Theory of Computing Systems (2021) 65:3–41
https://doi.org/10.1007/s00224-020-10002-z

# Characterizing Tractability of Simple Well-Designed Pattern Trees with Projection

**Stefan Mengel**[1] · **Sebastian Skritek**[2]

Published online: 10 September 2020


## 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. For non-simple pattern trees the tractability criteria for simple pattern trees do not capture all tractable classes. We thus extend the characterization for the non-simple case in order to capture some additional tractable cases.

**Keywords** SPARQL · Well-designed pattern trees · Query evaluation · FPT · Characterizing tractable classes

This article belongs to the Topical Collection: *Special Issue on Database Theory (ICDT 2019)*
Guest Editor: Pablo Baceló

S. Skritek was supported by the Austrian Science Fund (FWF): P30930-N35

✉ Sebastian Skritek
skritek@dbai.tuwien.ac.at

Stefan Mengel
mengel@cril.fr

[1] CRIL, CNRS & Université d'Artois, Lens, France

[2] Faculty of Informatics, TU Wien, Vienna, Austria

## 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, see Pérez et al. [21], 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 cases where mapping the complete query into the database is impossible—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, see Pérez et al. [21], Letelier et al. [17], Arenas and Pérez [1], Pichler and Skritek [23], Picalausa and Vansummeren [22], Kostylev et al. [15], Barceló et al. [3], Arenas et al. [2], Romero [24]. Thus, many properties of and problems related to these 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 [21] and $\Sigma_2^{\mathsf{P}}$-complete in the presence of projection [17]. 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 of how suitable the approach of extending tractability conditions of CQs to wdPTs is for describing the space of tractable classes of wdPTs—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 by Romero [24]. Notably, as also observed for Boolean Conjunctive Queries by Grohe et al. [13] and Grohe [12], for wdPTs without projection these classes coincide with the ones for which evaluation is in FPT.

However, Romero [24] does not consider 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 aforementioned work on wdPTs is that problems become much more complex once projection is included. This is true for the computational complexity of the problems (e.g., as mentioned, for the evaluation problem it increases from coNP- to $\Sigma_2^{\mathsf{P}}$-completeness; for classical query containment, the NP-complete problem becomes even undecidable [23]) 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 consists 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 node outside the subtree that is a child node of a node in the subtree into the database. This is illustrated by the following example (a precise definition is given in Section 2).

*Example 1* Fig. 1 shows a wdPT $p$ with four nodes $r$, $t_1$, $t_2$, and $t_3$ where $r$ is the root node.

At its root node, the query is looking for information on flights: the airline operating the flight, the flight number, origin and destination. This information shall be extended by some contact information ($t_1$), and information on available seats on the flight ($t_2$) in case any of this information is available. If, in addition to the information on available seats also some ratings of the free seats are available ($t_3$), these shall be returned as well. Observe that the extensions to $t_1$ and $t_2$ are independent of each other. An equivalent SPARQL query (replacing relational atoms by triple patterns, and abbreviating variable and predicate names) would be

```
{{?airl operates ?fn . ?fn from ?origin . ?fn to ?dest}
                    OPTIONAL {?airl contact ?cd} }
                    OPTIONAL { {?fn avail_seats ?sn}
                                       OPTIONAL {?sn reviews ?r} }
```

For the database instance **D** also shown in the figure, the mapping $\mu$ defined as $\mu(\mathit{airline}) =$ "XYZ", $\mu(\mathit{flight\_number}) = 1$, $\mu(\mathit{origin}) =$ "LHR", $\mu(\mathit{destination}) =$ "LIS", and $\mu(\mathit{contact\_details}) =$ "e@ma.il" is an answer to $p$ over **D**. This is because of the subtree of $p$ consisting of the nodes $r$ and $t_1$. It can be checked that it satisfies all four conditions mentioned earlier. For the fourth condition, just observe that there exists no extension of $\mu$ that maps $\mathit{available\_seats}(\mathit{flight\_number}, \mathit{seat\_number})$ into **D**. Because of the fourth condition, the mapping $\nu$ with $\nu(\mathit{airline}) =$ "XYZ", $\nu(\mathit{flight\_number}) = 2$, $\nu(\mathit{origin}) =$ "LHR", $\nu(\mathit{destination}) =$ "LIS", and $\mu(\mathit{contact\_details}) =$ "e@ma.il" is no solution, because this mapping can be extended by $\nu(\mathit{seat\_number}) =$ "1A" in a way that maps $\mathit{available\_seats}(\mathit{flight\_number}, \mathit{seat\_number})$ also into **D**.

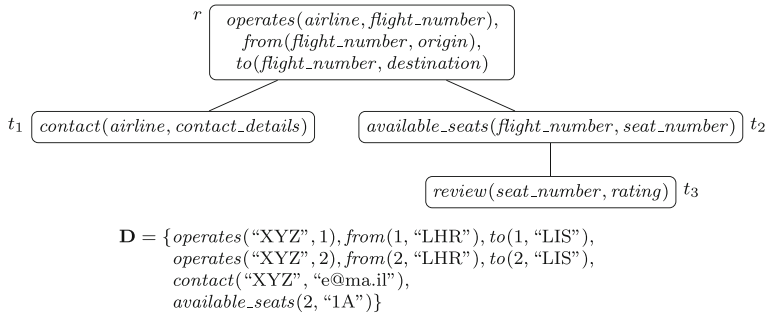


**Fig. 1** The wdPT $p$ and database **D** from Example 1

Without projection, the only hard part in deciding whether some mapping is a solution is to check for the existence of an extension. This requires a homomorphism test which is well known to be an NP-hard problem. 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 variables *flight_number* and *airline* are not part of the output but projected away. Then the mapping $\mu$ with $\mu(origin) =$ "LHR", $\mu(destination) =$ "LIS", and $\mu(contact\_details) =$ "e@ma.il" is a solution because of the extension $\mu(flight\_number) = 1$ and $\mu(airline) =$ "XYZ". Observe that the extension $\mu(airline) =$ "XYZ" and $\mu(flight\_number) = 2$ does not witness $\mu$ to be a solution, since, as already discussed before, this mapping is not maximal. As a result both, $\mu$ and its extension $\nu$ with $\nu(seat\_number) =$ "1A" (and otherwise defined like $\mu$) are solutions in this case.

As a consequence, besides testing some mapping for maximality, as a second source of hardness, different mappings on the existential variables have to be taken into account. In addition to the increased complexity of the evaluation problem, this also has the effect that the classes of wdPTs with projection for which query evaluation is in PTIME and in FPT no longer coincide, as observed by Kröll et al. [16]. 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 by Papadimitriou and Yannakakis [20], 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 [10, 11]. This allows for potentially costly preprocessing on the generally small query while the dependency on the generally far bigger database is polynomial for an exponent independent of the query. Parameterized complexity has found many applications in the complexity of query evaluation problems, see, e.g., Grohe et al. [13], Grohe [12], Marx [18], Chen [4], Romero [24].

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 Grohe et al. [13] and Chen [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.

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, i.e., such in which the same relational symbol may appear several times. However, in this case, there are classes of queries that do not satisfy our tractability criteria for simple queries and are still tractable. This shows that the restriction to simple wdPTs is crucial for the lower bounds. To extend the applicability of our techniques in the case of non-simple queries, we generalize our criteria by incorporating the notion of *cores* into well-designed pattern trees as it was done in the projection free case by Romero [24] and for conjunctive queries by Dalmau et al. [7]. While this allows us to show tractability for more classes of wdPTs, we do not achieve a full dichotomy in this setting.

**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. Letelier et al. [17], Kaminski and Kostylev [14], Romero [24], Barceló et al. [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 given by Romero [24] for the projection free case, we discuss how they describe the additional complexity introduced by projection. Note that some of the conditions provided here have precursors in Barceló et al. [3] and Kröll et al. [16] 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 show how to extend our tractability criteria for the non-simple case by incorporating the notion of cores and show how this allows us to capture more tractable classes. Besides a generalization of the tractability criteria from Section 3, we also introduce a generalization of the homomorphism problem, and completely characterize its tractable classes.

In Section 7, we discuss our results and potential extensions to conclude the paper.

This article is an extended version of the conference paper [19]. The main additional contribution compared to the conference version consists of Section 6, which is completely new. In addition, several minor improvements have been made throughout the article, including the extension of existing and addition of new examples.

## 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* $\sigma$ 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* $R_i(\mathbf{v})$ over $\sigma$ consists of a relation symbol $R_i \in \sigma$ and a tuple $\mathbf{v} \in (\mathcal{C}onst \cup \mathcal{V}ar)^{r_i}$. For an atom $\tau = R_i(\mathbf{v})$, let $\mathsf{dom}(\tau)$ denote the set of variables and constants occurring in $\mathbf{v}$. 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 the operators $\cup, \cap, \setminus$ also between sets $\mathcal{V}$ and tuples $\mathbf{v}$ of variables and constants. For example, $\mathsf{var}(\tau) = \mathbf{v} \cap \mathcal{V}ar$. We call a set of atoms *simple* if no relation symbol appears more than once in it.

Similarly, for a mapping $\mu$ we denote with $\mathsf{dom}(\mu)$ the set of elements on which $\mu$ is defined. For a mapping $\mu$ and a set $\mathcal{V} \subseteq \mathcal{V}ar$, we use $\mu|_{\mathcal{V}}$ to describe the restriction of $\mu$ to the variables in $\mathsf{dom}(\mu) \cap \mathcal{V}$. We say that a mapping $\mu$ is an *extension* of a mapping $\nu$ 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 $R(\mathbf{v}) \in \mathbf{A}$ by an atom $R^s(\mathbf{v}')$, where $\mathbf{v}'$ is obtained from $\mathbf{v}$ by removing elements of $\mathcal{A}$, and $s$ is the list of the removed positions and their values.

A *database* $\mathbf{D}$ over $\sigma$ is a finite set of atoms over $\sigma$ 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 Cores** A homomorphism $h$ between two sets $\mathbf{A}$ and $\mathbf{B}$ of atoms over $\sigma$ is a mapping $h\colon \mathsf{dom}(\mathbf{A}) \to \mathsf{dom}(\mathbf{B})$ such that for all atoms $R(\mathbf{v}) \in \mathbf{A}$ we have $R(h(\mathbf{v})) \in \mathbf{B}$, and such that $h(x) \neq x$ is only allowed if $x \in \mathsf{var}(\mathbf{A})$ (throughout the article, when defining homomorphisms we therefore only state the mapping on $\mathsf{var}(\mathbf{A})$, and assume the extension to constants via the identity mapping to be implicit). We write $h\colon \mathbf{A} \to \mathbf{B}$ to denote a homomorphism $h$ from $\mathbf{A}$ to $\mathbf{B}$.

Let $\mathbf{A}$ be a set of atoms. A minimal subset $\mathbf{A}' \subseteq \mathbf{A}$ such that there is a homomorphism $\mathbf{A} \to \mathbf{A}'$ is called a *core* of $\mathbf{A}$. We recall that all cores of $\mathbf{A}$ are unique up to isomorphism and thus speak of *the* core of $\mathbf{A}$ which we denote by $\mathsf{core}(\mathbf{A})$.

**Conjunctive Queries** We write conjunctive queries (short CQs) $q$ as $Ans(\mathbf{x}) \leftarrow \mathbf{B}$, where the body $\mathbf{B} = \{R_1(\mathbf{v}_1), \ldots, R_m(\mathbf{v}_m)\}$ is a set of atoms and $\mathbf{x}$ 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 $\mathsf{var}(\mathbf{B}) \setminus \mathbf{x}$. The result $q(\mathbf{D})$ of $q$ over a database $\mathbf{D}$ is the set of tuples $\{h(\mathbf{x}) \mid h\colon \mathbf{B} \to \mathbf{D}\}$, i.e., every homomorphism mapping the body of the query into the database is projected onto the values assigned to the free variables.

**Graphs** We consider only undirected, simple graphs $G = (V, E)$ with standard notations. We 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 $E(G_2) \subseteq E(G_1)$. For a

graph $G = (V, E)$ and a set $V' \subseteq V$, the *induced graph* $G[V']$ is the subgraph $G[V'] = (V', \{\{v_i, v_j\} \in E \mid v_i, v_j \in V'\})$. 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 $t$ (and $t$ 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 the root $r$ to the parent node of $t$ is denoted by $\mathsf{branch}(t)$. Moreover, $\mathsf{cbranch}(t) = \mathsf{branch}(t) \cup \{t\}$.

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 $\mathbf{A}$.

**Tree Decompositions and Treewidth** A *tree decomposition* of a graph $G = (V, E)$ is a pair $(T, \nu)$, 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 $(T, \nu)$ is $(\max\{|\nu(s)| \mid s \in V(T)\}) - 1$. The *treewidth* of $G$ is the minimum width of its tree decompositions.

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 $\sigma$ is a tuple $(T, \lambda, \mathcal{X})$ where $T$ is a rooted tree and $\lambda$ maps each node $t \in T$ to a set of relational atoms over $\sigma$. 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 $(T, \lambda)$ 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 1 (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 introduced by Letelier et al. [17], but borrow the term pp-solution from Kaminski and Kostylev [14].

**Definition 2** (**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 3** (**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 $\mu$ to $p'$ over $\mathbf{D}$ such that there exists no pp-solution $\mu'$ to $p'$ over $\mathbf{D}$ that is a proper extension of $\mu$.

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

*Example 3* Consider the PT $p = (T, \lambda, \mathcal{X})$ depicted in Fig. 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 $node_2(y_2)$ 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 $t_2$. Similarly, the wdPT in Fig. 1 is also well-designed. If there, in node $t_3$ one would ask for a review of the airline instead of the seat (i.e., having $\lambda(t_3) = \mathit{review}(\mathit{airline}, \mathit{rating})$), the resulting tree would no longer be well-designed, since the variable *airline* does not occur in $t_2$.

