Characterizing Tractability of Simple Well-Designed Pattern Trees with Projection

Stefan Mengel
CNRS, CRIL UMR 8188, Lens, France
mengel@cril.fr

Sebastian Skritek
Faculty of Informatics, TU Wien, Vienna, Austria
skritek@dbai.tuwien.ac.at

Abstract

We study the complexity of evaluating well-designed pattern trees, a query language extending conjunctive queries with the possibility to define parts of the query to be optional. This possibility of optional parts is important for obtaining meaningful results over incomplete data sources as it is common in semantic web settings.

Recently, a structural characterization of the classes of well-designed pattern trees that can be evaluated in polynomial time was shown. However, projection – a central feature of many query languages – was not considered in this study. We work towards closing this gap by giving a characterization of all tractable classes of simple well-designed pattern trees with projection (under some common complexity theoretic assumptions). Since well-designed pattern trees correspond to the fragment of well-designed {AND, OPTIONAL}-SPARQL queries this gives a complete description of the tractable classes of queries with projections in this fragment that can be characterized by the underlying graph structures of the queries.

2012 ACM Subject Classification Theory of computation → Database query languages (principles); Theory of computation → Database query processing and optimization (theory)

Keywords and phrases SPARQL, well-designed pattern trees, query evaluation, FPT, characterizing tractable classes

Digital Object Identifier 10.4230/LIPIcs.ICDT.2019.20

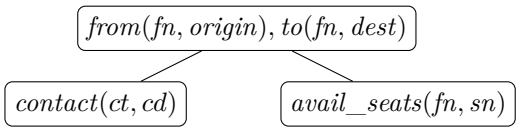
Related Version An extended version of the paper is available at `https://arxiv.org/abs/1712.08939`.

Funding *Sebastian Skritek*: Supported by the Austrian Science Fund (FWF): P30930-N35.

1 Introduction

Well-designed pattern trees (wdPTs) are a query formalism well-suited to deal with the ever increasing amount of incomplete data. Well-designed pattern trees over SPARQL triple patterns are equivalent to the class of well-designed {AND, OPTIONAL}-SPARQL queries [19] and were in fact originally introduced as a formalism to more easily study SPARQL queries. By replacing triple patterns with relational atoms, wdPTs can also be seen as an extension of Conjunctive Queries (CQs): a wdPT is a rooted tree where each node represents a conjunction of atoms, and the tree structure represents a nesting of optional matching. The idea is to start evaluating the CQ at the root and to iteratively extend the retrieved results as much as possible by the results of the CQs in the other nodes. This allows wdPTs to return partial answers in case that not the complete query can be mapped into the database – unlike CQs which in such a situation return no answer.

Well-designed pattern trees and the corresponding SPARQL fragment represent an important class of SPARQL queries and have been studied intensively within the last decade [19, 16, 2, 21, 20, 14, 3, 1, 22]. Thus many properties of and problems related to these



$\mathbf{D} = \{from(1, \text{“LHR”}), to(1, \text{“LIS”}),$
$from(2, \text{“LHR”}), to(2, \text{“LIS”}),$
$contact(email, \text{“e@ma.il”}),$
$avail\_seats(2, \text{“1A”})\}$

Figure 1 The wdPT p and database $\mathbf{D}$ from Example 1.

queries are now well understood. For example, the evaluation problem for wdPTs (i.e., given a wdPT, a database and a mapping, is this mapping an answer to the wdPT over the database) is coNP-complete for projection free wdPTs [19] and Σ_2P-complete in the presence of projection [16]. However, certain tractable classes of wdPTs have been identified [3]. The main idea there is to extend known tractability conditions for CQs to wdPTs. However, the question of characterizing exactly the classes of wdPTs for which tractable query evaluation is possible – and thus the question how good the approach of extending tractability conditions of CQs to wdPTs is – has been largely ignored. Only very recently, this question was addressed for wdPTs without projection, and a characterization of the classes for which query evaluation is in PTIME was given [22]. Notably, as also observed for Boolean Conjunctive Queries [12, 11], for wdPTs without projection these classes coincide with the ones for which evaluation is in FPT.

However, [22] does not touch projection, an essential and central feature of query languages. Thus the question “What are all tractable classes of wdPTs with projection?” remains open. We work towards closing this gap.

One observation consistently made in all the aforementioned work on wdPTs is that problems become much more complex once projection is included. This is true for both, the computational complexity of the problems (e.g., as mentioned, for the evaluation problem it increases from coNP- to Σ_2P-completeness; for classical query containment, the NP-complete problem becomes even undecidable [21]) as well as for establishing these results.

This is because of the particular semantics of well-designed SPARQL with projection. For wdPTs *without* projection, given some database, the set of answers consist of all variable mappings such that there exists a subtree of the wdPT satisfying the following conditions: first, it must contain the root node of the tree. Second, the set of variables occurring in the subtree must be the same as the domain of the mapping. Third, the mapping must map each atom in the subtree into the database, and fourth, no extension of the mapping is allowed to map all atoms of any child node of any node in the subtree into the database. This is illustrated by the following example (a precise definition is given in Section 2).

▶ **Example 1.** Figure 1 shows a wdPT p with three nodes where the top node represents the root of the tree, having two child nodes as depicted. At its root node, the query is looking for information on flights: flight number, origin and destination. This information should be extended by some contact information (left child), and information on available seats on the flight (right child) in case any of this information is available. Observe that the two extensions are independent of each other. The equivalent SPARQL query (replacing relational atoms by triple patterns) would be

```
{{?fn from ?origin . ?fn to ?dest} OPTIONAL {?ct contact ?cd}}
                                        OPTIONAL {?fn avail_seats ?sn}
```

For the database instance $\mathbf{D}$ also shown in the figure, the mapping μ with $\mu(fn) = 1$, $\mu(origin) =$ “LHR”, $\mu(dest) =$ “LIS”, $\mu(ct) = email$, and $\mu(cd) =$ “e@ma.il” is an answer to p over $\mathbf{D}$. This is because of the subtree of p consisting of the root node and the left

child. It can be checked that it satisfies all four conditions. For the fourth condition, just observe that there exists no extension of μ that maps $avail\_seats(fn, sn)$ into $\mathbf{D}$. Because of the fourth condition, the mapping ν with $\nu(fn) = 2$, $\nu(origin) =$ "LHR", $\nu(dest) =$ "LIS", $\nu(ct) = email$, and $\nu(cd) =$ "e@ma.il" is no solution, because this mapping can be extended by $\nu(sn) =$ "1A" in a way that maps $avail\_seats(fn, sn)$ also into $\mathbf{D}$.

Thus, without projection, the only hard part in deciding whether some mapping is a solution is to check for the existence of an extension, since this basically includes a homomorphism test. However, for wdPTs *with* projection, a mapping is a solution if there exists an extension of this mapping to some subset of the existential variables in the tree, such that the extended mapping is a solution to the wdPT considered without projection.

▶ **Example 2.** Consider the wdPT from Example 1, but now assume that the flight number fn is an existential variable and thus not part of the output. Then μ with $\mu(origin) =$ "LHR", $\mu(dest) =$ "LIS", $\mu(ct) = email$, and $\mu(cd) =$ "e@ma.il" is a solution because of the extension $\mu(fn) = 1$. Observe that the extension $\mu(fn) = 2$ does not witness μ to be a solution, since, as already discussed before, this mapping is not maximal.

As a consequence, beside testing some mapping for maximality, as a second source of hardness, different mappings on the existential variables have to be taken into account.

Besides the already mentioned increased complexity of many problems, it was also observed that for wdPTs with projection it is no longer the case that the classes for which query evaluation is in PTIME and in FPT coincide [15]. Thus, in this setting, the choice of the tractability notion makes a difference when describing all tractable classes.

We choose to study the complexity of query evaluation in the model of parameterized complexity where, as usual, we take the size of the query as the parameter. As already argued in [18], this model allows for a more fine-grained analysis than the classical perspectives of data- and query complexity. In parameterized complexity, query answering is considered tractable, formally in FPT, if, after a preprocessing that only depends on the query, the actual evaluation can be done in polynomial time [9, 10]. Parameterized complexity has found many applications in the complexity of query evaluation problems, see e.g. [12, 11, 17, 4, 22].

In our efforts to better understand the tractability frontier for wdPTs, we provide a complete characterization of the tractable classes of *simple* wdPTs, i.e., wdPTs where no two atoms share the same relation symbol. Because of the relationship between wdPTs and well-designed {AND, OPTIONAL}-SPARQL queries, this immediately gives a complete description of the tractable classes of well-designed {AND, OPTIONAL}-SPARQL queries with projection that can be characterized by only considering the graph structures of the queries, similar e.g. to the work of [12, 4]. We note that the results showing the existence of classes of wdPTs for which the evaluation problem is NP-hard but in FPT can be easily extended to simple wdPTs. Moreover, our tractability criteria are not restricted to simple wdPTs. In fact, the same tractability criteria can also directly be applied to give tractable classes of non-simple *wdPTs*. However, in this case, there are classes of queries that do not satisfy our tractability criteria and are still tractable. Thus, the restriction to simple wdPTs is crucial for the lower bounds.

Summary of results and organization of the paper. We study the following decision problem: Given a wdPT, a database, and a mapping, is the mapping a solution of the wdPT over the database? This is the standard formulation of the evaluation problem usually studied (cf. [16, 13, 22, 3]). It reveals the influence of the optional query parts on the evaluation problem, which is lost e.g. when considering Boolean queries. Instead of just SPARQL

triple patterns, we consider the more general case of wdPTs with arbitrary relational atoms where we always assume that the classes of queries we consider have bounded arity. Our main result is a characterization of the classes of simple wdPTs with projection that allow fixed-parameter tractable query evaluation.

After some preliminaries in Section 2, we define two tractability conditions in Section 3. By comparing these conditions with the tractability criterion from [22], we discuss how they describe the additional complexity introduced by projection. Note that some of the conditions provided here have precursors in [3] and [15] that had to be carefully refined to provide a fine-grained complexity analysis.

In Section 4 we prove that the two tractability conditions imply FPT membership of the evaluation problem by presenting an algorithm that exploits these conditions.

In Section 5 we then show that both tractability conditions are indeed necessary for a class of simple wdPTs to be tractable. That is, we show that if either of them is not satisfied by a class of wdPTs, the evaluation problem for this class is either W[1]- or coW[1]-hard.

In Section 6, we discuss our results and also potential extensions of the tractability conditions to conclude the paper.

