 1981*. IEEE Computer Society, 82–94.

[44] Xiaowang Zhang, Jan Van den Bussche, and François Picalausa. 2016. On the satisfiability problem for SPARQL patterns. *J. Artif. Intell. Res.* 56 (2016), 403–428.

Received December 2016; revised April 2018; accepted June 2018