Returning to the wdPT in Fig. 2, consider a database $\mathbf{D}$ that, for each atom $R(\mathbf{v})$ in $\lambda(T)$ contains one atom $R(1, \ldots, 1)$ (i.e., with the value 1 at each position) and in addition the atoms $b_1(1, 2)$, $b_2(2, 2)$, and $b_3(2, 1)$. 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 $\mu_1'$ and $\mu_2'$ of $\mu_1$ and $\mu_2$, respectively. For $\mu_1'$, we have $\mathsf{dom}(\mu_1') = \mathsf{var}(T)$ and $\mu_1'(x) = 1$ for all $x \in \mathsf{dom}(\mu_1')$, and for $\mu_2'$ we have $\mathsf{dom}(\mu_2') = \mathsf{var}(\lambda(\{r, t_1, t_2, t_3\}))$ 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 $\mu_2'$ maps $r$, $t_1$, $t_2$, and $t_3$ into $\mathbf{D}$, but cannot be extended to neither $t_4$ nor $t_5$.

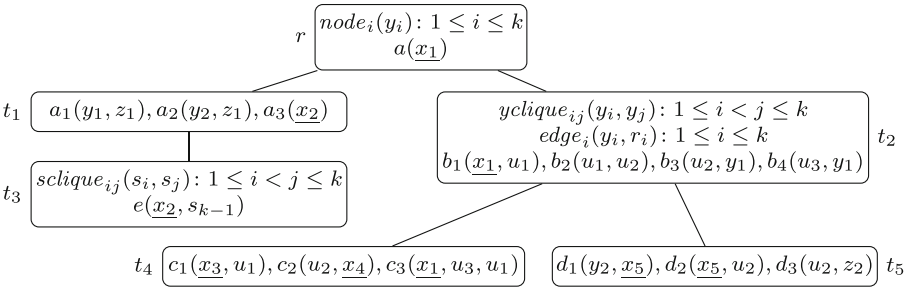


**Fig. 2** The well-designed pattern tree $p = (T, \lambda, \mathcal{X})$ of Example 3 that will serve as running example through Section 3. The free variables $x_1, \ldots, x_5$ are underlined

Orthogonally to wdPTs, *simple* pattern trees restrict the occurrences of relation symbols among their nodes: in a simple pattern tree, no relation symbol is allowed to occur more than once.

**Definition 4 (Simple PTs)** A PT $p = (T, \lambda, \mathcal{X})$ over $\sigma$ is a *simple pattern tree* if $\lambda(T)$ is simple and $\lambda(t) \cap \lambda(t') = \emptyset$ for all $t, t' \in T$ with $t \neq t'$.

**Parameterized Complexity** We only give a bare-bones introduction to parameterized complexity and refer the reader to [9] for more details. Let $\Sigma$ be a finite alphabet. A *parameterization* of $\Sigma^*$ is a polynomial time computable mapping $\kappa : \Sigma^* \to \mathbb{N}$. A *parameterized problem* over $\Sigma$ is a pair $(L, \kappa)$ where $L \subseteq \Sigma^*$ and $\kappa$ is a parameterization of $\Sigma^*$. 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 $f : \mathbb{N} \to \mathbb{N}$ 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 as follows.

| p-EVAL($\mathcal{P}$) | |
|---|---|
| INSTANCE: | A wdPT $p \in \mathcal{P}$, a database $\mathbf{D}$, and a mapping $\mu$. |
| 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 over $\Sigma$ and $\Sigma'$, respectively. An FPT-*reduction* from $E$ to $E'$ is a mapping $R : \Sigma^* \to (\Sigma')^*$ 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 $g : \mathbb{N} \to \mathbb{N}$ such that $\kappa'(R(x)) \leq g(\kappa(x))$ for all $x \in \Sigma^*$.

Of the rich hardness theory for parameterized problems, we will only use the classes W[1] and coW[1]. To keep this introduction short, we define a parameterized problem $(L, \kappa)$ to be W[1]-hard if there is a W[1]-hard problem $(L', \kappa')$ that FPT-reduces to $(L, \kappa)$. We define $(L, \kappa)$ 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 $(L', \kappa')$ from the literature. One important such problem is the homomorphism problem p−HOM($\mathcal{C}$) for a class $\mathcal{C}$ of sets of atoms defined as follows

| p-HOM($\mathcal{C}$) | |
|---|---|
| INSTANCE: | A set of atoms $\mathbf{A} \in \mathcal{C}$ and a set of atoms $\mathbf{B}$. |
| PARAMETER: | $\lvert \mathbf{A} \rvert$ |
| QUESTION: | Is there a homomorphism $h : \mathbf{A} \to \mathbf{B}$? |

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

Since simple sets of atoms are their own core, this immediately implies that for *simple* sets $\mathcal{C}$ of atoms, p-HOM($\mathcal{C}$) is in FPT if the *treewidth of each set* in $\mathcal{C}$ is bounded by $c$, and W[1]-hard otherwise. We remark that this result was in fact shown by Grohe et al. [13] in a predecessor paper of Grohe [12].

## 3 Tractability Conditions for Simple wdPTs

In this section we will introduce our tractability conditions for simple wdPTs. While, as mentioned, these criteria also apply to arbitrary wdPTs, they are not optimal in this case as we will see in Section 6. There we will also show generalizations of the conditions we give here that, for general wdPTs, describe more tractable classes. However, to simplify the presentation, in this section we tailor the definitions towards simple wdPTs.

We start by recalling that in *simple* pattern trees, no relation symbol is allowed to occur more than once. Our overall idea for solving p- EVAL($\mathcal{P}$) is as follows: given a wdPT $p$, a database $\mathbf{D}$, and a mapping $\mu$, construct a set of CQs $q$ with free variables $\mathbf{x}$ 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 $\mu(\mathbf{x})$ is in $q(\mathbf{D}')$. We give two tractability criteria ensuring that this algorithm is in FPT. Intuitively, one condition guarantees that deciding $\mu(\mathbf{x}) \in q(\mathbf{D}')$ 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. First of all, as already observed by Letelier et al. [17], some nodes of a wdPT may not be relevant for the answers returned by the query, in the following sense.

**Definition 5** (**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 $T$.

*Example 4* Recall the wdPT from Fig. 2, and consider the node $t_3$. Given any mapping $\mu$ on at least $\mathsf{var}(\lambda(r) \cup \lambda(t_1))$, whether this mapping can be extended to $t_3$ or not has no effect on the result, since an extension to $t_3$ does not include any new free variable. Note, however, that due to the clique of fresh existential variables $s_1, \ldots, s_k$ in $\lambda(t_3)$, deciding whether a mapping can be extended to $t_3$ is actually expensive. Thus, nodes like $t_3$ that do not influence any solution can safely be omitted.

Letelier et al. [17] 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 Letelier et al. [17] that these nodes can be easily detected.

**Proposition 1** (**Letelier et al.** [17][1]) *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 $t$.*

When testing whether a mapping can be extended to a node $t$ in a wdPT, the variables shared between $t$ and its parent node play a crucial role. First, they describe the relevant domain of the mapping to be extended. Second, the values for these variables are already determined. This not only reduces the number variables in $t$ for which a value must be found, but may also allow to partition the atoms in $t$ and then test each partition separately instead of all atoms in $t$ at once. We call these shared variables the interface of $t$.

**Definition 6** (**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., by Barceló et al. [3] and Kröll et al. [16] 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 Kröll et al. [16], 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 implement the idea of partitioning the set of atoms in a node using its interface. For this, we recall the notion of $S$-components from Durand and Mengel [8]. Originally, they were defined for graphs and then extended to sets of atoms. Since we will only use $S$-components of sets of atoms, we provide their definition directly, omitting the graph case. Let $\mathbf{A}$ be a set of atoms and $S \subseteq \mathsf{var}(\mathbf{A})$ a set of variables. Consider the dual graph $G_{\mathbf{A}} = (V_{\mathbf{A}}, E_{\mathbf{A}})$ with $V_{\mathbf{A}} = \{\tau \in \mathbf{A} \mid \mathsf{var}(\tau) \not\subseteq S\}$ and $E_{\mathbf{A}} = \{\{\tau_i, \tau_j\} \mid \tau_i, \tau_j \in \mathbf{A} \text{ s.t. } (\mathsf{var}(\tau_i) \cap \mathsf{var}(\tau_j)) \setminus S \neq \emptyset\}$. The connected components of $G_{\mathbf{A}}$ are the $S$-components of $\mathbf{A}$.

**Definition 7** (**Kröll et al.** [16]; **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 sets of atoms, defined as the union of:

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

[1] While not stated explicitly by Letelier et al. [17], it is an immediate consequence of their Theorem 3.21.

2. The set of all $\mathcal{I}(t)$-components of $\lambda(t)$.

In the following, node components of *type (1)* are the singleton sets of condition 1 and node components of *type (2)* are the $\mathcal{I}(t)$-components of condition 2.

*Example 5* Fig. 3 shows again the wdPT $p$ from Example 3, but omitting the non-relevant node $t_3$. In addition, at each node the node components are depicted by dotted boxes. To emphasize the difference between interface- and non interface variables and to highlight which variables connect atoms to node components, the interface variables at each node are grayed.

Consider the node $t_2$, which has the following node components: each atom $yclique_{ij}(y_i, y_j)$ forms a node component of type (1) since all variables $y_i$ occur also in $r$. In addition, there are several node components of type (2): each atom $edge_i(y_i, r_i)$ is such a node component (they contain a variable in $\mathsf{var}(t_2) \setminus \mathcal{I}(t_2)$ but do not share such a variable with another atom), and there are the two node components $\{b_1(x_1, u_1), b_2(u_1, u_2), b_3(u_2, y_1)\}$ and $\{b_4(u_3, y_1)\}$. Observe that these two sets are connected by $y_1$, but $y_1 \in \mathcal{I}(t_2)$ separates them into two node components. In contrast, the atoms $b_1(x_1, u_1)$, $b_2(u_1, u_2)$, and $b_3(u_2, y_1)$ are connected by $u_1$ and $u_2$, which are not in $\mathcal{I}(t_2)$.

Also, note the effect of considering interface variables and node components instead of looking at the complete node at once. For example, the Gaifman graph of $\lambda(t_2)$ has a treewidth of $k$. Thus, finding, e.g., values for $r_1, \ldots, r_k$ is a hard problem. However, by taking into account interface variables and node components, each of the resulting sets has a treewidth of 1. Therefore, the existence of an extension of some mapping on the interface variables to each node component independently can be decided efficiently.

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 exists if and only if the mapping can be extended to all of its node components. Thus, instead of testing extensions to the complete node at once (which might be intractable), we test the maximality of a mapping

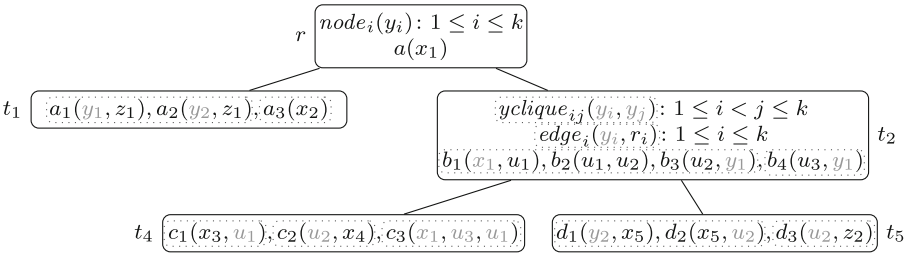


**Fig. 3** The well-designed pattern tree from Fig. 2, with the non-relevant node $t_3$ omitted and the interface variables at each node grayed to emphasize the node components, which are depicted by the dotted boxes (see Example 5)

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 8 (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 $\mathcal{I}(\mathbf{S}, t) = (\mathcal{I}(t) \cap \mathsf{var}(\mathbf{S}))$, and the *existential* interface is $\mathcal{I}^{\exists}(\mathbf{S}, t) = \mathcal{I}(\mathbf{S}, t) \setminus \mathcal{X}$.

We are now ready to formulate our first tractability condition. Intuitively, it 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. It exploits (and formalizes) the fact that once a mapping on $\mathcal{I}(\mathbf{S}, t)$ is given, these variables can be treated as constants.

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

*Example 6* To demonstrate the situation after introducing condition (a), let $p_k$ be the wdPT $p$ from Example 5 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 $\mu'$ 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, $t_4$ and $t_5$, a node component to which $\mu'$ cannot be extended is feasible in polynomial time as well. In fact, $\mathcal{NC}(t_4) = \{\{c_1(x_3, u_1)\}, \{c_2(u_2, x_4)\}, \{c_3(x_1, u_3, u_1)\}\}$ and $\mathcal{NC}(t_5) = \{\{d_1(y_2, x_5), d_2(x_5, u_2)\}, \{d_3(u_2, z_2)\}\}$ (this holds for every $p_k \in \mathcal{P}$). However, there may exist an exponential number of pp-solutions on $T'$ ($T'$ contains $2k+4$ 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 the second goal is to restrict the arity of

these relations. Towards the first goal, since the bodies of the resulting CQs are simple sets of atoms, tractability for the evaluation problem holds if these queries have bounded treewidth. As it turns out, a bound on the treewidth 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 $R(\mathbf{v})$ where $\mathbf{v}$ contains the variables in $\mathcal{I}^{\exists}(\mathbf{S}, t)$ 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 $R(\mathbf{v})$. Observe that these definitions imply $\mathsf{cia}(\mathbf{S}) = R()$ in case $\mathcal{I}^{\exists}(\mathbf{S}, t) = \emptyset$.