2 Preliminaries

Basics. Let $\mathcal{C}onst$ and $\mathcal{V}ar$ be two disjoint countable infinite sets of constants and variables, respectively. A *relational schema* σ is a set $\{R_1, \ldots, R_n\}$ of relation symbols R_i, each having an assigned arity $r_i \geq 0$. A *relational atom* $R_i(\vec{v})$ over σ consists of a relation symbol $R_i \in \sigma$ and a tuple $\vec{v} \in (\mathcal{C}onst \cup \mathcal{V}ar)^{r_i}$. For an atom $\tau = R_i(\vec{v})$, let $\mathsf{dom}(\tau)$ denote the set of variables and constants occurring in τ. This extends to sets $\mathbf{R} = \{\tau_1, \ldots, \tau_m\}$ of atoms as $\mathsf{dom}(\mathbf{R}) = \bigcup_{i=1}^{m} \mathsf{dom}(\tau_i)$. Furthermore, $\mathsf{var}(\tau) = \mathsf{dom}(\tau) \cap \mathcal{V}ar$ and $\mathsf{var}(\mathbf{R}) = \mathsf{dom}(\mathbf{R}) \cap \mathcal{V}ar$. Observe that, by slight abuse of notation, we use operators $\cup, \cap, \setminus$ also between sets $\mathcal{V}$ and tuples $\vec{v}$ of variables and constants. For example, $\mathsf{var}(\tau) = \vec{v} \cap \mathcal{V}ar$. We call a set of atoms *simple* if no relation symbol appears more than once in it.

Similarly, for a mapping μ we denote with $\mathsf{dom}(\mu)$ the set of elements on which μ is defined. For a mapping μ and a set $\mathcal{V} \subseteq \mathcal{V}ar$, we use $\mu|_{\mathcal{V}}$ to describe the restriction of μ to the variables in $\mathsf{dom}(\mu) \cap \mathcal{V}$. We say that a mapping μ is an *extension* of a mapping ν if $\mu|_{\mathsf{dom}(\nu)} = \nu$, and that two mappings are *compatible* if they agree on the shared variables.

For a set $\mathbf{A}$ of atoms and a set $\mathcal{A} \subseteq \mathsf{dom}(\mathbf{A})$, we write $\mathbf{A} \setminus \mathcal{A}$ to denote the restriction of $\mathbf{A}$ to $\mathsf{dom}(\mathbf{A}) \setminus \mathcal{A}$. That is, we substitute every atom $R(\vec{v}) \in \mathbf{A}$ by an atom $R^s(\vec{v}')$, where $\vec{v}'$ is obtained from $\vec{v}$ by removing elements of $\mathcal{A}$, and s is the list of the removed positions.

A *database* $\mathbf{D}$ over σ is a finite set of atoms over σ with $\mathsf{var}(\mathbf{D}) = \emptyset$. For a database $\mathbf{D}$ and relation symbol R we denote by $R^{\mathbf{D}}$ the set of all atoms in $\mathbf{D}$ with relation symbol R.

Homomorphisms and Conjunctive Queries. A homomorphism h between two sets $\mathbf{A}$ and $\mathbf{B}$ of atoms over σ is a mapping $h\colon \mathsf{dom}(\mathbf{A}) \to \mathsf{dom}(\mathbf{B})$ such that for all atoms $R(\vec{v}) \in \mathbf{A}$ we have $R(h(\vec{v})) \in \mathbf{B}$, and such that $h(x) \neq x$ is only allowed if $x \in \mathsf{var}(\mathbf{A})$ (we thus restrict the definition of homomorphisms to $\mathsf{var}(\mathbf{A})$, the extension to constants via the identity mapping is implicit). We write $h\colon \mathbf{A} \to \mathbf{B}$ to denote a homomorphism h from $\mathbf{A}$ to $\mathbf{B}$.

We write CQs q as $Ans(\vec{x}) \leftarrow \mathbf{B}$, where the body $\mathbf{B} = \{R_1(\vec{v}_1), \ldots, R_m(\vec{v}_m)\}$ is a set of atoms and $\vec{x}$ are the *free variables*. A Boolean CQ (BCQ) is a CQ with no free variables. We define $\mathsf{var}(q) = \mathsf{var}(\mathbf{B})$. The existential variables are implicitly given by $\mathsf{var}(\mathbf{B}) \setminus \vec{x}$. The result $q(\mathbf{D})$ of q over a database $\mathbf{D}$ is the set of tuples $\{h(\vec{x}) \mid h\colon \mathbf{B} \to \mathbf{D}\}$.

Graphs. We consider only undirected, simple graphs $G = (V, E)$ with standard notations but sometimes write $t \in G$ to refer to a node $t \in V(G)$. A graph G_2 is a subgraph of a graph G_1 if $V(G_2) \subseteq V(G_1)$ and $E(G_2) \subseteq E(G_1)$. A tree is a connected, acyclic graph. A subtree is a connected, acyclic subgraph. A *rooted tree* T is a tree with one node $r \in T$ marked as its root. Given two nodes $t, \hat{t} \in T$, we say that $\hat{t}$ is an ancestor of t if $\hat{t}$ lies on the path from r to t. Likewise, $\hat{t}$ is the parent node of t (t is a child of $\hat{t}$) if $\hat{t}$ is an ancestor of t and $\{t, \hat{t}\} \in E(T)$. For a subtree T' of T, a node $t \in T$ is a child of T' if $t \notin T'$ and $\hat{t} \in T'$ for the parent node $\hat{t}$ of t. We write $ch(T')$ for the set of all children of T'. For a node $t \in T$ the set of nodes on the path from r to the parent node of t is denoted by $\mathsf{branch}(t)$. Moreover, $\mathsf{cbranch}(t) = \mathsf{branch}(t) \cup \{t\}$.

A *tree decomposition* of a graph $G = (V, E)$ is a pair (T, ν), where T is a tree and $\nu \colon V(T) \to 2^V$, that satisfies the following: (1) For each $u \in V$ the set $\{s \in V(T) \mid u \in \nu(s)\}$ is a connected subset of $V(T)$, and (2) each edge of E is contained in at least one of the sets $\nu(s)$, for $s \in V(T)$. The *width* of (T, ν) is $(\max\{|\nu(s)| \mid s \in V(T)\}) - 1$. The *treewidth* of G is the minimum width of its tree decompositions.

For a set $\mathbf{A}$ of atoms, the *Gaifman graph* of $\mathbf{A}$ is the graph $G = (V, E)$ with $V = \{v_i \mid v_i \in \mathsf{var}(\mathbf{A})\}$ and E contains an edge $\{v_i, v_j\}$ if v_i and v_j occur together in some atom in $\mathbf{A}$. The treewidth of a set of atoms is the treewidth of its Gaifman graph.

Well-designed pattern trees (wdPTs). A *pattern tree* (short: PT) p over a relational schema σ is a tuple $(T, \lambda, \mathcal{X})$ where T is a rooted tree and λ maps each node $t \in T$ to a set of relational atoms over σ. We may write $((T, r), \lambda, \mathcal{X})$ to emphasize that r is the root node of T. The set $\mathcal{X}$ of variables denotes the *free variables* of the PT. For a PT $(T, \lambda, \mathcal{X})$ and a subtree T' of T, let $\lambda(T') = \bigcup_{t \in V(T')} \lambda(t)$. We may write $\mathsf{var}(t)$ instead of $\mathsf{var}(\lambda(t))$, and $\mathsf{var}(T')$ instead of $\mathsf{var}(\lambda(T'))$. We further define $\mathsf{fvar}(t) = \mathsf{var}(t) \cap \mathcal{X}$ as the free variables in t. Again this definition extends naturally to subtrees T' of T. We call a PT $(T, \lambda, \mathcal{X})$ *projection free* if $\mathcal{X} = \mathsf{var}(T)$ and may write (T, λ) to emphasize a PT to be projection free. The size $|p|$ of a pattern tree is $\sum_{t \in V(T)} |\lambda(t)|$.

Well-designed PTs restrict the distribution of variables among their nodes.

▶ **Definition 3** (Well-Designed Pattern Tree (wdPT)). *A PT $(T, \lambda, \mathcal{X})$ is* well-designed *if for every variable $y \in \mathsf{var}(T)$, the set of nodes of T where y appears is connected.*

As an immediate consequence of this restriction, in a wdPT $p = (T, \lambda, \mathcal{X})$, for every variable $y \in \mathsf{var}(T)$ there exists a unique node $t \in T$ such that $y \in \mathsf{var}(t)$ and all nodes $t' \in T$ with $y \in \mathsf{var}(t')$ are descendants of t.

Evaluating a wdPT p with free variables $\mathcal{X}$ over a database $\mathbf{D}$ returns a set $p(\mathbf{D})$ of mappings $\mu \colon \mathcal{V} \to \mathsf{dom}(\mathbf{D})$ with $\mathcal{V} \subseteq \mathcal{X}$. We follow the characterization of $p(\mathbf{D})$ in terms of maximal subtrees [16], but borrow the term pp-solution from [13].

▶ **Definition 4** (pp-solution). *For a wdPT $p = ((T, r), \lambda)$ and a database $\mathbf{D}$, a mapping $\mu \colon \mathcal{V} \to \mathsf{dom}(\mathbf{D})$ (with $\mathcal{V} \subseteq \mathsf{var}(T)$) is a* potential partial solution (pp-solution) *to p over $\mathbf{D}$ if there is a subtree T' of T containing r such that $\mu \colon \lambda(T') \to \mathbf{D}$.*

The semantics of wdPTs can now be defined in terms of maximal pp-solutions.

▶ **Definition 5** (Semantics of wdPTs). *Let $p = (T, \lambda, \mathcal{X})$ be a wdPT, and let $p' = (T, \lambda, \mathsf{var}(T))$, i.e., the projection-free wdPT retrieved from p by considering all of its variables as free, and let $\mathbf{D}$ be a database. The set $p'(\mathbf{D})$ contains all pp-solutions μ to p' over $\mathbf{D}$ such that there exists no pp-solution μ' to p' over $\mathbf{D}$ that is a proper extension of μ.*

The set $p(\mathbf{D})$ is then defined as $p(\mathbf{D}) = \{\mu|_{\mathcal{X}} \mid \mu \in p'(\mathbf{D})\}$.

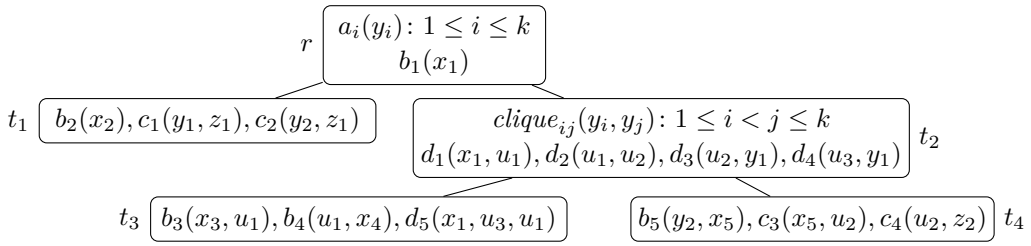


Figure 2 The well-designed pattern tree of Example 6.