The intuition for $\mathsf{cia}(\mathbf{S})$ is that for each node component, we get one atom that covers exactly the variables in $\mathcal{I}^{\exists}(\mathbf{S}, t)$. 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 7* Recall again the wdPT depicted in Fig. 3 and consider the node $t_1$. It contains two node components: $\mathbf{S}_1 = \{a_1(y_1, z_1), a_2(y_2, z_1)\}$ and $\mathbf{S}_2 = \{a_3(x_2)\}$. Observe that whether $\mathbf{S}_2$ can be mapped into some database $\mathbf{D}$ is completely independent of the interface variables. Thus, $\mathsf{cia}(\mathbf{S}_2) = R_{\mathbf{S}_2}()$. However, for $\mathbf{S}_1$ the values of $y_1$ and $y_2$ influence whether $\mathbf{S}_1$ may be mapped. Thus, we get $\mathsf{cia}(\mathbf{S}_1) = R_{\mathbf{S}_1}(y_1, y_2)$. In case of the node component $\{c_3(x_1, u_3, u_1)\}$ of $t_4$, observe that we can assume some fixed value $\mu(x_1)$ for $x_1$, and thus can reduce the atom $c_3(\mu(x_1), u_3, u_1)$ according to this value. We therefore get as interface atom $R_{c_3}(u_3, u_1)$, with the corresponding database being computed based on $c_3^{x_1=\mu(x_1)}(u_3, u_1)$.

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 $T$ that contains $r$ and satisfies $V(T') \cap \mathsf{relv}(T) = V(T')$, 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 8* Recall the setting in Example 6, as well as the database instance $\mathbf{D}$ from Example 3. Let $\mu$ be a mapping with $\mathsf{dom}(\mu) = \{x_1, x_2\}$, and $\mu(x_1) = \mu(x_2) = 1$. Assume that, in order to show that $\mu \in p_k(\mathbf{D})$ for any $k \geq 2$ we are looking for a pp-solution for $T'$ that does not extend neither to the node component $\mathbf{S}_1 = \{c_1(x_3, u_1)\}$ of $t_4$ (and thus not to $\lambda(t_4)$), nor to the node component $\mathbf{S}_2 = \{d_1(y_2, x_5), d_2(x_5, u_2)\}$ of $t_5$ (and thus not to $\lambda(t_5)$). 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, note that because of the $yclique_{ij}$-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 $c_1(1,1) \in \mathbf{D}$, 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 $\{R_2(1,2), R_2(2,1), R_2(2,2)\}$.

Consider the mapping $\mu'$ with $\mathsf{dom}(\mu') = \mathsf{var}(\lambda(\{r, t_1, t_2, t_3\}))$ and $\mu'(x) = 1$ for all $x \in \mathsf{dom}(\mu')$ except for $u_1$ and $u_2$, for which $\mu'(u_i) = 2$. Now the mapping $\mu'$ witnesses the fact that $(1,1)$ is an answer to the query over $\mathbf{D}$, and thus $\mu'_2$ is a maximal pp-solution also witnessing $\mu \in p_k(\mathbf{D})$.

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 2** *Assume that* FPT $\neq$ 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 2 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 2, 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 8, we have $\mathcal{I}^{\exists}(\mathbf{S}, t) = \emptyset$ 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 1** *Assume that* FPT $\neq$ 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 1 could also be inferred as a special case of the main result of Romero [24]. 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 a 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 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 9** (**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 $|\mathcal{I}^{\exists}(\mathbf{S}, t)|$. 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 2** *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.

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

```
1:  T = T[relv(T)]                          ▷ Remove all nodes from T that are not relevant
2:  for all subtrees T' of T with fvar(T') = dom(μ) and r ∈ T' do
3:      Let {t_1, ..., t_n} = ch(T')
4:      for all (S_1, ..., S_n) ∈ NC(t_1) × ··· × NC(t_n) do
5:          q = "Ans(x) ← λ(T') ∪ {cia(S_1), ..., cia(S_n)}" ▷ Let x contain all x ∈ fvar(T')
6:          D' = D ∪ ⋃_{i=1}^{n} {R_i(ν(v_i)) | ν ∈ stop(S_i, D)}   ▷ Assume cia(S_i) = R_i(v_i)
7:          if μ(x) ∈ q(D') then EXIT(YES)
8:          end if
9:      end for
10: end for
11: EXIT(NO)
```

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

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

With this in place, we describe the idea of Algorithm 1. Recall that, given $\mu$, we have to find a mapping $\mu'$ extending $\mu$ 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, cf. Example 4), we thus check each of these subtrees (line 2).

If the required mapping $\mu'$ exists, then, as discussed earlier, for each child node of $T'$ there exists at least one node component to which $\mu'$ 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 $\mu$ 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 $\mu'$ 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 $\mathit{stop}(\mathbf{S}_i, \mathbf{D})$. Observe that the CQ $q$ in line 5 contains all $x \in \mathsf{fvar}(T')$, and that replacing them by $\mu(x)$ is only part of the evaluation strategy in line 7.

Most of the central components and ideas of the algorithm have already been demonstrated in isolation in the examples throughout Section 3. The next example brings together all these ideas by illustrating how the algorithm works.

*Example 9* Consider again the wdPT $p = (T, \lambda, \{x_1, \ldots, x_5\})$ depicted in Fig. 2, but without the atoms $\mathit{yclique}_{ij}(y_i, y_j)$ for $1 \leq i < j \leq k$. The resulting tree now satisfies conditions (a) and (b). Assume we want to decide $\mu \in p(\mathbf{D})$ for the mapping $\mu$ with $\mu(x_1) = 1$ and $\mu(x_2) = 2$ and

$$\begin{aligned}\mathbf{D} = {} & \{R(1, \ldots, 1) \mid R(\mathbf{v}) \in (\lambda(T) \setminus \{a_3(x_2)\})\} \cup \{a_3(2)\} \cup \\ & \{b_1(1,3), b_2(3,3), b_3(3,1), \} \cup \{b_1(2,2), b_2(2,2), b_3(2,2)\} \cup \\ & \{c_1(2,2), c_1(3,3)\} \cup \{c_2(3,3)\} \cup \\ & \{d_1(2,1), d_1(2,2), d_1(3,1), d_2(1,3), d_2(2,2)\}.\end{aligned}$$

In a first step, the algorithm now reduces $T$ by removing the non-relevant node $t_3$. Thus, from now on $T$ corresponds to the wdPT shown in Fig. 3 (again, without the $yclique_{ij}(y_i, y_j)$ atoms in $t_2$).

Next, there exist two subtrees $T'$of $T$ with $\mathsf{fvar}(T') = \mathsf{dom}(\mu) = \{x_1, x_2\}$. These are the subtrees $T_1 = T[\{r, t_1\}]$ and $T_2[\{r, t_1, t_2\}]$, respectively. Starting with $T_1$, we get $ch(T_1) = \{t_2\}$. Without the $yclique_{ij}$-atoms, the node $t_2$ has $k + 2$ node components. Since $t_2$ is the only child of $T_1$, the algorithm simply iterates over these $k + 2$ node components (line 4). We distinguish four different kinds of node components in $t_2$.

For components $\mathbf{S}$ of the form $edge_i(y_i, r_i)$, the component interface atoms are $\mathsf{cia}(\mathbf{S}) = R_i^{edge}(y_i)$. Thus, the CQ $q$ is $Ans(x_1, x_2) \leftarrow \lambda(T_1) \cup \{R_i^{edge}(y_i)\}$. Also, $extend(\mathbf{S}, \mathbf{D}) = \{\nu\}$ with $\nu(y_i) = 1$, and thus $\mathbf{D}' = \mathbf{D} \cup \{R_i^{edge}(2),\ R_i^{edge}(3)\}$. Clearly, we get $(1, 2) \notin q(\mathbf{D}')$.

For the node component $\mathbf{S} = \{b_1(x_1, u_1), b_2(u_1, u_2), b_3(u_2, y_1)\}$, we get $\mathsf{cia}(\mathbf{S}) = R_1(y_1)$ and therefore $q$ is $Ans(x_1, x_2) \leftarrow \lambda(T_1) \cup \{R_1(y_1)\}$, with $\mathbf{D}' = \mathbf{D} \cup \{R_1(2), R_1(3)\}$. The reason $R_1(2)$ is in this list is that $\mu(x_1) = 1$ is already given. Observe that because of $\{b_1(2, 2), b_2(2, 2), b_3(2, 2)\} \subseteq \mathbf{D}$, in principle a mapping $\nu(y_1) = 2$ can be extended to a mapping on $\mathbf{S}$. However, because of $\mu(x_1) = 1$, such a mapping would not be compatible with $\mu$, since mapping $b_1(x_1, u_1)$ to $b_1(2, 2)$ is not an option. As a result, again $(1, 2) \notin q(\mathbf{D}')$.

We omit the similar case for the last node component $\mathbf{S} = \{d_4(u_3, y_1)\}$.

Hence, $T_1$ does not witness $\mu$ being a solution, and therefore $T_2$ is tested next (line 2). The children of $T_2$ are $t_4$ and $t_5$. As already laid out in Example 6, $\mathcal{NC}(t_4) = \{\{c_1(x_3, u_1)\}, \{c_2(u_2, x_4)\}, \{c_3(x_1, u_3, u_1)\}\}$ and $\mathcal{NC}(t_5) = \{\{d_1(y_2, x_5), d_2(x_5, u_2)\}, \{d_3(u_2, z_2)\}\}$. This gives six different combinations of node components from $t_4$ and $t_5$. For the purpose of illustration, we give the component interface atoms and database extensions for each component.

For $\mathbf{S}_1 = \{c_1(x_3, u_1)\}$, we get $\mathsf{cia}(\mathbf{S}_1) = R_1(u_1)$ and $\mathbf{D}_1 = \{\}$; for $\mathbf{S}_2 = \{c_2(u_2, x_4)\}$, we get $\mathsf{cia}(\mathbf{S}_2) = R_2(u_2)$ and $\mathbf{D}_2 = \{R_2(2)\}$; and for $\mathbf{S}_3 = \{c_3(x_1, u_3, u_1)\}$ we get $\mathsf{cia}(\mathbf{S}_3) = R_3(u_3, u_1)$ with $\mathbf{D}_3 = \{R(a, b) \mid (a, b) \in (\{1, 2, 3\}^2 \setminus (1, 1))\}$. For the node components of $t_5$ we get for $\mathbf{S}_4 = \{d_1(y_2, x_5), d_2(x_5, u_2)\}$ the atom $\mathsf{cia}(\mathbf{S}_4) = R_4(y_2, u_2)$ and $\mathbf{D}_4 = \{R_4(1, 2),\ R_4(3, 2)\}$. Finally, for $\mathbf{S}_5 = \{d_3(u_2, z_2)\}$, we get $\mathsf{cia}(\mathbf{S}_5) = R_5(u_2)$ and $\mathbf{D}_5 = \{R_5(2),\ R_5(3)\}$.

Thus, for the first combination $(\mathbf{S}_1, \mathbf{S}_4)$ we get $q$ as $Ans(x_1, x_2) \leftarrow \lambda(T_2) \cup \{R_1(u_1), R_4(y_2, u_2)\}$ and $\mathbf{D}' = \mathbf{D} \cup \mathbf{D}_1 \cup \mathbf{D}_4$. Because of $\mathbf{D}_1 = \emptyset$, clearly $q(\mathbf{D}') = \emptyset$, and thus $(1, 2) \notin q(\mathbf{D}')$. By the same argument, this is also true for the combination $(\mathbf{S}_1, \mathbf{S}_5)$.

For the combination $(\mathbf{S}_2, \mathbf{S}_4)$, we get $q$ as $Ans(x_1, x_2) \leftarrow \lambda(T_2) \cup \{R_2(u_2), R_4(y_2, u_2)\}$ and $\mathbf{D}' = \mathbf{D} \cup \mathbf{D}_2 \cup \mathbf{D}_4$. Observe that $\mathbf{D}_2$ and $\mathbf{D}_4$ do not contain a compatible value for $u_2$. Thus, $q(\mathbf{D}')$ is again empty, showcasing that there must be a combined check for the node components of different nodes, and that this cannot be done in isolation.

For $(\mathbf{S}_2, \mathbf{S}_5)$, now $\mathbf{D}_2$ and $\mathbf{D}_5$ contain a compatible value for $u_2$, namely 2. However, by mapping $u_2$ to 2, there is no way to map $t_2$ into $\mathbf{D}$ such that $x_1$ is mapped to 1. Thus, also in this case $(1, 2) \notin q(\mathbf{D}')$.

For $(\mathbf{S}_3, \mathbf{S}_4)$, again $(1, 2) \notin q(\mathbf{D}')$: mapping all existential variables to 1 clearly does not allow to map neither $R_3(u_3, u_1)$ nor $R_4(y_2, u_2)$ into $\mathbf{D}'$. The only other way

to map $\lambda(t_2)$ into $\mathbf{D}'$ is by mapping $y_1$ to 1 and $u_2$ to 3. However, such a mapping cannot map $R_4(y_2, u_2)$ into $\mathbf{D}'$.

Finally, for $(\mathbf{S}_3, \mathbf{S}_5)$, it can now be checked that $(1, 2) \in q(\mathbf{D}')$, witnessed by the mapping $\mu$ with $\mu(x_1) = 1$, $\mu(x_2) = 2$, $\mu(u_1) = \mu(u_2) = 3$, and $\mu(v) = 1$ for all remaining variables $v \in \mathsf{var}(T_2)$. Thus, in line 7 the algorithm returns YES.

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 2). 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 $\nu\colon \mathcal{I}^{\exists}(\mathbf{S}, t) \to \mathsf{dom}(\mathbf{D})$ deciding membership in $\mathit{stop}(\mathbf{S}, \mathbf{D})$ is in PTIME. Observe that the variables in $\mathcal{I}(\mathbf{S}, t)$ 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 $\mu$, these variables can be treated as constants.

We note that the algorithm is an extension and refinement of the FPT algorithm of Kröll et al. [16]. An inspection of that paper 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 ensure the problem is in FPT. From Algorithm 1 we thus derive the following result.

**Lemma 1** *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 2. We provide individual results for both conditions. In addition, we show that 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 2** *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 $\mathbf{S} \setminus \mathcal{I}(\mathbf{S}, t)$ 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 1, as $\mathsf{rcs}(\mathcal{P})$ does not have bounded treewidth by assumption.

Consider an instance $\mathbf{E}, \mathbf{F}$ of p-HOM($\mathsf{rcs}(\mathcal{P})$). As 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 $\mathbf{E} = \mathbf{S} \setminus \mathcal{I}(\mathbf{S}, t)$. They exist by assumption and, since $\mathcal{P}$ is decidable, can be computed as follows: enumerate all possible candidate triples $p, t$ and $\mathbf{S}$, check if $p \in \mathcal{P}$ and if so verify the conditions on $p, t$ and $\mathbf{S}$. Since $p$ exists by assumption and all other steps are computable, this procedure eventually yields the desired triple.

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 $R(\mathbf{v}) \in \lambda(T)$, we will assume that $\mathbf{v}$ contains only variables, i.e., elements from $\mathcal{V}ar$. We implicitly assume that for all positions where $\mathbf{v}$ contains a constant, all atoms in $R^{\mathbf{D}}$ contain the same constant as in $\mathbf{v}$. Recall that we deal with simple wdPTs, thus each relation symbol occurs at most once within $\lambda(T)$. Also recall that, for any node $\bar{t} \in T$, $\mathsf{branch}(\bar{t})$ contains the nodes on the path from $r$ to the parent node of $\bar{t}$, while $\mathsf{cbranch}(\bar{t})$ in addition includes $\bar{t}$ itself. 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 $R^{\mathbf{D}} = \{R(\mathbf{v}) \mid \mathbf{v} \in (\mathsf{dom}(\mathbf{F}) \cup \{d\})^k\}$.

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 $\mathbf{S} \setminus \mathcal{I}(\mathbf{S}, t)$. The idea is now to use $R^{\mathbf{D}}$ to simulate the atoms with relation symbol $R'$ in $\mathbf{F}$ 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 $\mathcal{I}(\mathbf{S}, t)$, and $\{o_1, \ldots, o_m\} = \{1, \ldots, k\} \setminus \{i_1, \ldots, i_\ell\}$ those positions of $R$ that contain values from $\mathsf{var}(\mathbf{S}) \setminus \mathcal{I}(\mathbf{S}, t)$. 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 $\mu$ 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 $\mu$ can be extended to $t$ or not. After this we show that such an extension of $\mu$ exists if and only if there is a homomorphism $h\colon \mathbf{E} \to \mathbf{F}$.

First, observe that the only possible extension $\mu'$ of $\mu$ such that $\mu'(\tau) \in \mathbf{D}$ for every $\tau \in \lambda(\mathsf{branch}(t))$ is $\mu'$ 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 $\mu'$ cannot be extended to $\lambda(t'')$, since for all relation symbols $R$ mentioned in $\lambda(t'')$ we have $R^{\mathbf{D}} = \emptyset$. Thus, $\mu'$

is a pp-solution, and is a maximal pp-solution if and only if there exists no extension $\mu''$ of $\mu'$ with $\mu''(\tau) \in \mathbf{D}$ for all $\tau \in \lambda(t)$.

Clearly, if $\mu'$ is a maximal pp-solution, then $\mu \in p(\mathbf{D})$. To see that $\mu \notin p(\mathbf{D})$ if $\mu'$ is not a maximal pp-solution, assume the above mentioned extension $\mu''$ of $\mu'$ exists. Then $\mu''$ can be obviously extended to $\mu'''$ 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 $\mu''$ of $\mu'$ 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 $\mu'$ gives a homomorphism from $\mathbf{S}$ into $\mathbf{D}$, and vice versa, every homomorphism $\mu\colon \mathbf{S} \to \mathbf{D}$ 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 3** *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 from the problem of model checking FO sentences $\phi_k$ 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 $\phi_k$ and a database $\mathbf{E}$.

First, compute an arbitrary wdPT $p = (T, \lambda, \mathcal{X}) \in \mathcal{P}$ with a component width of at least $k$. Such a wdPT consists by assumption (otherwise the component width was bounded by $k$). 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 some 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 $k$ (they exist by construction). Since we assume relations to be of some bounded arity, $\mathbf{S}$ cannot be of type (1) (Definition 7). 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 $t$.

As in the proof of Lemma 2, for the description of $\mathbf{D}$ we assume for all $R(\mathbf{v}) \in \lambda(T)$ that $\mathbf{v}$ contains only variables. 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 $\ell$ be the arity of $R$ and $R^{\mathbf{D}} = \{R(\mathbf{v}) \mid \mathbf{v} \in \mathsf{dom}(\mathbf{E})^{\ell}\}$.

For the relation symbols $R$ mentioned in $\mathbf{S}$, proceed as follows. Choose $k$ interface variables $v_1, \ldots, v_k \in \mathcal{I}(\mathbf{S}, t)$. 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 $L \neq \emptyset$. For the same reason, 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, $\mu$ 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 2 that we have $\mu \notin p(\mathbf{D})$ if and only if for every extension $\mu'$ of $\mu$ to $\mathsf{var}(\mathsf{branch}(t))$, there exists an extension $\nu$ of $\mu'$ 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 $\phi_k$ is satisfied. First, assume that $\phi_k$ is satisfied. Then, for all $z \in L$, define $\nu(z)$ to be the value of $y$ in $\phi_k$. This clearly maps $\mathbf{S}$ into $\mathbf{D}$. Next, assume that $\phi_k$ 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 $\mu'$ assigning exactly those values to the selected interface variables $v_1, \ldots, v_k$, there exists no extension of $\mu'$ 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 $\mu'$. Thus, $\mu'$ 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 4** *Let $\mathcal{P}$ be a decidable class of simple wdPTs of bounded arity that does not satisfy tractability condition (b). Then* p-EVAL*($\mathcal{P}$) is either* coW[1]- *or* W[1]*-hard.*

*Proof* First, assume that there exists 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 3. 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 the problem p-HOM(*solcheck*($\mathcal{P}$)) to p-EVAL($\mathcal{P}$) via an FPT reduction. The result then follows directly by Theorem 1, 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 $\mu$ 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 $\{t_1, \ldots, t_n\} = ch(T') \cap \mathsf{relv}(T)$. Such $p$ exists by assumption and, since $\mathcal{P}$ is decidable, can be computed.

Next, the goal is to define a database $\mathbf{D}$ and a mapping $\mu$ such that $\mu \in p(\mathbf{D})$ if and only if a homomorphism from $\mathbf{E}$ to $\mathbf{F}$ exists. Before providing the formal construction, we sketch the idea first. Observe that $\mathbf{E}$ contains two types of atoms: $\lambda(T')$ and the interface atoms $\mathsf{cia}(\mathbf{S}_i)$. The idea is to define $\mathbf{D}$ in such a way that there is a homomorphism $\mu' \colon \lambda(T') \to \mathbf{D}$ if and only if there exists $h \colon \lambda(T') \to \mathbf{F}$, and that $\mu'$ *cannot* be extended to any child node of $T'$ if and only if $h$ can be extended to all atoms $\mathsf{cia}(\mathbf{S}_i)$. Consider the depiction of a possible wdPT $p$ in Fig. 4.

To implement the overall idea, we proceed as follows. We use atoms with the relation symbols in $\lambda(T')$ to simulate $\mathbf{F}$. For each child node $t_i$ of $T'$, we distinguish between the relation symbols in $\mathbf{S}_i$ and those not in $\mathbf{S}_i$. For those not in $\mathbf{S}_i$, we provide all possible atoms over the domain in $\mathbf{D}$. I.e., every homomorphism on $\lambda(T')$ can always be extended to the atoms not in $\mathbf{S}_i$. Whether it can be extended to all of $\lambda(t_i)$ thus only depends on $\mathbf{S}_i$. For those atoms we provide values that are only compatible with a homomorphism on $\lambda(T')$ if this homomorphism cannot be extended to $\mathsf{cia}(\mathbf{S}_i)$. However, given a mapping $\mu$ on $\mathsf{fvar}(T')$, for $\mu \notin p(\mathbf{D})$ it is not sufficient that every homomorphism $\mu' \colon \lambda(T') \to \mathbf{D}$ can be extended to at least one child node of $T'$: this child node must also contain a free variable not occurring in $T'$. If this is not the case for some child $t_i$, we pick one descendant $s_i$ of $t_i$ that contains a new free variable (since $t_i$ is relevant, $s_i$ exists), and for all relation symbols on the path from $t_i$ to $s_i$, we let $\mathbf{D}$ contain all possible atoms over the domain, thus making sure that if $\mu'$ can be extended to $t_i$, it can also be extended all the way to $s_i$.

We continue the formal definition of the reduction. To define a database $\mathbf{D}$ and a mapping $\mu$ 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 on the path from $t_i$ to $s_i$. Then $N_i = \mathsf{cbranch}(s_i) \setminus \mathsf{cbranch}(t_i)$. (E.g., in Fig. 4, $N_1$ contains the two nodes with bold borders.) Finally, let $N = \bigcup_{i=1}^{n} N_i$. We can now describe the database $\mathbf{D}$. While

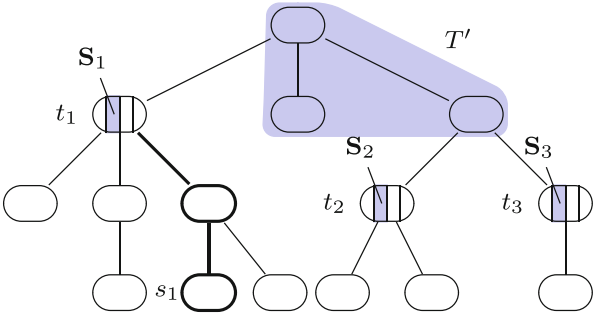


**Fig. 4** Illustration of the different parts of a wdPT distinguished in the proof of Lemma 4. The "slots" in the nodes $t_1$, $t_2$, and $t_3$ represent the node components of these nodes. While we assume $t_2$ and $t_3$ to contain a free variable not occurring in $T'$, $t_1$ does not contain such a variable. The node $s_1$ is one possible descendant of $t_1$ with a free variable not in $T'$

doing so, we implicitly assume that for all positions where an atom $R(\mathbf{v})$ of $p$ contains a constant, all the atoms in $R^{\mathbf{D}}$ contain the same constant as $\mathbf{v}$. I.e., we describe only the values for "variable positions" of $\mathbf{v}$. Recall that we are dealing with simple wdPTs, thus each relation symbol $R$ occurs at most once within $\lambda(T)$.

For all atoms $R(\mathbf{y}) \in \lambda(T) \setminus (\lambda(T') \cup \lambda(K) \cup \lambda(N))$, 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 $R(\mathbf{y}) \in \lambda(T')$, 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 $R'(\mathbf{z}) \in \mathbf{E}$ that was derived from $R(\mathbf{y})$ by removing the free variables $\mathsf{fvar}(T')$. Thus, for each atom of the form $R'(\mathbf{a})$ (i.e., atoms with relation symbol $R'$) in $\mathbf{F}$, we add one atom $R(\mathbf{b})$ to $R^{\mathbf{D}}$ where $\mathbf{b}$ contains a fixed domain value $d \in \mathsf{dom}(\mathbf{F})$ at all positions where $\mathbf{y}$ contains a free variable, and the value from $\mathbf{a}$ at those positions where the variable still occurs in $R'(\mathbf{z}')$. 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 $\mathbf{v}$ 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 $\mathsf{cia}(\mathbf{S}_i) = R_{\mathsf{cia}}(\mathbf{v})$.

For all atoms $R(\mathbf{y}) \in \lambda(t_i) \setminus \mathbf{S}_i$, set $R^{\mathbf{D}} = \{R(\mathbf{a}) \mid \mathbf{a} \in \mathsf{dom}(\mathbf{F})^k\}$ 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 $\mathbf{S}_i = \{R(\mathbf{y})\}$ for some $R(\mathbf{y}) \in \lambda(t_i)$, define $R^{\mathbf{D}} = \{R(\mathbf{a}) \mid \mathbf{a} \in \mathsf{dom}(\mathbf{F})^k$ and $R_{\mathsf{cia}}(\mathbf{a}_{\mathbf{v}}) \notin \mathbf{F}\}$, where $\mathbf{a}_{\mathbf{v}}$ is the restriction of $\mathbf{a}$ to those positions in $\mathbf{y}$ with variables from $\mathbf{v}$ (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 $\mathcal{I}(\mathbf{S}_i, t)$, 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 $\mathsf{dom}(\mathbf{F})^{|\mathbf{v}|}$, i.e., the set of all possible assignments of values from $\mathbf{F}$ to the variables in $\mathbf{v}$ from $R_{\mathsf{cia}}(\mathbf{v})$. We assume that the encoding of the assignments $\mathbf{a} \in \mathsf{dom}(\mathbf{F})^{|\mathbf{v}|}$ is such that we can look up the value that is given to a variable $v_i \in \mathbf{v}$ by $\mathbf{a}$. For the remaining variables in $\mathsf{var}(\mathbf{S}_i)$, we will use values from $\mathsf{dom}(\mathbf{F})$. For each atom $R(\mathbf{y}) \in \mathbf{S}_i$, the values in $R^{\mathbf{D}}$ are defined as follows:

Let $\mathbf{z} = \mathbf{y} \cap (\mathsf{var}(\mathbf{S}_i) \setminus \mathbf{v})$ (because $\mathbf{S}_i$ is of type (2), $\mathbf{z} \setminus \mathsf{fvar}(T') \neq \emptyset$). 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 $\mathbf{z} \setminus \mathsf{fvar}(T')$ get assigned the same value. Denote this value by $a$, and recall that $a$ represents an assignment $\mathbf{a} \in \mathsf{dom}(\mathbf{F})^{|\mathbf{v}|}$.
3. For all $v_i \in \mathbf{v} \cap \mathbf{y}$, the value of $v_i$ is consistent with the vector $\mathbf{a}$ represented by $a$.
4. We have $R_{\mathsf{cia}}(\mathbf{a}) \notin \mathbf{F}$.