▶ **Example 6.** Consider the PT $p = (T, \lambda, \mathcal{X})$ depicted in Figure 2, where k may be any integer with $k \geq 2$, and $\mathcal{X} = \{x_1, x_2, x_3, x_4, x_5\}$. All variable occurrences in p are connected, thus it is well-designed. If for example the atom $a_2(y_2)$ was missing in the root node, the tree would not be well-designed because of the occurrences of y_2 in both, t_1 and t_2.

Consider a database $\mathbf{D}$ that, for each atom $R(\vec{v})$ in $\lambda(T)$ contains one atom $R(1, \ldots, 1)$ (i.e., with the value 1 at each position) and in addition the atoms $d_1(1, 2)$, $d_2(2, 2)$, and $d_3(2, 1)$. Then $p(\mathbf{D}) = \{\mu_1, \mu_2\}$ where $\mathsf{dom}(\mu_1) = \{x_1, \ldots, x_5\}$, $\mathsf{dom}(\mu_2) = \{x_1, x_2\}$, and $\mu_i(x) = 1$ for $i \in \{1, 2\}$ and all $x \in \mathsf{dom}(\mu_i)$. This is because of the following extensions μ_1' and μ_2' of μ_1 and μ_2, respectively. For μ_1', we have $\mathsf{dom}(\mu_1') = \mathsf{var}(T)$ and $\mu_1'(x) = 1$ for all $x \in \mathsf{dom}(\mu_1')$, and for μ_2' we have $\mathsf{dom}(\mu_2') = \mathsf{var}(\lambda(\{r, t_1, t_2\}))$ with $\mu_2'(x) = 1$ for all $x \in \mathsf{dom}(\mu_2')$ except for u_1 and u_2, for which $\mu_2'(u_i) = 2$. Observe that μ_2' maps r, t_1, and t_2 into $\mathbf{D}$, but cannot be extended to neither t_3 nor t_4.

Parameterized complexity. We only give a bare-bones introduction to parameterized complexity and refer the reader to [8] for more details. Let Σ be a finite alphabet. A *parameterization* of Σ^* is a polynomial time computable mapping $\kappa\colon \Sigma^* \to \mathbb{N}$. A *parameterized problem* over Σ is a pair (L, κ) where $L \subseteq \Sigma^*$ and κ is a parameterization of Σ^*. We refer to $x \in \Sigma^*$ as the instances of a problem, and to the numbers $\kappa(x)$ as the parameters.

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

In this paper, for classes $\mathcal{P}$ of wdPTs, we study the problem p-Eval($\mathcal{P}$) defined below.

p-Eval($\mathcal{P}$)	
INSTANCE:	A wdPT $p \in \mathcal{P}$, a database $\mathbf{D}$, and a mapping μ.
PARAMETER:	$\lvert p \rvert$
QUESTION:	Does $\mu \in p(\mathbf{D})$ hold?

We always assume that the arity of all atoms of the queries in $\mathcal{P}$ is bounded by a constant, i.e., that there is a constant c (possibly depending on $\mathcal{P}$) such that no atom in the queries in $\mathcal{P}$ has an arity of more than c.

Let $E = (L, \kappa)$ and $E' = (L', \kappa')$ be parameterized problems. An FPT-*reduction* from E to E' is a mapping $R\colon \Sigma^* \to (\Sigma')^*$ such that (1) for all $x \in \Sigma^*$ we have $x \in L$ if and only if $R(x) \in L'$, (2) there is a computable function f and a polynomial *pol* such that $R(x)$ can be computed in time $f(\kappa(x)) \cdot pol(|x|)$, and (3) there is a computable function $g\colon \mathbb{N} \to \mathbb{N}$ such that $\kappa'(R(x)) \leq g(\kappa(x))$ for all $x \in \Sigma^*$.

Of the rich parameterized hardness theory, we will only use the classes $\mathsf{W}[1]$ and $\mathsf{coW}[1]$ of parameterized problems. To keep this introduction short, we define a parameterized problem

(L, κ) to be W[1]-hard if there is a W[1]-hard problem (L', κ') that FPT-reduces to (L, κ). We define (L, κ) to be coW[1]-hard if its complement is W[1]-hard. It is generally conjectured that FPT $\neq$ W[1] and thus in particular W[1]-hard problems and coW[1]-hard problems are not in FPT. We will take the hardness results for problems (L', κ') from the literature. One important such problem is the homomorphism problem p-HOM($\mathcal{C}$) for a class $\mathcal{C}$ of sets of atoms. Its input is one set $\mathbf{A} \in \mathcal{C}$ of atoms and another set $\mathbf{B}$ of atoms, and the question is whether there exists a homomorphism $h\colon \mathbf{A} \to \mathbf{B}$. The parameter is the size of $\mathbf{A}$.

▶ **Theorem 7** ([12]). *Let $\mathcal{C}$ be a decidable class of simple sets of atoms. Then* p-HOM($\mathcal{C}$) *is in* FPT *if there exists some constant c such that the treewidth of each set in $\mathcal{C}$ is bounded by c, and* W[1]*-hard otherwise.*

3 Tractability Conditions

In this section we will introduce our tractability criteria which we tailor towards simple wdPTs to simplify the presentation. While, as mentioned, our criteria also apply to arbitrary wdPTs, they are not optimal in this case. In fact, in the extended version we show generalizations of these conditions that, for general wdPTs, describe more tractable classes.

We start with the definition of what we consider as *simple* pattern trees. Basically, in a simple pattern tree, no relation symbol is allowed to occur more than once.

▶ **Definition 8** (Simple PTs). *A PT $p = (T, \lambda, \mathcal{X})$ over σ is a* simple pattern tree *if $\lambda(T)$ is simple and $\lambda(t) \cap \lambda(t') = \emptyset$ for all $t, t' \in T$ with $t \neq t'$.*

Our overall idea of solving p-Eval($\mathcal{P}$) is as follows: given a wdPT p, a database $\mathbf{D}$, and a mapping μ, construct a set of CQs q with free variables $\vec{x}$ and associated databases $\mathbf{D}'$ such that $\mu \in p(\mathbf{D})$ if and only if for at least one of these CQs q the tuple $\mu(\vec{x})$ is in $q(\mathbf{D}')$. We give two tractability criteria ensuring that the algorithm is in FPT. Intuitively, one condition guarantees that deciding $\mu(\vec{x}) \in q(\mathbf{D}')$ is in PTIME, while both conditions in combination guarantee that $\mathbf{D}'$ can be computed efficiently.

We will state the tractability conditions with respect to a class $\mathcal{P}$ of wdPTs. So in the remainder of this section let $\mathcal{P}$ be an arbitrary but fixed class of wdPTs.

We start with some additional notation and results. One effect introduced by projections is that some nodes of wdPTs may not be relevant for the results in the following sense as already observed in [16].

▶ **Definition 9** (Relevant Nodes). *Let $p = (T, \lambda, \mathcal{X})$ be a wdPT. A node $t \in T$ is* relevant *if there exists a database $\mathbf{D}$ such that $p(\mathbf{D}) \neq p'(\mathbf{D})$ where p' is constructed from p by removing from T the subtree rooted in t. We use* $\mathsf{relv}(T)$ *to denote the set of relevant nodes in T.*

In [16], the authors introduced a normal form excluding non-relevant nodes. Here, in order to make our results more explicit, we do not follow this approach but allow wdPTs to contain non-relevant nodes. Luckily, it follows from [16] that these nodes can be easily detected.

▶ **Proposition 10.** *Let $p = (T, \lambda, \mathcal{X})$ be a wdPT. Then a node $t \in T$ is* relevant *if and only if $\mathsf{fvar}(T') \setminus \mathsf{fvar}(\hat{t}) \neq \emptyset$, where T' is the subtree of T rooted in t and $\hat{t}$ is the parent node of t.*

In what is to follow, the variables that are shared between a node and its parent node will be of crucial importance. To this end, we make the following definition.

▶ **Definition 11** (Interface $\mathcal{I}(t)$ of a Node). *Let $(T, \lambda, \mathcal{X})$ be a wdPT, $t \in T$ (but not the root node), and $\hat{t}$ the parent node of t. The* interface $\mathcal{I}(t)$ *of t is the set $\mathcal{I}(t) = \mathsf{var}(t) \cap \mathsf{var}(\hat{t})$. The interface of the root node r is $\mathcal{I}(r) = \emptyset$.*

It was already remarked e.g. in [3] and [15] that restrictions on the number of variables shared between different sets of nodes can be used to define tractable classes. The above definition however differs slightly from the notion of interfaces in these works. E.g., in [15], the interface of a node describes the set of variables shared between the node and any of its neighbors, while here it is restricted to the variables shared with its parent node.

Restrictions on the size of node interfaces turn out to be quite coarse, and we provide more fine grained tractability criteria here. To this end, we recall the notion of an S-component from [7]: let $G = (V, E)$ be a graph, and $S \subseteq V$. Then let $\mathcal{K}$ be the set of connected components of $G[V \setminus S]$, and for each $K \in \mathcal{K}$, let $S_K \subseteq S$ be the set of nodes in S that have (in G) an edge to some node in K, i.e., $S_K = \{v \in S \mid \{v, v'\} \in E \text{ for some } v' \in K\}$. The set of S-components of G now is the set $\{G[K \cup S_K] \mid K \in \mathcal{K}\}$.

For a set S of variables, the notion of S-components extends to sets of atoms in the obvious way via the Gaifman graph. We will thus talk about S-components of sets of atoms.

▶ **Definition 12** ([15] Node Components). *Let $p = (T, \lambda, \mathcal{X})$ be a wdPT and $t \in T$. The set of* node components $\mathcal{NC}_t$ of t *is a set of set of atoms, defined as the union of:*

1. *The set $\{\{\tau\} \mid \tau \in \lambda(t)$ and $\mathsf{var}(\tau) \subseteq \mathcal{I}(t)\}$ consisting of singleton sets for every atom $\tau \in \lambda(t)$ which contains only "interface variables", i.e., variables from $\mathcal{I}(t)$.*
2. *The set of all $\mathcal{I}(t)$-components of $\lambda(t)$.*

In the following, node components of *type (1)* are those containing single atoms, while node components of *type (2)* are sets of possibly several atoms.

▶ **Example 13.** Recall the wdPT p from Example 6 and consider the node t_2. It contains the following node components: each atom $clique_{ij}(y_i, y_j)$ forms a node component of type (1) since all variables y_i occur also in r. In addition, there are two node components of type (2): the sets $\{d_1(x_1, u_1), d_2(u_1, u_2), d_3(u_2, y_1)\}$ and $\{d_4(u_3, y_1)\}$. Observe that while all atoms d_i $(1 \leq i \leq 4)$ are within the same connected component of the Gaifman graph, they are not in the same node component, since y_1 separates the connected component into two parts.