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 $R(\mathbf{y}) \in \lambda(N)$, set $R^{\mathbf{D}} = \{R(\mathbf{a}) \mid \mathbf{a} \in \mathsf{dom}(\mathbf{D})^k\}$, 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 $\mu$ 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* Let $R_{\mathsf{cia}}(\mathbf{v})$ be an interface atom and $\mathbf{S}_i$ the corresponding node component. Then, for a mapping $\mu' \colon \mathbf{v} \to \mathsf{dom}(\mathbf{F})$, we have that $R_{\mathsf{cia}}(\mu'(\mathbf{v})) \in \mathbf{F}$ if and only if there is no extension $\mu''$ 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 $\mu''$ of $\mu' \cup \mu$ that maps $\lambda(t_i)$ into $\mathbf{D}$).

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

For the other direction, assume that no extension $\mu''$ 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 $\mathbf{z} \setminus \mathsf{fvar}(T')$ to the same value, representing a mapping on $\mathbf{v}$. Also, $\mu''|_{\mathbf{v}} = \mu'$. Since $\mu''$ fails to map $\mathbf{S}_i$ into $\mathbf{D}$ because of 4., we get that $R_{\mathsf{cia}}(\mu'(\mathbf{v})) \in R^{\mathbf{F}}_{\mathsf{cia}}$, 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 $\mu'$ of $\mu$ 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 $\mu'$ 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 $\mu'$ 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 $\mu''$ of $\mu'$ onto $\lambda(t_i)$ can be further extended to all atoms in $\lambda(N_i)$.

Note that the existence of an extension of $\mu'$ onto $\lambda(t_i)$ is, by the Claim shown above, equivalent to $\mu'$ sending $R_{\mathsf{cia}}(\mathbf{v})$ 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 Tractability Conditions for Non-simple wdPTs

As already mentioned at the beginning of Section 3, the tractability conditions presented there are not restricted to simple wdPTs, but also apply to arbitrary wdPTs. I.e., deciding p-EVAL($\mathcal{P}$) is in FPT also for classes $\mathcal{P}$ of non-simple wdPTs. However, in case that the same relation symbol may occur in more than one atom throughout the query, the tractability criteria stated in the previous section miss important classes of tractable wdPTs. In the following, we discuss more general tractability notions.

The key difference for non-simple wdPTs is that in this setting, a set of atoms is not necessarily its own core. Since for the homomorphism problem, not the treewidth of the set of atoms, but the treewidth of its *core* determines the complexity of the problem as shown by Grohe [12] and recalled in Theorem 1, the concept of cores must also be taken into account for the tractability conditions. To do so, we revisit both, the notion of cores and the homomorphism problem, studying variations more suitable to our setting, and then apply these results to extend the definition of tractable classes.

### 6.1 Extension Cores

In this section, we will introduce a variant of cores we call *extension core* that will turn out to be the right notion for wdPTs. The reason for this is that while the core is the suitable concept for the homomorphism problem (cf. Theorem 1), when evaluating wdPTs we actually deal with a variant of the homomorphism problem. To further motivate the idea of these extensions, recall that when deciding p-EVAL($\mathcal{P}$) for some wdPT $p = (T, \lambda, \mathcal{X})$, one important step is to check, given some subtree $T'$ of $T$, a mapping $\mu$ and some child node of $T'$, whether $\mu'$ can be extended to this child. While this problem can be correctly stated as an instance of HOM, such a formulation could not adequately express all available information. In fact, compared to HOM, the problem at hand contains both, additional constraints (for some variables in the child node, the value is already determined by $\mu'$) and additional information (we know that $\mu'$ maps $\lambda(T')$ into the database). The formulation as an instance of HOM could not utilize all of the information at hand, and as a result the problem might look harder than it actually is.

*Example 10* Consider the (non-simple) wdPT $p_k = (T, \lambda_k, \{x_1, x_2\})$ (parameterized by $k$) depicted in Fig. 5 consisting of two nodes. Clearly, the class $\mathcal{P} = \{p_k \mid k \geq 2\}$ of wdPTs does neither satisfy tractability condition (a) nor (b). In both cases, the reason for the violation is the clique of $\mathit{boss\_of}(y_i, y_j)$ atoms in $t_1$, which resides in a single node component.

Nevertheless, deciding p-EVAL($\mathcal{P}$) is in FPT. As mentioned, the reason for this is that tractability for the homomorphism problem depends on the treewidth of the core of the structure, and not on the treewidth of the structure.

That is, consider tractability condition (b) first. It ensures that in line 7 of Algorithm 1 the CQ can be evaluated efficiently. Now condition (b) is not satisfied because of the "subtree" of $T$ which contains all of $T$. In this case, the

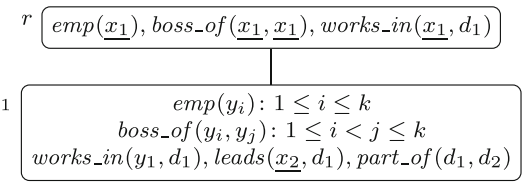


**Fig. 5** The wdPT $p_k$ from Example 10. The free variables are underlined

resulting CQ $q_k$ is $Ans(x_1, x_2) \leftarrow \lambda_k(T)$. However, given concrete values $a$ and $b$ for $x_1$ and $x_2$, the evaluation problem for the class of Boolean CQs $\{q_k(a, b) \mid p_k \in \mathcal{P}\}$ (where $q_k(a, b)$ is the query $q_k$ with $x_1$ and $x_2$ in $\lambda_k(T)$ being replaced by $a$ and $b$, respectively) is tractable. This is because $\mathsf{core}(\lambda_k(T)) = \{emp(x_1), boss\_of(x_1, x_1), works\_in(x_1, d_1), leads(x_2, d_1), part\_of(d_1, d_2)\}$ due to the homomorphism $h$ with $h(y_i) = x_1$ for $1 \leq i \leq k$. The treewidth of these cores is obviously bounded by some constant, and thus deciding the problem is tractable (c.f. Theorem 1).

While adapting tractability condition (b) is rather straightforward, the situation is more interesting for condition (a). Recall that the idea of condition (a) is to ensure that deciding whether a mapping on the interface variables of a node component can be extended to this node component is tractable. Consider the node component $\mathbf{S}$ in $t_1$ that contains the set of $boss\_of(y_i, y_j)$ atoms. Clearly, for increasing $k$ the treewidth of this component is not bounded, and neither is the treewidth of its core (since it already is its own core). Deciding whether a mapping on $d_1$ (the only interface variable) can be extended to this component is nevertheless tractable.

The reason for this is that it is only necessary to consider mappings $\mu$ on $d_1$ that have an extension which maps $\lambda_k(r)$ into a given database. Thus, for deciding the extension to $t_1$, we can assume the existence of such a mapping on $\lambda_k(r)$. In our case, we thus know that the database contains a target for $emp(x_1)$, $boss\_of(x_1, x_1)$, and $works\_in(x_1, d_1)$. Thus, instead of considering just the core of $\mathbf{S}$, we can consider the core of $\mathbf{S} \cup \lambda_k(r)$. Again by the same homomorphism as before, that is $h(y_i) = x_1$ for all $1 \leq i \leq k$, we can now fold parts of $\mathbf{S}$ into $\lambda_k(r)$. In the next step, we can remove again all atoms from $\lambda_k(r)$, since for them the existence of a mapping into the database is already known, and finally only need to decide the existence of an extension of $\mu$ on the remaining set of atoms, which again, is tractable for $\mathcal{P}$.

To formalize this idea of "folding" parts of node components into the set of atoms on the branch to this component, we introduce the notion of an *extension core*. We then introduce the problem EXT($\mathcal{C}$) which captures the idea of finding extensions to a given mapping, and characterize its tractable classes. Using this problem, we then introduce refined versions of tractability conditions (a) and (b) based on the notion of the extension core, and show that these improved conditions indeed guarantee tractability for p-EVAL($\mathcal{P}$).

Towards the definition of the *extension core*, for a set $\mathcal{A} \subseteq \mathcal{C}onst \cup \mathcal{V}ar$ of elements, let $\mathbf{Fix}_{\mathcal{A}}$ be the set of atoms $\mathbf{Fix}_{\mathcal{A}} = \{R_a(a) \mid a \in \mathcal{A}\}$ where each $R_a$ is a unique relation symbol.

**Definition 10** (**Extension Core**) Let $(\mathbf{A}, \mathbf{B})$ be a pair of sets of atoms. The *extension core* $\mathsf{extcore}(\mathbf{A}, \mathbf{B})$ is the set $\mathsf{extcore}(\mathbf{A}, \mathbf{B}) = (\mathsf{core}(\mathbf{A} \cup \mathbf{B} \cup \mathbf{Fix}_{\mathsf{dom}(\mathbf{A})}) \setminus \mathbf{Fix}_{\mathsf{dom}(\mathbf{A})}) \setminus \mathsf{dom}(\mathbf{A})$.

Said differently, the extension core is constructed by introducing a new relation for every domain element in $\mathbf{A}$ and then computing the core. That is, the extension core accounts on the one hand for the possibility that parts of $\mathbf{B}$ might be folded into $\mathbf{A}$ (and thus the extension of the homomorphism to these parts is guaranteed), and on the other hand for the fact that the mapping on $\mathsf{dom}(\mathbf{A})$ is fixed. Removing $\mathsf{dom}(\mathbf{A})$ is then possible because the mapping is already provided for these values.

*Example 11* To illustrate Definition 10, consider the wdPT from Fig. 5, and let $\mathbf{A} = \lambda(r)$ and $\mathbf{B} = \lambda(t_1)$. Then $\mathbf{Fix}_{\mathsf{dom}(\mathbf{A})} = \{R_{x_1}(x_1), R_{d_1}(d_1)\}$. Then $\mathsf{core}(\mathbf{A} \cup \mathbf{B} \cup \mathbf{Fix}_{\mathsf{dom}(\mathbf{A})}) = \{emp(x_1),\ boss\_of(x_1, x_1),\ works\_in(x_1, d_1),\ leads(x_2, d_1),\ part\_of(d_1, d_2),\ R_{x_1}(x_1),\ R_{d_1}(d_1)\}$ (mapping all $y_i$ to $x_1$). The extension core then is $\{emp^{1=x_1}(),\ boss\_of^{1=x_1,2=x_1}(),\ works\_in^{1=x_1,2=d_1}(),\ leads^{2=d_1}(x_2),\ part\_of^{1=d_1}(d_2)\}$.

### 6.2 The Extension Problem

A task within the evaluation problem of wdPTs is to test whether a mapping on some subtree is maximal or can be extended to some child node. We formalize this by the following problem, where $\mathcal{C}$ is a class of pairs of sets of atoms.

| EXT($\mathcal{C}$) | |
|---|---|
| INPUT: | A pair $(\mathbf{A}, \mathbf{B}) \in \mathcal{C}$ of sets $\mathbf{A}, \mathbf{B}$ of atoms, a set $\mathbf{C}$ of atoms, and a homomorphism $h\colon \mathbf{A} \to \mathbf{C}$. |
| QUESTION: | Exists a homomorphism $h'\colon \mathbf{B} \to \mathbf{C}$ compatible with $h$? |

The parameterized problem p-EXT($\mathcal{C}$) is the problem EXT($\mathcal{C}$) parameterized by the size of $(\mathbf{A}, \mathbf{B})$.

The main difference between HOM and EXT is that in addition to a set of atoms, the input of EXT gets another set of atoms and a homomorphism that is already guaranteed to map this additional set of atoms into the target. When looking for tractable classes, this additional input has to be taken into account. This is exactly the role of extension cores as defined in the last section.

While here we use the extension problem to define tractable fragments of the evaluation problem of wdPTs with projection, we note that it also allows for an alternative formulation of the tractable classes of projection free wdPTs defined by Romero [24].

With these definitions settled, we next use extension cores to provide an exact characterization of the tractable classes $\mathcal{C}$ of the extension problem EXT($\mathcal{C}$). To this end, we define the treewidth of $\mathsf{extcore}(\mathcal{C})$ to be the maximum of the treewidth of $\mathsf{extcore}(\mathbf{A}, \mathbf{B})$ for $(\mathbf{A}, \mathbf{B}) \in \mathcal{C}$ if this maximum exists and $\infty$ otherwise.

**Theorem 3** *Assume that* FPT $\neq$ W[1] *and let* $\mathcal{C}$ *be a decidable class of pairs of sets of atoms. Then the following statements are equivalent:*

1. *The treewidth of* $\mathsf{extcore}(\mathcal{C})$ *is bounded by a constant.*
2. *The problem* EXT$(\mathcal{C})$ *is in* PTIME.
3. *The problem* p-E*XT*$(\mathcal{C})$ *is in* FPT.

Theorem 3 is shown using a sequence of lemmas given below. First, we state an easy but important observation that we use tacitly throughout this section.

**Observation 4** – *Extension cores are unique up to isomorphism.*
– *For all sets* $\mathbf{A}, \mathbf{B}$ *of atoms, we have* $\mathsf{core}(\mathsf{extcore}(\mathbf{A}, \mathbf{B})) = \mathsf{extcore}(\mathbf{A}, \mathbf{B})$.

The first lemma in the sequence describes a crucial property of extension cores that will be used several times throughout the remainder of this section.

**Lemma 5** *An instance* $(\mathbf{A}, \mathbf{B}), \mathbf{D}, h$ *of* EXT *is a positive instance of* EXT *if and only if there exists a homomorphism* $h' \colon (\mathbf{A} \cup \mathbf{E}) \to \mathbf{D}$ *that extends* $h$*, where* $\mathbf{E}$ *is the set* $\mathsf{core}(\mathbf{A} \cup \mathbf{B} \cup \mathbf{Fix}_{\mathsf{dom}(\mathbf{A})}) \setminus \mathbf{Fix}_{\mathsf{dom}(\mathbf{A})}$ *of atoms from the definition of extension cores.*

*Proof* Solving the instance $(\mathbf{A}, \mathbf{B}), \mathbf{D}, h$ of EXT is equivalent to solving the instance $((\mathbf{A} \cup \mathbf{B} \cup \mathbf{Fix}_{\mathsf{dom}(\mathbf{A})}), (\mathbf{D} \cup h(\mathbf{Fix}_{\mathsf{dom}(\mathbf{A})}))$ of HOM (where $h(\mathbf{Fix}_{\mathsf{dom}(\mathbf{A})})$ denotes the set $\mathbf{Fix}_{\mathsf{dom}(\mathbf{A})}$ of atoms where all elements $a \in \mathsf{dom}(h)$ are replaced by $h(a)$). This in turn is equivalent to deciding the existence of a homomorphism $h' \colon (\mathbf{A} \cup \mathbf{E}) \to \mathbf{D}$ extending $h$. □

Next, we show the positive result, i.e., that the problem EXT$(\mathcal{C})$ can be solved efficiently if the treewidth of the extension cores in $\mathcal{C}$ is bounded.

**Lemma 6** *Let* $\mathcal{C}$ *be a class of pairs of sets of atoms such that the treewidth of* $\mathsf{extcore}(\mathbf{A}, \mathbf{B})$ *for all* $(\mathbf{A}, \mathbf{B}) \in \mathcal{C}$ *is bounded by some constant* $c$*. Then* EXT$(\mathcal{C})$ *is in* PTIME.

The proof of this result heavily relies on the notion of restricting a set $\mathbf{A}$ of atoms to a subset of its domain $\mathsf{dom}(\mathbf{A}) \setminus \mathcal{A}$ for $\mathcal{A} \subseteq \mathsf{dom}(\mathbf{A})$ already introduced in Section 2. Another important concept is the *projection of a pair* $(\mathbf{A}, \mathbf{B})$ *of sets of atoms under a mapping* $h$, where $\mathsf{dom}(h) \subseteq \mathsf{dom}(\mathbf{A})$, and the range of $h$ is (some subset of) $\mathsf{dom}(\mathbf{B})$. This describes the process of first replacing in $\mathbf{A}$ all values $a \in \mathsf{dom}(h)$ by $h(a)$, and then, for the resulting structure $\mathbf{A}'$, computing $\mathbf{A}' \setminus \mathsf{dom}(\mathbf{B})$ (w.l.o.g. assuming $\mathsf{dom}(\mathbf{A}) \cap \mathsf{dom}(\mathbf{B}) = \emptyset$). In the second step, for every new relation symbol $R^s$ in $\mathbf{A}' \setminus \mathsf{dom}(\mathbf{B})$ created from some relation symbol $R$, and every atom $R(\mathbf{a}) \in \mathbf{B}$ such that $\mathbf{a}$ is consistent with the values encoded in $s$ (recall that $s$ records the positions and their values that were removed from $R$ when creating $R^s$), we add an atom $R^s(\mathbf{b})$ to $\mathbf{B}$, where $\mathbf{b}$ is the projection of $\mathbf{a}$ to the remaining positions in $R^s$.

Both, restricting a set of atoms and the projection of pairs of atoms are well-known techniques. However, they occur in slightly different interpretations throughout the

literature. Thus, to avoid any ambiguities, we provide full formal definitions for both of them. Being very technical definitions for common notions, they are given in the Appendix.

The following observation not only is an immediate consequence of the above definitions, but also describes the intuition behind projecting a pair of sets of atoms and will be used in the proof of Lemma 6.

**Observation 5** *Let two sets* $\mathbf{Q}$, $\mathbf{D}$ *of atoms over a relational schema* $\sigma$ *and a mapping* $h\colon \mathcal{Q} \to \mathsf{dom}(\mathbf{D})$ *be given where* $\mathcal{Q} \subseteq \mathsf{dom}(\mathbf{Q})$*, and let* $\mathbf{Q}', \mathbf{D}'$ *be the projection of* $(\mathbf{Q}, \mathbf{D})$ *under* $h$*. Then there exists some extension* $h'\colon \mathbf{Q} \to \mathbf{D}$ *of* $h$ *if and only if there exists a homomorphism* $\hat{h}\colon \mathbf{Q}' \to \mathbf{D}'$ *(i.e., if* $(\mathbf{Q}', \mathbf{D}')$ *is a positive instance of* HOM*).*

We now have everything in place to prove Lemma 6.

*Proof (of Lemma 6)* Let $(\mathbf{A}, \mathbf{B}) \in \mathcal{C}$, $\mathbf{D}$, and $h$ be an instance of EXT($\mathcal{C}$). By Lemma 5, this problem is equivalent to asking for the existence of a homomorphism $h'\colon (\mathbf{A} \cup \mathbf{E}) \to \mathbf{D}$ that extends $h$, where $\mathbf{E} = \mathsf{core}(\mathbf{A} \cup \mathbf{B} \cup \mathbf{Fix}_{\mathcal{A}}) \setminus \mathbf{Fix}_{\mathcal{A}}$ and $\mathcal{A} = \mathsf{dom}(\mathbf{A})$. By Observation 5, this is equivalent to the instance $((\mathbf{A} \cup \mathbf{E})', \mathbf{D}')$ of HOM, where $((\mathbf{A} \cup \mathbf{E})', \mathbf{D}')$ is the projection of $((\mathbf{A} \cup \mathbf{E}), \mathbf{D})$ under $h$.

For $\mathbf{EXT} = \mathsf{extcore}(\mathbf{A}, \mathbf{B})$ and $\mathbf{F} = \mathbf{D}$, we next show that $((\mathbf{A} \cup \mathbf{E})', \mathbf{D}') = (\mathbf{EXT}', \mathbf{F}')$ where $(\mathbf{EXT}', \mathbf{F}')$ is the projection of $(\mathbf{EXT}, \mathbf{F})$ under $h$. First of all, observe that $\mathbf{D}' = \mathbf{F}'$ does not necessarily hold, since the content of $\mathbf{D}'$ and $\mathbf{F}'$ depends on the result of the projection with $(\mathbf{A} \cup \mathbf{E})'$ and $\mathbf{EXT}'$, respectively. However, if these left hand sides coincide, the equality $\mathbf{D}' = \mathbf{F}'$ obviously holds.

We thus show that $(\mathbf{A} \cup \mathbf{E})' = \mathbf{EXT}'$. First of all, we have that $\mathbf{A} \cup \mathbf{E} = \mathbf{A} \cup \Big(\mathsf{core}\big(\mathbf{A} \cup \mathbf{B} \cup \mathbf{Fix}_{\mathcal{A}}\big) \setminus \mathbf{Fix}_{\mathcal{A}}\Big) \subseteq (\mathbf{A} \cup \mathbf{B})$. Moreover, since $h\colon \mathbf{A} \to \mathbf{D}$, when computing the projection under $h$ we drop all atoms from $A$, and therefore $(\mathbf{A} \cup \mathbf{E})'$ does not contain any atoms derived from $\mathbf{A}$. It is thus safe to conclude that $(\mathbf{A} \cup \mathbf{E})' = \mathbf{E}'$, where $(\mathbf{E}', \mathbf{D}')$ is the projection of $(\mathbf{E}, \mathbf{D})$ under $h$.

Next, recall that $\mathsf{extcore}(\mathbf{A}, \mathbf{B}) = \mathbf{E} \setminus \mathcal{A}$. Thus, the only difference between $\mathbf{EXT}' = (\mathbf{E} \setminus \mathcal{A})'$ and $(\mathbf{A} \cup \mathbf{E})' = \mathbf{E}'$ is that in the first case, the restriction of $\mathbf{E}$ under $\mathcal{A}$ is computed, before the projection under $h$. However, since $\mathsf{dom}(h) = \mathsf{dom}(\mathbf{A}) = \mathcal{A}$, it can be easily checked that this results in the same structures, and thus $\mathbf{EXT}' = (\mathbf{A} \cup \mathbf{E})'$.

Now the treewidth of $\mathbf{EXT}$ is bounded by $c$, and therefore also the treewidth of $\mathbf{EXT}'$ (taking subgraphs does not increase the treewidth). As a result, the existence of a homomorphism $\mathbf{EXT}' \to \mathbf{F}'$ can be decided in polynomial time [7], which proves the lemma. □

With this showing the upper bounds in Theorem 3, we turn towards the lower bound, for which the following property will turn out to be important.

**Lemma 7** *Let* $\mathbf{A}$ *and* $\mathbf{B}$ *be sets of atoms and let* $\mathbf{C}$ *be the set of atoms* $\mathsf{core}(\mathbf{A} \cup \mathbf{B} \cup \mathbf{Fix}_{\mathsf{dom}(\mathbf{A})})$ *from the definition of the extension core. If* $h$ *is a homomorphism from* $\mathbf{C}$ *to itself, then* $h$ *is bijective.*

*Proof* Since $\mathbf{C}$ is, by definition, a core, any homomorphism from $\mathbf{C}$ to itself is an isomorphism. □

The next result shows that the lower bounds in Theorem 3 are optimal by using the characterization of tractable classes (for both, PTIME and FPT) of p-HOM($\mathcal{C}$) provided by Grohe [12].

**Lemma 8** *Let $\mathcal{C}$ be a decidable class of pairs of sets of atoms and let* $\mathsf{extcore}(\mathcal{C})$ *be the class of extension cores of the pairs in $\mathcal{C}$. Then* $\text{p-HOM}(\mathsf{extcore}(\mathcal{C})) \leq_{\mathsf{FPT}} \text{p-EXT}(\mathcal{C})$.

*Proof* Let $\mathcal{C}$ be a decidable class of pairs of sets of atoms, and let $(\mathbf{L}, \mathbf{T})$ be an instance of p-HOM($\mathsf{extcore}(\mathcal{C})$). We reduce this problem to an instance of p-EXT($\mathcal{C}$).

Borrowing from the notation of databases introduced in Section 2, for an arbitrary set $\mathbf{A}$ of atoms and a relation symbol $R$, throughout this proof we write $R^{\mathbf{A}}$ to denote the set of all atoms in $\mathbf{A}$ with relation symbol $R$.

In the first step, we compute some pair $(\mathbf{A}, \mathbf{B}) \in \mathcal{C}$ such that $\mathsf{extcore}(\mathbf{A}, \mathbf{B}) = \mathbf{L}$. By assumption, such a pair exists and, because $\mathcal{C}$ is decidable, can be computed. Next, we need to show that there exists a set $\mathbf{D}$ of atoms and a homomorphism $h\colon \mathbf{A} \to \mathbf{D}$ such that there exists a homomorphism $h'\colon (\mathbf{A} \cup \mathbf{B}) \to \mathbf{D}$ that is an extension of $h$ if and only if there exists a homomorphism from $\mathbf{L}$ to $\mathbf{T}$. However, by utilizing Lemma 5, we will work in a slightly different setting.

Let $\mathbf{E}$ be the set of atoms $\mathsf{core}(\mathbf{A} \cup \mathbf{B} \cup \mathbf{Fix}_{\mathsf{dom}(\mathbf{A})}) \setminus \mathbf{Fix}_{\mathsf{dom}(\mathbf{A})}$ from the definition of extension cores. By Lemma 5, the desired extension $h'$ of $h$ exists if and only if there exists a homomorphism $\hat{h}\colon (\mathbf{A} \cup \mathbf{E}) \to \mathbf{D}$ that extends $h$. Observe that the set $\mathbf{D}$ and homomorphism $h$ are still the same as above. We will work in the latter setting with $\mathbf{E}$ instead of $\mathbf{B}$ as this turns out to be easier.

We define the set $\mathbf{D}$ of atoms over the same schema as $\mathbf{E}$ as follows:

- The domain $\mathsf{dom}(\mathbf{D}) = \mathsf{dom}(\mathbf{T}) \times \mathsf{dom}(\mathbf{E})$, i.e., the elements represent pairs of elements from $\mathbf{T}$ and $\mathbf{E}$, respectively.
- For each relation symbol $R$ of arity $k$, and every $R(a_1, \ldots, a_k) \in \mathbf{E}$, the set $\mathbf{D}$ contains the following atoms:

  Let $\{i_1, \ldots, i_\ell\} \subseteq \{1, \ldots, k\}$ be all those positions of $(a_1, \ldots, a_k)$ such that $a_{i_j} \in \mathsf{dom}(\mathbf{A})$, and let $\{o_1, \ldots, o_m\} = \{1, \ldots, k\} \setminus \{i_1, \ldots, i_\ell\}$ be all those positions such that $a_{o_j} \notin \mathsf{dom}(\mathbf{A})$, i.e., $a_{o_j} \in \mathsf{dom}(\mathbf{B}) \setminus \mathsf{dom}(\mathbf{A})$. Let furthermore $R'$ be the relation symbol derived for $R(a_1, \ldots, a_k) \in \mathbf{E}$ when computing the projection $\mathbf{E} \setminus \mathsf{dom}(\mathbf{A}) = \mathsf{extcore}(\mathbf{A}, \mathbf{B})$.

  Now, for every pair $(\mathbf{d}_1, \mathbf{d}_2)$ of tuples $\mathbf{d}_1 = (d_{o_1}, \ldots, d_{o_m})$ with $R'(\mathbf{d}_1) \in \mathbf{T}$ and $\mathbf{d}_2 = (d_{i_1}, \ldots, d_{i_\ell}) \in \mathsf{dom}(\mathbf{T})^\ell$, add the atom $R((d_1, a_1), \ldots, (d_k, a_k))$ to $\mathbf{D}$. (Observe that by slight abuse of notation, in order to simplify the description we denote the positions in $\mathbf{d}_1$ and $\mathbf{d}_2$ according to the position in $R$ they originate from.) Thus, intuitively, we replace all domain elements from $\mathsf{dom}(\mathbf{A})$ with all possible combinations of elements from $\mathsf{dom}(\mathbf{T})$.

  These are all the tuples in $\mathbf{D}$.