To understand why node components are essential for our results, recall that solutions to wdPTs must be maximal, i.e., they map some subtree into the database, but cannot be extended to map some child node of this subtree into the database as well. But such an extension to a node does not exist if and only if the mapping cannot be extended to one of the node components. Thus instead of testing extensions to the complete node at once (which might be intractable), we test the maximality of a mapping independently for each node component (which might be tractable). This is possible because for all variables shared between any two node components, the values are already determined by the mapping to be extended. Extensions to different node components are thus independent of each other.

For node components, we are in particular interested in the contained interface variables.

▶ **Definition 14** (Interface of a Node Component). *For a wdPT $(T, \lambda, \mathcal{X})$ and a node $t \in T$, the* interface of a node component $\mathbf{S} \in \mathcal{NC}_t$ *is $\mathcal{I}_t(\mathbf{S}) = (\mathcal{I}(t) \cap \mathsf{var}(\mathbf{S}))$, and the* existential *interface is $\mathcal{I}_t^{\exists}(\mathbf{S}) = \mathcal{I}_t(\mathbf{S}) \setminus \mathcal{X}$.*

We are now ready to formulate our first tractability condition.

Tractability condition (a): There is a constant c, such that for each $p = (T, \lambda, \mathcal{X}) \in \mathcal{P}$, the treewidth of $\mathbf{S} \setminus \mathcal{I}_t(\mathbf{S})$ is bounded by c for all $t \in \mathsf{relv}(T)$ (with $t \neq r$) and all $\mathbf{S} \in \mathcal{NC}_t$.

Intuitively, condition (a) guarantees that for each node component $\mathbf{S}$, given some mapping on the variables in its interface, one can decide in polynomial time whether this mapping can be extended to map all atoms in the node component into a given database. This is because once a mapping on $\mathcal{I}_t(\mathbf{S})$ is given, these variables can be treated as constants.

▶ **Example 15.** To demonstrate the situation after introducing condition (a), let p_k be the wdPT p from Example 6 parameterized by k, let $\mathcal{P} = \{p_k \mid k \geq 2\}$, and let $T' = T[\{r, t_1, t_2\}]$. Assume, for some $k \geq 2$, in order to test if some mapping is a solution to p_k, we would like to verify whether some mapping μ' with $\mathsf{dom}(\mu') = \mathsf{var}(T')$ is a maximal pp-solution. Deciding whether it is a pp-solution is easy, and because $\mathcal{P}$ satisfies tractability condition (a), testing if there exists in both, t_3 and t_4 a node component to which μ' cannot be extended is feasible in polynomial time as well. In fact, $\mathcal{NC}_{t_3} = \{\{b_3(x_3, u_1)\}, \{b_4(u_1, x_4)\}, \{d_5(x_1, u_3, u_1)\}\}$ and $\mathcal{NC}_{t_4} = \{\{b_5(y_2, x_5), c_3(x_5, u_2)\}, \{c_4(u_2, z_2)\}\}$ (this holds for every $p_k \in \mathcal{P}$). However, there may exist an exponential number of pp-solutions on T' (T' contains $k + 4$ existential variables). Thus testing one mapping on $\mathsf{var}(T')$ after the other is not feasible.

One way to overcome the problem sketched in the example is to not have two separate tests for being a pp-solution and being maximal. To combine these tests, we first compute for each node component the set of mappings on its interface variables that do not extend to the component, and then require pp-solutions to be consistent with these mappings.

It turns out that this idea can be encoded as an evaluation problem for CQs. One important step in this encoding is to introduce new relations, one for each node component, that store those mappings on the interface variables that cannot be extended to the component. In order for the resulting CQ evaluation problem to be in polynomial time, we require two properties. First, the resulting CQs must be from some tractable fragment of CQ evaluation, and, second, the size of the newly added relations must be at most polynomial. One way to achieve this second goal is to restrict the arity of these relations. Tractability for the CQ evaluation problem holds exactly if the class of resulting CQs is of bounded treewidth. As it turns out, this restriction also implies a bound on the arity of the new relations, and thus represents our second tractability condition.

To formalize the construction, we introduce the notion of a *component interface atom*. For a wdPT $(T, \lambda, \mathcal{X})$, a subtree T' of T, a node $t \in ch(T')$, and node component $\mathbf{S} \in \mathcal{NC}_t$, let the interface atom be an atom $R(\vec{v})$ where $\vec{v}$ contains the variables in $\mathcal{I}_t^{\exists}(\mathbf{S})$ and R is a fresh relation symbol. For a node component $\mathbf{S}$, we use $\mathsf{cia}(\mathbf{S})$ to refer to the corresponding interface atom $R(\vec{v})$. Observe that these definitions imply $\mathsf{cia}(\mathbf{S}) = R()$ in case $\mathcal{I}_t^{\exists}(\mathbf{S}) = \emptyset$.

The intuition for $\mathsf{cia}(\mathbf{S})$ is that for each node component, we get one atom that covers exactly the variables in $\mathcal{I}_t^{\exists}(\mathbf{S})$. The free variables in the interface can be excluded from the considerations since a fixed value is provided for them as part of the input.

▶ **Example 16.** Recall again the wdPT from Example 6 and consider the node t_1. It contains two node components: $\mathbf{S}_1 = \{b_2(x_2)\}$ and $\mathbf{S}_2 = \{c_1(y_1, z_1), c_2(y_2, z_1)\}$. Observe that whether $\mathbf{S}_1$ can be mapped into some database $\mathbf{D}$ is completely independent of the interface variables. Thus $\mathsf{cia}(\mathbf{S}_1) = R_{\mathbf{S}_1}()$. However, for $\mathbf{S}_2$ the values of y_1 and y_2 influence whether $\mathbf{S}_2$ may be mapped. Thus we get $\mathsf{cia}(\mathbf{S}_2) = R_{\mathbf{S}_2}(y_1, y_2)$. In case of the node component $\{d_5(x_1, u_3, u_1)\}$ of t_3, observe that we can assume some fixed value $\mu(x_1)$, and thus can reduce the atom $d_5(\mu(x_1), u_3, u_1)$ according to this value, and get as interface atom $d_5^{x_1=\mu(x_1)}(u_3, u_1)$.

Since we are looking for *one* pp-solution that cannot be extended to *any* child node, combining the two tests as sketched means that we must test all children simultaneously instead of individually. However, since each CQ tests only one node component for each child, we need one CQ for each possible combination, leading to our second tractability condition.

Tractability condition (b): There is a constant c such that for every well-designed pattern tree $p = ((T, r), \lambda, \mathcal{X}) \in \mathcal{P}$ and every subtree T' of T containing r, the treewidth of $(\lambda(T') \cup \bigcup_{i=1}^{n} \{\mathsf{cia}(\mathbf{S}_i)\}) \setminus \mathsf{fvar}(T')$ is bounded by c for every $(\mathbf{S}_1, \ldots, \mathbf{S}_n) \in \mathcal{NC}_{t_1} \times \cdots \times \mathcal{NC}_{t_n}$ where $\{t_1, \ldots, t_n\} = ch(T') \cap \mathsf{relv}(T)$.

The following example breaks down condition (b) to demonstrate its intuition.

▶ **Example 17.** Recall the setting in Example 15, as well as the database instance $\mathbf{D}$ and the mappings μ_2 and μ_2' from Example 6. Assume that, in order to show that $\mu_2 \in p_k(\mathbf{D})$ for any $k \geq 2$, we are looking for a pp-solution for T' that does not extend neither to the node component $\mathbf{S}_1 = \{b_3(x_3, u_1)\}$ of t_3 (and thus not to $\lambda(t_3)$), nor to the node component $\mathbf{S}_2 = \{b_5(y_2, x_5), c_3(x_5, u_2)\}$ of t_4 (and thus not to $\lambda(t_4)$). The idea is to construct a CQ $Ans(x_1, x_2) \leftarrow \lambda(T') \cup \{R_1(u_1), R_2(y_2, u_2)\}$, where $R_1(u_1)$ and $R_2(y_2, u_2)$ are the component interface atoms for $\mathbf{S}_1$ and $\mathbf{S}_2$, respectively. Observe that this query has exactly the structure described in condition (b), illustrating the motivation for the definition of this condition. Also, because of the $clique_{ij}$-atoms, $\mathcal{P}$ does not satisfy tractability condition (b).

The query is evaluated over the instance $\mathbf{D}$ extended by relations for R_1 and R_2. As mentioned, these relations contain all values that cannot be extended to map $\mathbf{S}_1$ and $\mathbf{S}_2$ into $\mathbf{D}$, respectively. For $\mathbf{S}_1$, we get $\{R_1(2)\}$ (since $b_3(1,1) \in \mathbf{D}$, thus a mapping assigning 1 to u_1 could be extended to map $\mathbf{S}_1$ into $\mathbf{D}$), and for $\mathbf{S}_2$ we get $\{R_2(1,2), R_2(2,1), R_2(2,2)\}$.

Now the mapping μ_2' witnesses the fact that $(1,1) \in q(\mathbf{D})$, and thus μ_2' is a maximal pp-solution also witnessing $\mu_2 \in p_k(\mathbf{D})$.

The main result of this paper is that the conditions (a) and (b) characterize exactly the classes of simple wdPTs which can be evaluated efficiently.

▶ **Theorem 18.** *Assume that* $\mathsf{FPT} \neq \mathsf{W}[1]$*, and let* $\mathcal{P}$ *be a decidable class of simple wdPTs of bounded arity. Then the following statements are equivalent.*

1. *The tractability conditions (a) and (b) hold for* $\mathcal{P}$*.*
2. p-Eval($\mathcal{P}$) *is in* FPT*.*

We will show the upper bound of Theorem 18 in Section 4, where we describe how the different ideas described so far can be combined to an FPT algorithm, while the lower bounds will be shown in Section 5.

But before we turn to the proof of Theorem 18, let us interpret the result in the setting without projections to better understand the influence of projection. First note that in that case, by Definition 14, we have $\mathcal{I}_t^{\exists}(\mathbf{S}) = \emptyset$ for every $t \in T$ and every $\mathbf{S} \in \mathcal{NC}_t$. Thus all atoms $\mathsf{cia}(\mathbf{S})$ (for any node component $\mathbf{S}$) are of arity 0 as are all atoms in $(\lambda(T') \cup \bigcup_{i=1}^{n} \{\mathsf{cia}(\mathbf{S}_i)\}) \setminus \mathsf{fvar}(T')$. Tractability condition (b) is therefore void in this setting, leaving only (a) as a useful condition in the projection free case. This immediately implies the following corollary.