It is worth pointing out that in case $R'$ is not part of the schema of $\mathbf{T}$ or $(R')^{\mathbf{T}}$ is empty, then by this definition also $R^{\mathbf{D}}$ is empty. The resulting instance will therefore be a simple "no" instance, because $R^{\mathbf{E}}$ is non-empty. However, in this case we also have that $(R')^{\mathbf{L}}$ is nonempty, and therefore also $(\mathbf{L}, \mathbf{T})$ is a trivial "no" instance.

Finally, we define the mapping $h\colon \mathsf{dom}(\mathbf{A}) \to \mathsf{dom}(\mathbf{D})$ as $h(a) = (d, a)$ for some arbitrary but fixed element $d \in \mathsf{dom}(\mathbf{T})$. Since $\mathbf{D}$ contains one atom $R((d_1, a_1), \ldots, (d_k, a_k))$ for every atom $R(a_1, \ldots, a_k) \in \mathbf{A}$ and every combination of $d_1, \ldots, d_k \in \mathsf{dom}(\mathbf{T})$, clearly $h$ is a homomorphism $h\colon \mathbf{A} \to \mathbf{D}$.

It remains to prove that there indeed exists a homomorphism $g\colon \mathbf{L} \to \mathbf{T}$ if and only if $h$ can be extended to a homomorphism $\hat{h}\colon \mathbf{E} \to \mathbf{D}$.

First assume that $g$ exists. Then define an extension $\hat{h}$ of $h$ to $\mathsf{dom}(\mathbf{E})$ as $\hat{h}(a) = (g(a), a)$ for all $a \in \mathsf{dom}(\mathbf{E}) \setminus \mathsf{dom}(\mathbf{A})$. The mapping $g$ is indeed defined on all these elements, since $\mathsf{dom}(\mathbf{E}) \setminus \mathsf{dom}(\mathbf{A}) = \mathsf{dom}(\mathsf{extcore}(\mathbf{A}, \mathbf{B})) = \mathsf{dom}(\mathbf{L})$ because $\mathsf{extcore}(\mathbf{A}, \mathbf{B}) = \mathbf{L}$. For $a \in \mathsf{dom}(\mathbf{A})$ we need not define $\hat{h}$ since $h$ is already defined on these elements, and $\hat{h}$ extends $h$. It now follows immediately from the construction of $\mathbf{D}$ that $\hat{h}$ is indeed the required homomorphism.

For the other direction, assume that $\hat{h}$ exists. First, observe that $\mathbf{D}$ projected onto the second component of its domain elements gives $\mathbf{E}$. Thus, $\hat{h}$ is a bijection in this second coordinate by Lemma 7. Let $\pi_2$ be the projection to the second coordinate. Then $\pi_2 \circ h$ is an automorphism of $\mathbf{E}$, and thus there is a $n \in \mathbb{N}$ such that $(\pi_2 \circ h)^n = id$ (where $id$ denotes the identity mapping). Consequently, w.l.o.g. we assume that $\pi_2 \circ h = id$. For every $a \in \mathsf{dom}(\mathbf{L}) = \mathsf{dom}(\mathsf{extcore}(\mathbf{A}, \mathbf{B}))$ define $g(a)$ to be the value $d$ such that $\hat{h}(a) = (d, a)$. Then again by definition of $\mathbf{D}$ it follows immediately that for all relation symbols $R$ and tuples $\mathbf{a} \in R^{\mathbf{L}}$ we have $g(\mathbf{a}) \in R^{\mathbf{T}}$.

Observing that all constructions can be done efficiently completes the proof. □

Theorem 3 now follows immediately. (1) ⇒ (2) follows from Lemma 6. The implication (2) ⇒ (3) follows immediately. Finally, if the treewidth of $\mathsf{extcore}(\mathcal{C})$ is not bounded, then p-HOM($\mathsf{extcore}(\mathcal{C})$) is not in FPT by Grohe [12]. Thus, by Lemma 8, the problem p-EXT($\mathcal{C}$) is not in FPT, which shows (3) ⇒ (1).

### 6.3 Tractability Conditions for Arbitrary wdPTs

With the notion of extension cores, we now have a tool to adapt tractability conditions (a) and (b) to also account for the core of sets of atoms, and to make use of the knowledge that when looking for extensions, the existence of certain mappings can be assumed. Recall the intuition described at the beginning of Section 6.1 and Example 10. In this situation, the maximality test towards a single node component can be easily expressed as an instance of EXT($\mathcal{C}$). More precisely, given a wdPT $(T, \lambda, \mathcal{X})$, a database $\mathbf{D}$, a subtree $T'$ of $T$, and a node component $\mathbf{S} \in \mathcal{NC}(t)$ for some node $t \in ch(T)$, testing if some mapping $\mu'$ can be extended to $\mathbf{S}$ is the instance $(\lambda(T'), \mathbf{S}), \mathbf{D}, \mu$ of EXT($\mathcal{C}$). One way to adapt tractability condition (a) – recall that intuitively this is the condition ensuring that the test for maximality is tractable – would thus be to associate with each class $\mathcal{P}$ of wdPTs a class $\mathcal{C}$ of all relevant pairs $(\lambda(T'), \mathbf{S})$, and to require EXT($\mathcal{C}$) to be in FPT. However, using the easy

characterization of Theorem 3, we state the refined variant of tractability condition (a) directly in terms of extension cores.

**Tractability condition (a'):** There is a constant $c$, such that for each $p = (T, \lambda, \mathcal{X}) \in \mathcal{P}$, the treewidth of $\mathsf{extcore}(\lambda(\mathsf{branch}(t)), \mathbf{S})$ is bounded by $c$ for all relevant nodes $t \in T$ (with $t \neq r$) and all $\mathbf{S} \in \mathcal{NC}(t)$.

Three comments are in order. First, observe that for the case of simple wdPTs, tractability condition (a') is equivalent to tractability condition (a). Second, note that unlike in the above discussion, condition (a') mentions $\mathsf{branch}(t)$ instead of $T'$. This is because the condition has to be satisfied for all subtrees $T'$ with $t \in ch(T')$. Among these, $\mathsf{branch}(t)$ is the minimal one. Thus, for all subtrees containing $\mathsf{branch}(t)$, the treewidth of $\mathsf{extcore}(\lambda(T'), \mathbf{S})$ is at most the treewidth of $\mathsf{extcore}(\lambda(\mathsf{branch}(t)), \mathbf{S})$. Third, recall that the intuition used above for motivating tractability condition (a') does not match the actual idea implemented in Algorithm 1. In fact, not testing maximality for one possible mapping on subtrees $T'$ after the other, but merging this test with finding mappings on the existential variables in $T'$ was actually a crucial step in the development of the algorithm. We will later describe the necessary changes to the algorithm in order for it to utilize the additional information given by the existence of a mapping that maps $\lambda(T')$ into the database, but first we reconsider tractability condition (b).

Recall that tractability condition (b) ensures that the CQs for finding suitable, maximal mappings on the existential variables are tractable. By Grohe [12], this is the case if and only if the core of the CQs has bounded treewidth. However, to account for the fact that for some free variables a mapping was provided as part of the input, when computing the core, these variables must be mapped onto themselves. This requirement is again naturally expressed in terms of extension cores.

**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 $T$ that contains $r$ and satisfies $V(T') \cap \mathsf{relv}(T) = V(T')$, the treewidth of $\mathsf{extcore}(\mathbf{Fix}_{\mathsf{fvar}(T')}, \lambda(T') \cup \bigcup_{i=1}^{n} \{\mathsf{cia}(\mathbf{S}_i)\})$ 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 $\{t_1, \ldots, t_n\} = ch(T') \cap \mathsf{relv}(T)$.

Analogously to tractability condition (a'), for simple wdPTs tractability condition (b') is equivalent to condition (b). However, there exist classes of non-simple wdPTs that satisfy conditions (a') and (b'), but not (a) and (b). An example for such a class of wdPTs was described in Example 10. Also, the two tractability conditions are independent of each other, as illustrated by the following example.

*Example 12* For a class of wdPTs that satisfies condition (a') but not (b'), consider the class of wdPTs containing the wdPTs from Fig. 3 for all $k \geq 1$. For any subtree containing the node $t_2$, condition (b') is not satisfied because of the $k$-clique of $yclique_{ij}(y_i, y_j)$ atoms. However, condition (a') is satisfied, since all variables in this clique are interface variables.

For the opposite case, recall the wdPT in Fig. 5, but assume that the atom $emp(y_1)$ was part of $\lambda(r)$ instead of $\lambda(t_1)$. Clearly, the core of $\lambda(T)$ remains unchanged, and

thus of bounded treewidth. Also the set of atoms *boss_of* remains part of a single node component $\mathbf{S}$. Now when computing the extension core of $(\lambda(r), \mathbf{S})$, $y_1$ is part of $\lambda(r)$, and thus must be mapped onto itself. As a result, all $y_i$ must be mapped onto themselves. After removing the variables from $\lambda(r)$, a $(k-1)$-clique remains, and thus condition (a') is not satisfied.

Conditions (a') and (b') therefore describe a proper extended set of classes of wdPTs. Next, we discuss why query evaluation is indeed tractable for these classes.

In fact, Algorithm 1 already describes a correct FPT algorithm even for classes of arbitrary wdPTs satisfying conditions (a') and (b'). The only change necessary does not happen directly within Algorithm 1, but in the computation of $stop(\mathbf{S}, \mathbf{D})$ (for some node component $\mathbf{S}$ and database $\mathbf{D}$ in line 6). So far, given an instance $p, \mathbf{D}, \mu$ of p-EVAL($\mathcal{P}$), the content of $stop(\mathbf{S}, \mathbf{D})$ was computed as follows: for every possible mapping $\nu$ on $\mathcal{I}(\mathbf{S}, t)$ compatible with $\mu$, check if it can be extended to a homomorphism $\nu' \colon \mathbf{S} \to \mathbf{D}$. If this is not the case, include $\nu_{|\mathcal{I}^\exists(\mathbf{S},t)}$ in $stop(\mathbf{S}, \mathbf{D})$. The only change is that instead of testing for a homomorphism from $\mathbf{S}$ into $\mathbf{D}$, we now decide whether to add the tuple into $stop(\mathbf{S}, \mathbf{D})$ based on the non-existence of a homomorphism $h' \colon \mathsf{extcore}(\lambda(\mathsf{branch}(t)), \mathbf{S}) \to \mathbf{D}$ extending $h$. I.e., formulated in terms of the problem EXT($\mathcal{C}$), we have $h_{|\mathcal{I}^\exists(\mathbf{S},t)} \in stop(\mathbf{S}, \mathbf{D})$ if $(\emptyset, \mathsf{extcore}(\lambda(\mathsf{branch}(t)), \mathbf{S}))$, $h, \mathbf{D}$ is a negative instance of EXT($\mathcal{C}$).

Note that this new definition does not necessarily give the same sets $stop(\mathbf{S}, \mathbf{D})$ as we would get under the original definition, as demonstrated by the following very simple example.

*Example 13* Consider the wdPTs $p$ as depicted in Fig. 6. For the node component $\mathbf{S} = \{a(y_3, y_2)\}$ in $t_1$, we get $\mathsf{cia}(\mathbf{S}) = R(y_2)$, and $\mathsf{extcore}(\lambda(r), \mathbf{S}) = \emptyset$. Then, following the definition in Section 4, we get $stop(\mathbf{S}, \mathbf{D}) = \{R(0), R(1)\}$ since mapping $y_2$ to either of these values does not map $a(y_3, y_2)$ into $\mathbf{D}$. In contrast, when working with $\mathsf{extcore}(\lambda(r), \mathbf{S})$, we get $stop(\mathbf{S}, \mathbf{D}) = \emptyset$.

The reason for this is that replacing $\mathbf{S}$ by $\mathsf{extcore}(\lambda(\mathsf{branch}(t)), \mathbf{S})$ implies that the mapping $h$ that shall be extended to $\mathbf{S}$ is (or can be extended to) a homomorphism $\mathsf{branch}(t) \to \mathbf{D}$. Obviously, in the example this is not the case for any mapping that maps $y_2$ to either 0 or 1.

As we will show below, this difference has no effect on the output of Algorithm 1, since the additional values in $stop(\mathbf{S}, \mathbf{D})$ according to the original definition are never involved in any solution anyway.

By combining all of this, we get the following result as a proper extension of Lemma 1.

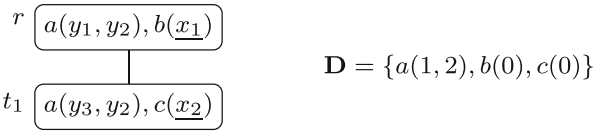


**Fig. 6** The wdPT $p$ and database instance $\mathbf{D}$ from Example 13. The free variables are underlined

**Theorem 6** *Let $\mathcal{P}$ be a decidable class of wdPTs. If tractability conditions (a') and (b') hold for $\mathcal{P}$, then* p-EVAL*($\mathcal{P}$) can be solved in* FPT.

*Proof* First of all, observe that under these conditions Algorithm 1 is still in FPT: with the exception of evaluating the query $q$ (line 7) and computing the set $stop(\mathbf{S}, \mathbf{D})$ (line 6), all the arguments from Section 4 still apply.

For deciding $\mu(\mathbf{x}) \in q(\mathbf{D}')$ in line 7, observe that this is equivalent to deciding the instance $(\emptyset, \lambda(T') \cup \{\mathsf{cia}(\mathbf{S}_1), \ldots, \mathsf{cia}(\mathbf{S}_n)\}), \mu, \mathbf{D}'$ of EXT($\mathcal{C}$), which is in FPT because of Theorem 3 and the fact that tractability condition (b') is satisfied.

For computing $stop(\mathbf{S}, \mathbf{D})$, first of all observe that since the component interface atoms contain unique relation symbols and no variables from $\mathsf{fvar}(T')$, they occur unchanged in the core. Thus, the component width of $\mathcal{P}$ is bounded by some constant, and therefore there exist at most polynomial many candidate mappings to be included in $stop(\mathbf{S}, \mathbf{D})$. Furthermore, testing each of these mappings is in FPT. To see this, recall that testing requires to decide an instance $(\emptyset, \mathsf{extcore}(\lambda(\mathsf{branch}(t)), \mathbf{S})), h, \mathbf{D}$ of EXT($\mathcal{C}$). By Theorem 3, this decision is in FPT if $\mathsf{extcore}(\emptyset, \mathsf{extcore}(\lambda(\mathsf{branch}(t)), \mathbf{S})) = \mathsf{extcore}(\lambda(\mathsf{branch}(t)), \mathbf{S})$ has bounded treewidth, which is guaranteed by tractability condition (a').

It thus remains to show the correctness of the algorithm, which follows by the same arguments as in Section 4. Therefore, the only point that needs to be shown is that the presented computation of $stop(\mathbf{S}, \mathbf{D})$ is correct. First of all, $stop(\mathbf{S}, \mathbf{D})$ according to the definition via the extension core being a subset of $stop(\mathbf{S}, \mathbf{D})$ according to the original definition in Section 4 follows immediately. For the opposite direction, where we have already shown that this is not necessarily the case, we show that the additional mappings in $stop(\mathbf{S}, \mathbf{D})$ according to the original definition have no effect on the result of Algorithm 1.

Towards this, first assume that for a mapping $\nu \in stop(\mathbf{S}, \mathbf{D})$ according to the definition in Section 4, there exists an extension $\nu' \colon \lambda(\mathsf{branch}(t)) \to \mathbf{D}$. Then, since all variables shared between $\mathbf{S}$ and $\lambda(\mathsf{branch}(t))$ occur in $\mathcal{I}(\mathbf{S}, t)$, we have that $\nu$ can be extended to a homomorphism $\mathbf{S} \to \mathbf{D}$ if and only if $(\lambda(\mathsf{branch}(t)), \mathbf{S}), \nu', \mathbf{D}$ is a positive instance of EXT($\mathcal{C}$). I.e., for such $\nu$ we still have $\nu \in stop(\mathbf{S}, \mathbf{D})$ by the new definition, and thus the test is correct. Next, assume that there exists no such extension $\nu'$. In this case, in line 7 of the algorithm, $q(\mathbf{D}') = q(\mathbf{D}' \setminus \{R_i(\nu(\mathbf{v}_i))\})$, since $\lambda(\mathsf{branch}(t))$ is contained in the body of the query, and thus $\nu$ cannot be part of any solution mapping. Hence, $\nu \notin stop(\mathbf{S}, \mathbf{D})$ still gives a correct solution. □

## 7 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 of Theorem 2 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., Grohe et al. [13], Chen [4], and Chen and Dalmau [5] did in similar contexts.

As we have seen, by incorporating cores, we can also characterize larger tractable classes in the non-simple case, and thus again for well-designed pattern trees resulting from the translation of well-designed SPARQL queries. However, in this case we do not get a dichotomy result.

Let us mention one major stumbling block towards a characterization of non-simple well-designed pattern trees with projections: In the proof of Lemma 3, we have used a reduction from quantified conjunctive queries. Unfortunately, the tractable classes for the non-simple fragment for that problem are not well understood which limits our result to simple queries since we are using the respective results by Chen and Dalmau [5]. Note that we might have been able to give a more fine-grained result in sorted logics by using the work of Chen and Marx [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.

**Funding Information** Open access funding provided by TU Wien (TUW).



## Appendix: Additional Definitions

In Sections 2 and 6, we introduced the notions of restricting a set $\mathbf{A}$ of atoms to some subset $\mathcal{A} \subseteq \mathsf{dom}(\mathbf{A})$ and projecting a pair of sets of atoms under some mapping. As mentioned in Section 6, below we provide a full formal definition of these notion to avoid any ambiguities a reader may have.

**Restricting a Set of Atoms ($\mathbf{S} \setminus \mathcal{V}$)** A complete definition was already given in Section 2. In the following, we clarify the names of the newly introduced relation symbols $R^s$. For a set $\mathbf{S}$ of atoms over some relational schema $\sigma$ and $\mathcal{V} \subseteq \mathsf{dom}(\mathbf{S})$, $\mathbf{S} \setminus \mathcal{V}$ contains the following set of atoms over the relational schema $\sigma'$, which is defined implicitly by the newly introduced atoms described below. For every atom $R(a_1, \ldots, a_k) \in \mathbf{S}$, let $o_1, \ldots, o_\ell$ be those positions of $(a_1, \ldots, a_k)$ that do not contain values from $\mathcal{V}$, i.e., $a_{o_j} \notin \mathcal{V}$ for $1 \leq j \leq \ell$, and let $\{i_1, \ldots, i_m\} = \{1, \ldots, k\} \setminus \{o_1, \ldots, o_\ell\}$ be those positions such that $a_{i_j} \in \mathcal{V}$ for $1 \leq j \leq m$. Then the set $\mathbf{S} \setminus \mathcal{V}$ contains the atom $R^{i_1=a_{i_1},\ldots,i_m=a_{i_m}}(a_{o_1}, \ldots, a_{o_\ell})$. These are all the atoms in $\mathbf{S} \setminus \mathcal{V}$.

**Projecting a Pair of Sets of Atoms Under a Mapping** Let $(\mathbf{Q}, \mathbf{D})$ be a pair of sets of atoms over the same relational schema $\sigma$ and let $h$ be a mapping on (a subset of) $\mathsf{dom}(\mathbf{Q})$. We define the *restriction of* $(\mathbf{Q}, \mathbf{D})$ *under* $h$ *as the pair* $(\mathbf{Q}', \mathbf{D}')$ of sets of atoms over the schema $\sigma'$ as follows (the relational schema $\sigma'$ is defined implicitly).

We start by defining $\mathbf{Q}'$. For every relation symbol $R \in \sigma$ and every atom $R(a_1, \ldots, a_k) \in \mathbf{Q}$, we distinguish two cases.

- If $\{a_1, \ldots, a_k\} \subseteq \mathsf{dom}(h)$ and $R(h(a_1), \ldots, h(a_k)) \in \mathbf{D}$, then just ignore $R(a_1, \ldots, a_k)$ (i.e., no atom derived from $R(a_1, \ldots, a_k)$ occurs in $\mathbf{Q}'$).
- Otherwise, let $\{i_1, \ldots, i_\ell\} \subseteq \{a_1, \ldots, a_k\}$ be those positions of $(a_1, \ldots, a_k)$ such that $a_{i_j} \in \mathsf{dom}(h)$ for $1 \leq j \leq \ell$, and $\{o_1, \ldots, o_m\} = \{1, \ldots, k\} \setminus \{i_1, \ldots, i_\ell\}$ those positions such that $a_{o_j} \notin \mathsf{dom}(h)$. Then $\mathbf{Q}'$ contains the atom $R^{i_1=h(a_{i_1}),\ldots,i_\ell=h(a_{i_\ell})}(a_{o_1}, \ldots, a_{o_m})$.

These are all atoms in $\mathbf{Q}'$.

Observe that the second case includes the possibility that $\ell = k$, i.e., that $a_i \in \mathsf{dom}(h)$ for all $1 \leq i \leq k$, but that $R(h(a_1), \ldots, h(a_k)) \notin \mathbf{D}$. In this case, the result is a 0-ary relation symbol $R^{1=h(a_1),\ldots,k=h(a_k)}$, and $R^{1=h(a_1),\ldots,k=h(a_k)}() \in \mathbf{Q}'$.

Next we define $\mathbf{D}'$. For every relation symbol $R^{i_1=b_1,\ldots,i_\ell=b_\ell} \in \sigma'$ introduced by the definition of $\mathbf{Q}'$ and every atom $R^{i_1=b_1,\ldots,i_\ell=b_\ell}(a_1,\ldots,a_m) \in \mathbf{Q}'$, let $k$ be the arity of the original relation symbol $R$ from $\sigma$. Then the positions $1,\ldots,m$ in $R^{i_1=b_1,\ldots,i_\ell=b_\ell}$ correspond to positions $j_1,\ldots,j_m$ in $R$. In fact, $\{j_1,\ldots,j_m\} = \{1,\ldots,k\} \setminus \{i_1,\ldots,i_\ell\}$. For each atom $R(\mathbf{a}) \in \mathbf{D}$ with $a_{i_r} = b_r$ for all $1 \leq r \leq \ell$, the set $\mathbf{D}'$ contains an atom $R^{i_1=b_1,\ldots,i_\ell=b_\ell}(a_{j_1},\ldots,a_{j_m})$.

We need to take care of one special case: Assume that, for a pair $(\mathbf{Q},\mathbf{D})$ and a homomorphism $h$ such that $\mathbf{Q}$ is already a projection of some set $\mathbf{L}$ of atoms under a set $V \subseteq \mathsf{dom}(\mathbf{L})$, we want to get the projection of $(\mathbf{Q},\mathbf{D})$ under $h$. I.e. the schema of $\mathbf{Q}$ already contains relation symbols of the form $R^{i_1=b_1,\ldots,i_\ell=b_\ell}$. If for any of the $i_j$ $(1 \leq j \leq \ell)$ it is the case that $b_j \in \mathsf{dom}(h)$, then in the resulting schema we replace $b_j$ in the name of the resulting atom by $h(b_j)$. In certain situations, this renaming of the relation symbols will ensure the resulting sets of atoms to be over the same relational schema, which is a prerequisite for finding homomorphisms.

## References


1. Arenas, M., Pérez, J.: Querying semantic web data with SPARQL. In: Proceedings of the PODS 2011, pp. 305–316. ACM (2011)
2. Arenas, M., Diaz, G.I., Kostylev, E.V.: Reverse engineering SPARQL queries. In: Proceedings of WWW 2016, pp. 239–249. ACM (2016)
3. Barceló, P., Kröll, M., Pichler, R., Skritek, S.: Efficient evaluation and static analysis for well-designed pattern trees with projection. ACM Trans. Database Syst. **43**(2), 8:1–8:44 (2018)
4. Chen, H.: The tractability frontier of graph-like first-order query sets. In: Henzinger, T.A., Miller, D. (eds.) Proceedings of CSL-LICS 2014, pp. 31:1–31:9. ACM (2014)
5. Chen, H., Dalmau, V.: Decomposing quantified conjunctive (or disjunctive) formulas. In: Proceedings of LICS 2012, pp. 205–214. IEEE Computer Society (2012)
6. Chen, H., Marx, D.: Block-sorted quantified conjunctive queries. In: Fomin, F.V., Freivalds, R., Kwiatkowska, M.Z., Peleg, D. (eds.) Proceedings of ICALP 2013. Lecture Notes in Computer Science, vol. 7966, pp. 125–136. Springer (2013)
7. Dalmau, V., Kolaitis, P., Vardi, M.: Constraint satisfaction, bounded treewidth, and finite-variable logics. In: International Conference on Principles and Practice of Constraint Programming, vol. 2002, pp. 310–326 (2002)
8. Durand, A., Mengel, S.: Structural tractability of counting of solutions to conjunctive queries. Theory Comput. Syst. **57**, 1202–1249 (2015)
9. Flum, J., Grohe, M.: Parameterized Complexity Theory Texts in Theoretical Computer Science. An EATCS Series. Springer, Berlin (2006)
10. Grohe, M.: The parameterized complexity of database queries. In: Buneman, P. (ed.) Proceedings of PODS 2001, pp. 82–92. ACM (2001)
11. Grohe, M.: Parameterized complexity for the database theorist. SIGMOD Record **31**(4), 86–96 (2002)
12. Grohe, M.: The complexity of homomorphism and constraint satisfaction problems seen from the other side. J. ACM **54**(1), 1:1–1:24 (2007)
13. Grohe, M., Schwentick, T., Segoufin, L.: When is the evaluation of conjunctive queries tractable? In: Vitter, J.S., Spirakis, P.G., Yannakakis, M. (eds.) Proceedings of STOC 2001, pp. 657–666. ACM (2001)
14. Kaminski, M., Kostylev, E.V.: Beyond well-designed SPARQL. In: Proceedings of ICDT 2016. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, LIPIcs, vol 48, pp. 5:1–5:18 (2016)
15. Kostylev, E.V., Reutter, J.L., Ugarte, M.: CONSTRUCT queries in SPARQL. In: Proceedings of ICDT 2015, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, LIPIcs, vol. 31, pp. 212–229 (2015)

16. Kröll, M., Pichler, R., Skritek, S.: On the complexity of enumerating the answers to well-designed pattern trees. In: Proceedings of ICDT 2016. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, LIPIcs, vol. 48, pp. 22:1–22:18 (2016)
17. Letelier, A., Pérez, J., Pichler, R., Skritek, S.: Static analysis and optimization of semantic web queries. ACM Trans. Database Syst. **38**(4), 25 (2013)
18. Marx, D.: Tractable hypergraph properties for constraint satisfaction and conjunctive queries. In: Schulman, L.J. (ed.) Proceedings of STOC 2010, pp. 735–744. ACM (2010)
19. Mengel, S., Skritek, S.: Characterizing tractability of simple well-designed pattern trees with projection. In: Proceedings of ICDT 2019, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, LIPIcs, vol. 127, pp. 20:1–20:18 (2019)
20. Papadimitriou, C.H., Yannakakis, M.: On the complexity of database queries. J. Comput. Syst. Sci. **58**(3), 407–427 (1999)
21. Pérez, J., Arenas, M., Gutierrez, C.: Semantics and complexity of SPARQL. ACM Trans. Database Syst. 34(3) (2009)
22. Picalausa, F., Vansummeren, S.: What are real SPARQL queries like? In: Proceedings of SWIM 2011, p. 7. ACM (2011)
23. Pichler, R., Skritek, S.: Containment and equivalence of well-designed SPARQL. In: Proceedings of PODS 2014, pp. 39–50. ACM (2014)
24. Romero, M.: The tractability frontier of well-designed SPARQL queries. In: Proceedings of PODS 2018, pp. 295–306. ACM (2018)