▶ **Corollary 19.** *Assume that* $\mathsf{FPT} \neq \mathsf{W}[1]$*, and let* $\mathcal{P}$ *be a decidable class of simple wdPTs of bounded arity without projections. Then* p-Eval($\mathcal{P}$) *is in* FPT *if and only if tractability condition (a) holds for* $\mathcal{P}$*.*

We remark that Corollary 19 could also be inferred as a special case of the main result of [22]. Stating the corollary explicitly here lets us better understand the role of projection for our problem: in fact, the role of condition (a) is essentially to deal with the complexity that we already have without projection, while condition (b) is necessary to deal with the additional source of hardness that is introduced by projections and does not appear without them.

Since it will simplify the discussion in the upcoming sections, we conclude the section by explicitly working out the third tractability condition already mentioned above in the discussion towards tractability condition (b). As described there, at some point we extend a given database by relations for the atoms $\mathsf{cia}(\mathbf{S})$ that contain for the corresponding node component all mappings on its existential interface that cannot be extended to a mapping

Algorithm 1 EvalFPT($p = ((T, r), \lambda, \mathcal{X})$, $\mathbf{D}$, μ).

1: $T = T[\mathsf{relv}(T)]$ ▷ Remove all nodes from T that are not relevant
2: **for all** subtrees T' of T with $\mathsf{fvar}(T') = \mathsf{dom}(\mu)$ and $r \in T'$ **do**
3: Let $\{t_1, \ldots, t_n\} = ch(T')$
4: **for all** $(\mathbf{S}_1, \ldots, \mathbf{S}_n) \in \mathcal{NC}_{t_1} \times \cdots \times \mathcal{NC}_{t_n}$ **do**
5: $q = \text{“}Ans(\vec{x}) \leftarrow \lambda(T') \cup \{\mathsf{cia}(\mathbf{S}_1), \ldots, \mathsf{cia}(\mathbf{S}_n)\}\text{”}$ ▷ Let $\vec{x}$ contain all $x \in \mathsf{fvar}(T')$
6: $\mathbf{D}' = \mathbf{D} \cup \bigcup_{i=1}^{n} \{R_i(\nu(\vec{v}_i)) \mid \nu \in stop(\mathbf{S}_i, \mathbf{D})\}$ ▷ Assume $\mathsf{cia}(\mathbf{S}_i) = R_i(\vec{v}_i)$
7: **if** $\mu(\vec{x}) \in q(\mathbf{D}')$ **then** EXIT(YES)
8: EXIT(NO)

on the whole node component. To guarantee that all these relations are of polynomial size, we restrict the number of variables in the existential interfaces of the node components by some constant c. We formalize this notion in terms of a suitable width measure.

▶ **Definition 20** (Component Width). *Let $p = (T, \lambda, \mathcal{X})$ be a wdPT, $t \in T$, and $\mathbf{S} \in \mathcal{NC}_t$. The* width *of the node component $\mathbf{S}$ is $|\mathcal{I}_t^{\exists}(\mathbf{S})|$. For a node $t \in T$, the* component width *of t is the maximum width over all node components $\mathbf{S}$ of t. The component width of p is the maximum component width over all $t \in \mathsf{relv}(T)$.*

By the definition of the treewidth of a set of atoms – specifically by the fact that in the Gaifman graph all variables occurring together in an atom form a clique – and the fact that for the existential interface of each node component its variables occur together in some $\mathsf{cia}(\mathbf{S})$-atom, the number of variables in any existential interface is bound by the treewidth of the CQs defined in condition (b). Thus, we get the following corollary.

▶ **Corollary 21.** *Let $\mathcal{P}$ be a class of wdPTs that satisfies tractability condition (b) for some constant c. Then, for every $p \in \mathcal{P}$, the component width of p is at most $c + 1$.*

4 The FPT algorithm

Having defined the tractability conditions, we now show how they are used in the FPT-algorithm for p-EVAL($\mathcal{P}$) outlined in Algorithm 1.

The missing ingredient of Algorithm 1 that we have not yet introduced is $stop(\mathbf{S}, \mathbf{D})$ for a node component $\mathbf{S}$ and a database $\mathbf{D}$ which we explain now. Recall that we said earlier that the intention of the node components is to ensure a mapping to be maximal not by testing for extensions to the complete node, but to do these tests for smaller, independent units.

The idea how to realize this is to store in $\mathbf{D}'$ for each node component $\mathbf{S}$ those variable assignments ν to the variables in its existential interface such that there exists no extension $\nu' \colon \mathbf{S} \to \mathbf{D}$ of $\nu \cup \mu$. These are the values stored in $stop(\mathbf{S}_i, \mathbf{D})$.

In more detail, for a wdPT $((T, r), \lambda, \mathcal{X})$, a subtree T' of T containing r, a child node $t \in ch(T')$, node component $\mathbf{S} \in \mathcal{NC}_t$, a database $\mathbf{D}$, and a mapping $\mu \colon \mathsf{fvar}(T') \to \mathsf{dom}(\mathbf{D})$, consider the set $extend(\mathbf{S}, \mathbf{D}) = \{\eta \colon \mathcal{I}_t^{\exists}(\mathbf{S}) \to \mathsf{dom}(\mathbf{D}) \mid \text{there exists } \eta' \colon \mathbf{S} \to \mathbf{D} \text{ extending } \eta \text{ and } \mu|_{\mathsf{var}(\mathbf{S})}\}$. So $extend$ contains exactly those mappings on $\mathcal{I}_t^{\exists}(\mathbf{S})$ that *can* be extended in a way that is compatible with μ and maps $\mathbf{S}$ into $\mathbf{D}$. We thus set $stop(\mathbf{S}, \mathbf{D}) = \{\nu \colon \mathcal{I}_t^{\exists}(\mathbf{S}) \to \mathsf{dom}(\mathbf{D}) \mid \nu \notin extend(\mathbf{S}, \mathbf{D})\}$.

With this in place, we describe the idea of Algorithm 1. Recall that, given μ, we have to find a mapping μ' extending μ that is (1) a pp-solution, and (2) maximal. Because of the existential variables, there may be exponentially many subtrees T' of T containing r with $\mathsf{fvar}(T') = \mathsf{dom}(\mu)$, each being a potential candidate for witnessing (1) and (2). After removing all irrelevant nodes in line 1 (they might make evaluation unnecessarily hard), we thus check each of these subtrees (line 2).

If the required mapping μ' exists, then, as discussed earlier, for each child node of T' there exists at least one node component to which μ' cannot be extended. Not knowing which node components these are, the algorithm iterates over all possible combinations (line 4). In lines 5–7, the algorithm now checks whether there exists an extension of μ that maps all of $\lambda(T')$ into $\mathbf{D}$ (ensured by adding $\lambda(T')$ to q), but none of the node components $\mathbf{S}_1, \ldots, \mathbf{S}_n$. The latter property is equivalent to asking that μ' must assign a value to the existential interface variables of each $\mathbf{S}_i$ that cannot be extended. This is guaranteed by adding the atoms $\mathsf{cia}(\mathbf{S}_i)$ to q and providing in $\mathbf{D}'$ exactly the values from $\mathit{stop}(\mathbf{S}_i, \mathbf{D})$.

In order to see that this indeed gives an FPT algorithm in case tractability conditions (a) and (b) are satisfied, note that condition (b) ensures that the arity of each of the new relations for the atoms $\mathsf{cia}(\mathbf{S})$ is at most $c + 1$ (cf. Corollary 21). Thus the size of these relations (and thus the number of possible mappings in $\mathit{stop}(\mathbf{S}, \mathbf{D})$) is at most $|\mathsf{dom}(\mathbf{D})|^{(c+1)}$. Next, condition (a) ensures that for each mapping $\nu \colon \mathcal{I}_t^{\exists}(\mathbf{S}) \to \mathsf{dom}(\mathbf{D})$ deciding membership in $\mathit{stop}(\mathbf{S}, \mathbf{D})$ is in PTIME. Observe that the variables in $\mathcal{I}_t(\mathbf{S})$ are not considered in the computation of the treewidth since a fixed value is provided for them, thus they can be treated as constants. Finally, condition (b) also ensures that the test in line 7 is feasible in polynomial time. Again, since a fixed value is provided for the domain of μ, these variables can be treated as constants.

We note that the algorithm is an extension and refinement of the FPT algorithm presented in [15]. An inspection of [15] reveals that the conditions provided there imply our tractability conditions (a) and (b), but there is no implication in the other direction. In fact, our conditions explicitly describe the crucial properties of their restrictions that make the problem to be in FPT. From Algorithm 1 we thus derive the following result.

▶ **Theorem 22.** *Let $\mathcal{P}$ be a decidable class of wdPTs. If the tractability conditions (a) and (b) hold for $\mathcal{P}$, then* p-Eval($\mathcal{P}$) *can be solved in* FPT.

The correctness of the algorithm follows immediately from the previous discussion. For the runtime, in addition to what was already discussed, the number of loop-iterations in lines 2 and 4 is bounded by a function in the size of p, which is the parameter for the problem.

5 Optimality of the Tractability Conditions

We now show that both tractability criteria are necessary, and thus finish the proof of Theorem 18. We provide individual results for both conditions. In addition, we show that also the bound on the component width is necessary (and not just a side effect), which will turn out to be a useful result for proving that tractability condition (b) is necessary.

▶ **Lemma 23.** *Let $\mathcal{P}$ be a decidable class of simple wdPTs of bounded arity such that tractability condition (a) is not satisfied. Then* p-Eval($\mathcal{P}$) *is* coW[1]*-hard.*

Proof. For a wdPT $p \in \mathcal{P}$, let the *relevant components set* $\mathsf{rcs}(p)$ contain all the sets $\mathbf{S} \setminus \mathcal{I}_t(\mathbf{S})$ as defined in tractability condition (a). Moreover, let $\mathsf{rcs}(\mathcal{P}) = \bigcup_{p \in \mathcal{P}} \mathsf{rcs}(p)$. We will – by an FPT-reduction – reduce p-HOM($\mathsf{rcs}(\mathcal{P})$) to the complement of p-Eval($\mathcal{P}$). The result then follows from Theorem 7, as $\mathsf{rcs}(\mathcal{P})$ does not have bounded treewidth by assumption.

Consider an instance $\mathbf{E}, \mathbf{F}$ of p-HOM($\mathsf{rcs}(\mathcal{P})$). In a first step, find $p = ((T, r), \lambda, \mathcal{X}) \in \mathcal{P}$, a node $t \in \mathsf{relv}(T)$ such that $t \neq r$, and a node component $\mathbf{S} \in \mathcal{NC}_t$ such that $\mathbf{E} = \mathbf{S} \setminus \mathcal{I}_t(\mathbf{S})$. They exist by assumption and, since $\mathcal{P}$ is decidable, can be computed.

Since t is relevant, either for $t' = t$ or some descendant t' of t we have $\mathsf{fvar}(t') \setminus \mathsf{fvar}(\mathsf{branch}(t')) \neq \emptyset$. Among all possible candidates, pick some t' at a minimal distance to t.

We next define a database $\mathbf{D}$ over the set of relation symbols in p. For the description of $\mathbf{D}$, for all relation symbols R occurring in any atom $R(\vec{v}) \in \lambda(T)$, we will assume that $\vec{v}$ contains only variables, i.e. elements from $\mathcal{V}ar$. We implicitly assume that for all positions where $\vec{v}$ contains a constant, all atoms in $R^{\mathbf{D}}$ contain the same constant as in $\vec{v}$. Recall that we deal with simple wdPTs, thus each relation symbol occurs at most once within $\lambda(T)$. In the following, let $d \in \mathcal{C}onst$ be some fresh value not occurring in $\mathsf{dom}(\mathbf{F})$.

For each relation symbol R mentioned outside of $\lambda(\mathsf{cbranch}(t'))$, let $R^{\mathbf{D}} = \emptyset$.

For each relation symbol R mentioned in $\lambda(\mathsf{branch}(t))$, let $R^{\mathbf{D}} = \{R(d, \ldots, d)\}$.

For each relation symbol R mentioned in $\lambda(\mathsf{cbranch}(t') \setminus \mathsf{branch}(t)) \setminus \mathbf{S}$, let k be the arity of R and $R^{\mathbf{D}} = \{R(\vec{v}) \mid \vec{v} \in (\mathsf{dom}(\mathbf{F}) \cup \{d\})^k\}$.

For each relation symbol R mentioned in $\mathbf{S}$, observe that there exists a relation symbol R' in $\mathbf{E}$ that was derived from R when computing $\mathbf{S} \setminus \mathcal{I}_t(\mathbf{S})$. The idea is now to use $R^{\mathbf{D}}$ to simulate $R'^{\mathbf{F}}$ by padding the additional fields with d. Thus let k be the arity of R, let m be the arity of R', let $\{i_1, \ldots, i_\ell\} \subseteq \{1, \ldots, k\}$ be those positions of R containing values from $\mathcal{I}_t(\mathbf{S})$, and $\{o_1, \ldots, o_m\} = \{1, \ldots, k\} \setminus \{i_1, \ldots, i_\ell\}$ those positions of R that contain values from $\mathsf{var}(\mathbf{S}) \setminus \mathcal{I}_t(\mathbf{S})$. Then, for every $R'(a_{o_1}, \ldots, a_{o_m}) \in \mathbf{F}$, let $R^{\mathbf{D}}$ contain the atom $R(b_1, \ldots, b_k)$ where, for $1 \leq \alpha \leq k$, we have $b_\alpha = a_{o_j}$ if $\alpha = o_j$ for some $1 \leq j \leq m$ and $b_\alpha = d$ if $\alpha = i_j$ for some $1 \leq j \leq \ell$. This completes the definition of $\mathbf{D}$.

Finally, we define the mapping μ as $\mu(x) = d$ for all $x \in \mathsf{fvar}(\mathsf{branch}(t))$.

With the description of the reduction complete, we claim that $\mu \in p(\mathbf{D})$ if and only if there is no homomorphism from $\mathbf{E}$ to $\mathbf{F}$. We prove this property in two steps. First, we show that $\mu \in p(\mathbf{D})$ only depends on whether μ can be extended to t or not. After this we show that such an extension of μ exists if and only if there is a homomorphism $h\colon \mathbf{E} \to \mathbf{F}$.

First, observe that the only possible extension μ' of μ such that $\mu'(\tau) \in \mathbf{D}$ for every $\tau \in \lambda(\mathsf{branch}(t))$ is μ' mapping every variable in $\mathsf{var}(\mathsf{branch}(t))$ to d. Moreover, for all nodes $t'' \neq t$ in $ch(\mathsf{branch}(t))$ the mapping μ' cannot be extended to $\lambda(t'')$, since for all relation symbols R mentioned in $\lambda(t'')$ we have $R^{\mathbf{D}} = \emptyset$. Thus μ' is a pp-solution, and is a maximal pp-solution if and only if there exists no extension μ'' of μ' with $\mu''(\tau) \in \mathbf{D}$ for all $\tau \in \lambda(t)$.

Clearly, if μ' is a maximal pp-solution, then $\mu \in p(\mathbf{D})$. To see that $\mu \notin p(\mathbf{D})$ if μ' is not a maximal pp-solution, assume that there exists the above mentioned extension μ'' of μ'. Then μ'' can be obviously extended to μ''' with $\mu'''(\tau) \in \mathbf{D}$ for all $\tau \in \mathsf{cbranch}(t')$ since for all atoms on $(\mathsf{cbranch}(t') \setminus \mathsf{cbranch}(t)) \cup \{t'\}$, every possible atom over $\mathsf{dom}(\mathbf{D})$ is contained in $\mathbf{D}$. Since $\mathsf{dom}(\mu''')$ contains at least one free variable not in $\mathsf{dom}(\mu')$, this shows $\mu \notin p(\mathbf{D})$.

It thus remains to show that the extension μ'' of μ' exists if and only if there is a homomorphism $h\colon \mathbf{E} \to \mathbf{F}$. To see that this is the case, observe that by construction every such homomorphism h in combination with μ' gives a mapping from $\mathbf{S}$ into $\mathbf{D}$, and vice versa, every mapping $\mu\colon \mathbf{S} \to \mathbf{D}$ restricted to $\mathsf{dom}(\mathbf{E})$ gives the desired homomorphism. For the remaining atoms in $\lambda(t) \setminus \mathbf{S}$, observe that every possible mapping sends them into $\mathbf{D}$, since $\mathbf{D}$ again contains every possible atom for these relations. ◀

To simplify the proof that tractability condition (b) is necessary, we first show that having bounded component width is a necessary condition on its own.

▶ **Lemma 24.** *Let $\mathcal{P}$ be a decidable class of simple wdPTs of bounded arity. If there does not exist some constant c such that for every $p \in \mathcal{P}$ the component width is bounded by c, then* p-Eval($\mathcal{P}$) *is* coW[1]*-hard.*

Proof. The proof is an FPT-reduction of the problem of model checking FO sentences ϕ_k of the form $\phi_k = \forall x_1 \ldots \forall x_k \exists y \bigwedge_{i=1}^{k} E_i(x_i, y)$. Model checking for this class of sentences, parameterized by their size, is W[1]-hard [5]. Thus consider a formula ϕ_k and a database $\mathbf{E}$.

First, compute a wdPT $p = (T, \lambda, \mathcal{X}) \in \mathcal{P}$ with a component width of at least k. W.l.o.g. we assume that p contains only binary atoms: Since we assume a bounded arity, binary atoms can be easily encoded into atoms of higher arity. Consider the relevant node $t \in T$ and a node component $\mathbf{S} \in \mathcal{NC}_t$ such that the component width of $\mathbf{S}$ is at least k. Since we assume relations to be of some bounded arity, $\mathbf{S}$ cannot be of type (1) (Definition 12). W.l.o.g. we thus assume that $\mathbf{S}$ is of type (2).

Since t is relevant, either for $t' = t$ or some descendant t' of t we have $\mathsf{fvar}(t') \setminus \mathsf{fvar}(\mathsf{branch}(t')) \neq \emptyset$. Choose one such t' at a minimal distance to t.

For the description of the atoms in $\mathbf{D}$ for a relation symbol R occurring in an atom $R(\vec{v}) \in \lambda(T)$, we will assume that $\vec{v}$ contains only variables, i.e. elements from $\mathcal{V}ar$. We implicitly assume that for all positions where $\vec{v}$ contains a constant, all the atoms in $R^{\mathbf{D}}$ contain the same constant as in $\vec{v}$. Recall that we are dealing with simple wdPTs, thus each relation symbol R occurs at most once within $\lambda(T)$.

For each relation symbol R mentioned outside of $\lambda(\mathsf{cbranch}(t'))$, let $R^{\mathbf{D}} = \emptyset$.

For each relation symbol R mentioned in $\lambda(\mathsf{cbranch}(t')) \setminus \mathbf{S}$, let ℓ be the arity of R and $R^{\mathbf{D}} = \{R(\vec{v}) \mid \vec{v} \in \mathsf{dom}(\mathbf{E})^{\ell}\}$.

For the relation symbols R mentioned in $\mathbf{S}$, proceed as follows. Choose k interface variables $v_1, \ldots, v_k \in \mathcal{I}_t(\mathbf{S})$. Let $L = \mathsf{var}(\mathbf{S}) \setminus \mathsf{var}(\mathsf{branch}(t))$ be the "local variables" of $\mathbf{S}$. Observe that $\mathbf{S}$ being a node component of type (2) implies $L \neq \emptyset$. This is because of $\mathsf{var}(\mathbf{S}) \cap \mathsf{var}(\mathsf{branch}(t)) = \mathcal{I}_t(\mathbf{S})$, and the fact that to be a node component of type (2), in the Gaifman graph, all variables in $\mathcal{I}_t(\mathbf{S})$ must be connected to some variable from the node component that is not in $\mathcal{I}_t(\mathbf{S})$. Thus there must exist at least one "local variable". By the same reasoning, for each of the variables v_i, there must exist at least one atom $R_i(v_i, z_i)$ or $R_i(z_i, v_i)$ for some $z_i \in L$. We will assume $R_i(v_i, z_i)$ in the following, the other case is analogous. Now for each v_i, fix one such atom. Based on this, we define the following atoms to be contained in $\mathbf{D}$:

For each of the selected atoms $R_i(v_i, z_i)$, let $R_i^{\mathbf{D}} = E_i^{\mathbf{E}}$, i.e., we let R_i simulate exactly E_i. For every atom $R(z, z') \in \mathbf{S}$ such that $z, z' \in L$, define $R^{\mathbf{D}} = \{R(d, d) \mid d \in \mathsf{dom}(\mathbf{E})\}$. For the remaining atoms $R(z, z') \in \mathbf{S}$, define $R^{\mathbf{D}} = \{R(a, b) \mid a, b \in \mathsf{dom}(\mathbf{E})\}$.

Finally, μ is an arbitrary mapping $\mathsf{fvar}(\mathsf{branch}(t)) \to \mathsf{dom}(\mathbf{E})$.

It now follows by the same arguments as in the proof of Lemma 23 that we have $\mu \notin p(\mathbf{D})$ if and only if for every extension μ' of μ to $\mathsf{var}(\mathsf{branch}(t))$, there exists an extension ν of μ' such that $\nu(\tau) \in \mathbf{D}$ for all $\tau \in \mathbf{S}$.

To complete this proof, we thus only need to show that such an extension exists if and only if ϕ_k is satisfied. First, assume that ϕ_k is satisfied. Then, for all $z \in L$, define $\nu(z)$ to be the value of y in ϕ_k. This clearly maps $\mathbf{S}$ into $\mathbf{D}$. Next, assume that ϕ_k is not satisfied. Then there exists some assignment to $x_1, \ldots, x_k$ such that no suitable value for y exists. Then for the mapping μ' assigning exactly those values to the selected interface variables $v_1, \ldots, v_k$, there exists no extension of μ' to $\mathbf{S}$. This is because L defines a connected component in the Gaifman graph and because the definition of $\mathbf{D}$ forces all variables in L that occur together in some atom in $\mathbf{S}$ to be mapped to the same value by μ'. Thus μ' has to map all "local variables" in $\mathbf{S}$ to the same value, which would provide a suitable value for y, leading to a contradiction. This concludes the proof. ◀

▶ **Lemma 25.** *Let $\mathcal{P}$ be a decidable class of simple wdPTs of bounded arity such that tractability condition (b) is not satisfied. Then* p-Eval($\mathcal{P}$) *is either* coW[1]- *or* W[1]*-hard.*

Proof. First, assume that there exist some constant that is, for all $p \in \mathcal{P}$, an upper bound on the component width. Otherwise, p-Eval($\mathcal{P}$) is coW[1]-hard by Lemma 24. In particular, we may thus assume that all relations in all instances of $(\lambda(T') \cup \bigcup_{i=1}^{n} \{\mathsf{cia}(\mathbf{S}_i)\}) \setminus \mathsf{fvar}(T')$ for all $p \in \mathcal{P}$ are of bounded arity.

Let $solcheck(\mathcal{P})$ be the class of all sets of atoms $(\lambda(T') \cup \bigcup_{i=1}^{n} \{\mathsf{cia}(\mathbf{S}_i)\}) \setminus \mathsf{fvar}(T')$ for $\mathcal{P}$ as defined in tractability condition (b). We reduce p-HOM($solcheck(\mathcal{P})$) to p-Eval($\mathcal{P}$) via an FPT reduction. The result then follows directly by Theorem 7, since if (b) is false then $solcheck(\mathcal{P})$ has unbounded treewidth. The rest of this proof gives the desired reduction.

Let $\mathbf{E}, \mathbf{F}$ be an instance of p-HOM($solcheck(\mathcal{P})$). We construct a wdPT p, a database $\mathbf{D}$, and a mapping μ such that $\mu \in p(\mathbf{D})$ if and only if there is a homomorphism from $\mathbf{E}$ to $\mathbf{F}$.

First of all, find a $p = ((T, r), \lambda, \mathcal{X}) \in \mathcal{P}$ and a subtree T' of T containing r such that $\mathbf{E} = (\lambda(T') \cup \bigcup_{i=1}^{n} \{\mathsf{cia}(\mathbf{S}_i)\}) \setminus \mathsf{fvar}(T')$ for some combination $(\mathbf{S}_1, \ldots, \mathbf{S}_n) \in \mathcal{NC}_{t_1} \times \cdots \times \mathcal{NC}_{t_n}$ where $\{t_1, \ldots, t_n\} = ch(T') \cap \mathsf{relv}(T)$. To define a database $\mathbf{D}$ and a mapping μ such that $\mu \in p(\mathbf{D})$ if and only if a homomorphism from $\mathbf{E}$ to $\mathbf{F}$ exists, we need to define the following sets of nodes first. Let $K = ch(T') \cap \mathsf{relv}(T) = \{t_1, \ldots, t_n\}$. For each $t_i \in K$, we define the set N_i of nodes as follows. If $\mathsf{fvar}(t_i) \setminus \mathsf{fvar}(\mathsf{branch}(t_i)) \neq \emptyset$ (i.e., t_i contains some "new" free variable), then $N_i = \emptyset$. Otherwise, let $s_i \in T$ be a descendant of t_i such that $\mathsf{fvar}(s_i) \setminus \mathsf{fvar}(\mathsf{branch}(t_i)) \neq \emptyset$ and such that this property holds for no other node $s_i' \in T$ on the path from t_i to s_i. Then $N_i = \mathsf{cbranch}(s_i) \setminus \mathsf{cbranch}(t_i)$. Finally, let $N = \bigcup_{i=1}^{n} N_i$. Now all notions are in place to describe the database $\mathbf{D}$. While doing so, we implicitly assume that for all positions where an atom $R(\vec{v})$ of p contains a constant, all the atoms in $R^{\mathbf{D}}$ contain the same constant as in $\vec{v}$. I.e., we only describe the values for "variable positions" of $\vec{v}$. Recall that we are dealing with simple wdPTs, thus each relation symbol R occurs at most once within $\lambda(T)$.

For all atoms $R(\vec{y}) \in \lambda(T) \setminus (\lambda(T') \cup \lambda(K) \cup \lambda(N))$, let $R^{\mathbf{D}} = \emptyset$, i.e., for all atoms neither in T' nor in any of the relevant child nodes of T' (or their extensions to some node with a "new" free variable), no matching values exist in the database.

For all atoms $R(\vec{y}) \in \lambda(T')$, we want to use them to simulate in $\mathbf{D}$ the relations in $\mathbf{F}$. Observe that for each such atom, there exists an atom $R'(\vec{z}) \in \mathbf{E}$ that was derived from $R(\vec{y})$ by removing the free variables $\mathsf{fvar}(T')$. Thus, for each atom $R'(\vec{a}) \in R'^{\mathbf{F}}$, we add one atom $R(\vec{b})$ to $R^{\mathbf{D}}$ where $\vec{b}$ contains a fixed domain value $d \in \mathsf{dom}(\mathbf{F})$ at all positions $\vec{y}$ contains a free variable, and the corresponding value from $\vec{a}$ where the variable also occurs in $R'(\vec{z}')$. I.e., $R^{\mathbf{D}}$ is designed in such a way that all variables $x \in \mathsf{fvar}(T')$ can only be mapped to d.

The definition for the atoms in K is more involved. Consider some $t_i \in K$. Let $\vec{v}$ contain the existential interface variables of the node component $\mathbf{S}_i \in \mathcal{NC}_{t_i}$ selected for the construction of $\mathbf{E}$, and assume $\mathsf{cia}(\mathbf{S}_i) = R_{\mathsf{cia}}(\vec{v})$.

For all atoms $R(\vec{y}) \in \lambda(t_i) \setminus \mathbf{S}_i$, set $R^{\mathbf{D}} = \{R(\vec{a}) \mid \vec{a} \in \mathsf{dom}(\mathbf{F})^k\}$ where k is the arity of R. For the atoms in $\mathbf{S}_i$, we distinguish between $\mathbf{S}_i$ being of type (1) or of type (2).

If $\mathbf{S}_i$ is of type (1), i.e., $\mathbf{S}_i$ is of the form $\mathbf{S}_i = \{R(\vec{y})\}$ for some $R(\vec{y}) \in \lambda(t_i)$, define $R^{\mathbf{D}} = \{R(\vec{a}) \mid \vec{a} \in \mathsf{dom}(\mathbf{F})^k$ and $R_{\mathsf{cia}}(\vec{a}_{\vec{v}}) \notin \mathbf{F}\}$, where $\vec{a}_{\vec{v}}$ is the restriction of $\vec{a}$ to those positions in $\vec{y}$ with variables from $\vec{v}$ (and thus not containing variables from $\mathsf{fvar}(T')$).

If $\mathbf{S}_i$ is of type (2), we distinguish two types of variables: those that occur in $\mathcal{I}_t(\mathbf{S}_i)$, and those that do not appear in any node $t' \in \mathsf{branch}(t_i)$. We call these latter variables *new* variables and use as their domain the set $\mathsf{dom}(\mathbf{F})^{|\vec{v}|}$, i.e., the set of all possible assignments of values from $\mathbf{F}$ to the variables in $\vec{v}$ from $R_{\mathsf{cia}}(\vec{v})$. We assume that the encoding of the assignments $\vec{a} \in \mathsf{dom}(\mathbf{F})^{|\vec{v}|}$ is such that we can look up the value that is given to a variable $v_i \in \vec{v}$ by $\vec{a}$. For the remaining variables in $\mathsf{var}(\mathbf{S}_i)$, i.e., the variables not in $\vec{v}$, we will use values from $\mathsf{dom}(\mathbf{F})$. For each atom $R(\vec{y}) \in \mathbf{S}_i$, the values in $R^{\mathbf{D}}$ are defined as follows:

Let $\vec{z} = \vec{y} \cap (\mathsf{var}(\mathbf{S}_i) \setminus \vec{v})$ (because $\mathbf{S}_i$ is of type (2), $\vec{z} \setminus \mathsf{fvar}(T') \neq \emptyset$). Then $R^{\mathbf{D}}$ contains an atom for each tuple satisfying all of the following four properties.

1. All variables in $\mathsf{fvar}(T')$ get the value d.
2. All the variables in $\vec{z} \setminus \mathsf{fvar}(T')$ get assigned the same value. Denote this value by a, and recall that a represents an assignment $\vec{a} \in \mathsf{dom}(\mathbf{F})^{|\vec{v}|}$.

3. For all $v_i \in \vec{v} \cap \vec{y}$, the value of v_i is consistent with the vector $\vec{a}$ represented by a.

4. We have $R_{\mathsf{cia}}(\vec{a}) \notin \mathbf{F}$.

Because their arity is assumed to be bounded, all these relations can be constructed in polynomial time. To conclude the definition of $\mathbf{D}$, for all atoms $R(\vec{y}) \in \lambda(N)$, set $R^{\mathbf{D}} = \{R(\vec{a}) \mid \vec{a} \in \mathsf{dom}(\mathbf{D})^k\}$, where k is the arity of R and $\mathsf{dom}(\mathbf{D})$ is implicitly defined to consist of all values mentioned in the definition of $\mathbf{D}$.

Finally, let μ be the mapping defined on all variables $x \in \mathsf{fvar}(T')$ as $\mu(x) = d$. To prove $\mu \in p(\mathbf{D})$ if and only if homomorphism $\mathbf{E} \to \mathbf{F}$ exists, we first show the following claim.

▷ Claim 26. Let $R_{\mathsf{cia}}(\vec{v})$ be an interface atom and $\mathbf{S}_i$ the corresponding interface component. Then, for a mapping $\mu' \colon \vec{v} \to \mathsf{dom}(\mathbf{F})$, we have that $R_{\mathsf{cia}}(\mu'(\vec{v})) \in \mathbf{F}$ if and only if there is no extension μ'' of $\mu' \cup \mu$ to $\mathsf{var}(\lambda(t_i))$ that maps all atoms in $\mathbf{S}_i$ into $\mathbf{D}$ (since $\mathbf{S}_i \subseteq \lambda(t_i)$, this implies that there exists no extension μ'' of $\mu' \cup \mu$ that maps $\lambda(t_i)$ into $\mathbf{D}$).

Proof. For node components of type (1), the claim is immediate. So let us assume for the rest of the proof that $\mathbf{S}_i$ is of type (2).

First let $R_{\mathsf{cia}}(\mu'(\vec{v})) \in \mathbf{F}$. Let us assume an arbitrary extension μ'' of $\mu' \cup \mu$ to $\mathsf{var}(\mathbf{S}_i)$. If μ'' does not satisfy conditions 1., 2., and 3. for all $R(\vec{y}) \in \mathbf{S}_i$, then clearly for this particular atom there exists no atom in $\mathbf{D}$ to which it can be mapped by μ''. We may thus assume that μ'' satisfies the first three conditions for all atoms $R(\vec{y}) \in \mathbf{S}_i$. Then all variables in $\mathsf{var}(\mathbf{S}_i) \setminus (\vec{v} \cup \mathsf{fvar}(T'))$ take the same value under μ'', and this value corresponds exactly to the tuple $\mu'(\vec{v})$. But then μ'' does not satisfy condition 4. for any $R(\vec{y}) \in \mathbf{S}_i$ since $R_{\mathsf{cia}}(\mu'(\vec{v})) \in R^{\mathbf{F}}_{\mathsf{cia}}$ by assumption. Thus $R^{\mathbf{D}}$ does not contain any atom onto which μ'' could map $R(\vec{y})$ and thus μ'' cannot exist which completes the first direction.

For the other direction, assume that no extension μ'' of $\mu' \cup \mu$ maps all atoms in $\mathbf{S}_i$ into $\mathbf{D}$. Then this is in particular true for those assignments satisfying conditions 1., 2., and 3. Note that every such assignment maps all variables in $\vec{z} \setminus \mathsf{fvar}(T')$ to the same value, representing a mapping on $\vec{v}$. Also, $\mu''|_{\vec{v}} = \mu'$. Since μ'' fails to map $\mathbf{S}_i$ into $\mathbf{D}$ because of 4., we get that $R_{\mathsf{cia}}(\mu'(\vec{v})) \in R^{\mathbf{F}}_{\mathsf{cia}}$, which completes the proof of the claim. ◁

We continue the proof that $\mu \in p(\mathbf{D})$ if and only if a homomorphism $\mathbf{E} \to \mathbf{F}$ exists. First observe that $\mu \in p(\mathbf{D})$ if and only if on the one hand there is an extension μ' of μ to $\mathsf{var}(T')$ that maps all atoms in $\lambda(T')$ into $\mathbf{D}$ (of course, in general every subtree T'' containing the root node of T with $\mathsf{fvar}(T'') = \mathsf{dom}(\mu)$ is a potential candidate, but given the construction of $\mathbf{D}$, the subtree T' is the only possible candidate) and, on the other hand, for all $t_i \in K$, we have that there does not exist an extension of μ' onto $\lambda(t_i) \cup \lambda(N_i)$. (In fact, extending the mapping to any descendant of t_i that contains some additional free variable would work. However, the only nodes with non-empty relations in $\mathbf{D}$ are those mentioned in N.)

By the construction of $\mathbf{D}$ for atoms in $\lambda(N)$, for every $t_i \in K$ it follows immediately that there exists an extension of μ' onto $\lambda(t_i) \cup \lambda(N_i)$ if and only if there exists an extension to $\lambda(t_i)$. This is because for the atoms in $\lambda(N)$ the database $\mathbf{D}$ contains all possible atoms, thus every extension μ'' of μ' onto $\lambda(t_i)$ can be further extended to all atoms in $\lambda(N_i)$.

Note that the existence of an extension of μ' onto $\lambda(t_i)$ is, by Claim 26, equivalent to μ' sending $R_{\mathsf{cia}}(\vec{v})$ into $\mathbf{F}$. So $\mu \in p(\mathbf{D})$ if and only if there is a homomorphism from $\mathbf{E}$ into $\mathbf{F}$. This completes the proof. ◀

6 Relationship with SPARQL and Conclusion

Our results give a fine understanding of the tractable classes of wdPTs in the presence of projection. In particular they show the different sources of hardness. As laid out in the introduction, there is a strong relationship between well-designed SPARQL queries and wdPTs: For every well-designed SPARQL query, an equivalent well-designed pattern tree can be computed in polynomial time, and vice versa, in a completely syntactic way.

Note that our characterization of tractable classes unfortunately cannot be immediately translated to well-designed SPARQL queries. This is because our characterization only applies to classes of *simple* well-designed pattern trees. However, RDF triples and SPARQL triple patterns, in the relational model, are usually represented with a single (ternary) relation. Thus, there is no direct translation to and from simple (well-designed) pattern trees. As a consequence, our result does not imply an immediate characterization of the tractable classes of well-designed {AND, OPTIONAL}-SPARQL queries.

Nevertheless, our results also give interesting insights to SPARQL with projections. First, Algorithm 1 directly applies to queries in which relation symbols appear several times and thus in particular for well-designed pattern trees resulting from the translation of well-designed SPARQL queries. Moreover, our result determines completely the tractable classes that can be characterized by analyzing only the underlying graph structure of the queries, i.e., the Gaifman graph. Indeed, since simple queries can simulate all other queries sharing the same Gaifman graph by duplicating relations, Gaifman graph based techniques have exactly the same limits as simple queries. Thus, our work gives significant information on limits of tractability for SPARQL queries in the same way as e.g. [12, 4, 5] did in similar contexts.

Let us mention one major stumbling block towards a characterization of non-simple well-designed pattern trees with projections: In the proof of Lemma 24, we have used a reduction from quantified conjunctive queries. Unfortunately, the tractable classes for the non-simple fragment for that problem is not well understood which limits our result to simple queries since we are using the respective results from [5]. Note that we might have been able to give a more fine-grained result in sorted logics by using [6], but since this would, in our opinion, not have been very natural in our setting, we did not pursue this direction. Thus a better understanding of non-simple pattern trees would either need progress on quantified conjunctive queries or a reduction from another problem that is better understood.

One prominent operator of SPARQL that we did not consider is UNION, whose correspondence in pattern trees are sets of pattern trees, so-called pattern forests. While the extension to simple pattern forests is immediate (since no two trees share any relation symbols), it is not clear how to approach the possible repetition of relation symbol within different trees in forests of simple pattern trees in combination with projection.

Finally, another interesting class of queries are weakly well-designed pattern trees. While the tractability conditions can be easily adapted to provide FPT algorithms for these queries, providing a characterization of the tractable classes is much harder due to the fact that relevant nodes need not have a descendant introducing a "new" free variable.

References

1 Marcelo Arenas, Gonzalo I. Diaz, and Egor V. Kostylev. Reverse Engineering SPARQL Queries. In *Proc. WWW 2016*, pages 239–249. ACM, 2016.

2 Marcelo Arenas and Jorge Pérez. Querying semantic web data with SPARQL. In *Proc. PODS 2011*, pages 305–316. ACM, 2011.

3 Pablo Barceló, Markus Kröll, Reinhard Pichler, and Sebastian Skritek. Efficient Evaluation and Static Analysis for Well-Designed Pattern Trees with Projection. *ACM Trans. Database Syst.*, 43(2):8:1–8:44, 2018.

4 Hubie Chen. The tractability frontier of graph-like first-order query sets. In Thomas A. Henzinger and Dale Miller, editors, *Proc. CSL-LICS 2014*, pages 31:1–31:9. ACM, 2014.

5 Hubie Chen and Víctor Dalmau. Decomposing Quantified Conjunctive (or Disjunctive) Formulas. In *Proc. LICS 2012*, pages 205–214. IEEE Computer Society, 2012.

6 Hubie Chen and Dániel Marx. Block-Sorted Quantified Conjunctive Queries. In Fedor V. Fomin, Rusins Freivalds, Marta Z. Kwiatkowska, and David Peleg, editors, *Proc. ICALP 2013*, volume 7966 of *Lecture Notes in Computer Science*, pages 125–136. Springer, 2013.

7 Arnaud Durand and Stefan Mengel. Structural Tractability of Counting of Solutions to Conjunctive Queries. *Theory of Computing Systems*, pages 1–48, 2014.

8 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2006.

9 Martin Grohe. The Parameterized Complexity of Database Queries. In Peter Buneman, editor, *Proc. PODS 2001*, pages 82–92. ACM, 2001.

10 Martin Grohe. Parameterized Complexity for the Database Theorist. *SIGMOD Record*, 31(4):86–96, 2002.

11 Martin Grohe. The complexity of homomorphism and constraint satisfaction problems seen from the other side. *Journal of the ACM*, 54(1), 2007.

12 Martin Grohe, Thomas Schwentick, and Luc Segoufin. When is the evaluation of conjunctive queries tractable? In Jeffrey Scott Vitter, Paul G. Spirakis, and Mihalis Yannakakis, editors, *Proc. STOC 2001*, pages 657–666. ACM, 2001.

13 Mark Kaminski and Egor V. Kostylev. Beyond Well-designed SPARQL. In *Proc. ICDT 2016*, volume 48 of *LIPIcs*, pages 5:1–5:18. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2016.

14 Egor V. Kostylev, Juan L. Reutter, and Martín Ugarte. CONSTRUCT queries in SPARQL. In *Proc. ICDT 2015*, volume 31 of *LIPIcs*, pages 212–229. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2015.

15 Markus Kröll, Reinhard Pichler, and Sebastian Skritek. On the Complexity of Enumerating the Answers to Well-Designed Pattern Trees. In *Proc. ICDT 2016*, volume 48 of *LIPIcs*, pages 22:1–22:18. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2016.

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

17 Dániel Marx. Tractable hypergraph properties for constraint satisfaction and conjunctive queries. In Leonard J. Schulman, editor, *Proc. STOC 2010*, pages 735–744. ACM, 2010.

18 Christos H. Papadimitriou and Mihalis Yannakakis. On the Complexity of Database Queries. *J. Comput. Syst. Sci.*, 58(3):407–427, 1999.

19 Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. Semantics and complexity of SPARQL. *ACM Trans. Database Syst.*, 34(3), 2009.

20 François Picalausa and Stijn Vansummeren. What are real SPARQL queries like? In *Proc. SWIM 2011*, page 7. ACM, 2011.

21 Reinhard Pichler and Sebastian Skritek. Containment and equivalence of well-designed SPARQL. In *Proc. PODS 2014*, pages 39–50. ACM, 2014.

22 Miguel Romero. The Tractability Frontier of Well-designed SPARQL Queries. In *Proc. PODS 2018*, pages 295–306. ACM, 2018.