Beyond Well-designed SPARQL

Mark Kaminski[1] and Egor V. Kostylev[2]

1 **Department of Computer Science, University of Oxford, UK**
2 **Department of Computer Science, University of Oxford, UK**

Abstract

SPARQL is the standard query language for RDF data. The distinctive feature of SPARQL is the OPTIONAL operator, which allows for partial answers when complete answers are not available due to lack of information. However, optional matching is computationally expensive – query answering is PSPACE-complete. The well-designed fragment of SPARQL achieves much better computational properties by restricting the use of optional matching – query answering becomes coNP-complete. However, well-designed SPARQL captures far from all real-life queries – in fact, only about half of the queries over DBpedia that use OPTIONAL are well-designed.

In the present paper, we study queries outside of well-designed SPARQL. We introduce the class of weakly well-designed queries that subsumes well-designed queries and includes most common meaningful non-well-designed queries: our analysis shows that the new fragment captures about 99% of DBpedia queries with OPTIONAL. At the same time, query answering for weakly well-designed SPARQL remains coNP-complete, and our fragment is in a certain sense maximal for this complexity. We show that the fragment's expressive power is strictly in-between well-designed and full SPARQL. Finally, we provide an intuitive normal form for weakly well-designed queries and study the complexity of containment and equivalence.

1998 ACM Subject Classification H.2.3 Languages – Query languages

Keywords and phrases RDF, Query languages, SPARQL, Optional matching

Digital Object Identifier 10.4230/LIPIcs.ICDT.2016.5

1 Introduction

The Resource Description Framework (RDF) [29, 17, 21] is the W3C standard for representing linked data on the Web. RDF models information in terms of labeled graphs consisting of triples of resource identifiers (IRIs). The first and last IRIs in such a triple, called *subject* and *object*, represent entity resources, while the middle IRI, called *predicate*, represents a relation between the two entities.

SPARQL [35, 20] is the default query language for RDF graphs. First standardised in 2008 [35], SPARQL is now recognised as a key technology for the Semantic Web. This is witnessed by a recently adopted new version of the standard, SPARQL 1.1 [20], as well as by active development of SPARQL query engines in academia and the industry, for instance, as part of the systems AllegroGraph [1], Apache Jena [2], Sesame [3], or OpenLink Virtuoso [4].

In recent years, SPARQL has been subject to a substantial amount of theoretical research, based on the foundational work by Pérez et al. [30, 31]. In particular, we now know much about evaluation [36, 28, 6, 32, 25, 23, 7, 22], optimisation [27, 33, 16, 15, 12, 24], federation [14, 13], expressive power [5, 34, 25, 39], and provenance tracking [18, 19] for queries from various fragments and extensions of SPARQL. These studies have had a great impact in the community, in fact influencing the evolution of SPARQL as a standard.

A distinctive feature of SPARQL as compared to SQL is the OPTIONAL operator (abbreviated as OPT in this paper). This operator was introduced to *"not reject (solutions)*

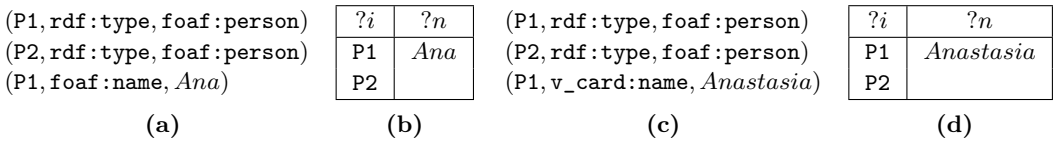

Figure 1 **(a)** Graph G; **(b)** answers to query (1) over G; **(c)** graph G'; and **(d)** answers over G'.

because some part of the query pattern does not match" [35]. For instance, consider the SPARQL query

$$\mathsf{SELECT}\ ?i, ?n\ \mathsf{WHERE}\ (?i, \texttt{rdf:type}, \texttt{foaf:person})\ \mathsf{OPT}\ (?i, \texttt{foaf:name}, ?n), \tag{1}$$

which retrieves all person IDs from the graph together with their names; names, however, are optional – if the graph does not contain information about the name of a person, the person ID is still retrieved but the variable $?n$ is left undefined in the answer. For instance, query (1) has two answers over the graph G in Figure 1(a), where the second answer is partial (see Figure 1(b)). However, if we extend G with a triple supplying a name for P2, the second answer will include this name.

The OPT operator accounts in a natural way for the open world assumption and the fundamental incompleteness of the Web. However, evaluating queries that use OPT is computationally expensive – Pérez et al. [31] showed PSPACE-completeness of SPARQL query evaluation, and Schmidt et al. [36] refined this result by proving PSPACE-hardness even for queries using no operators besides OPT. This is not surprising given that SPARQL queries are equivalent in expressive power to first-order logic queries, and translations in both directions can be done in polynomial time [5, 34, 25].

This spurred a search for restrictions on the use of OPT that would ensure lower complexity of query evaluation. It was also recognised that queries that are difficult to evaluate are often unintuive. For instance, they may produce less specified answers (i.e., answers with fewer bound variables) as the graph over which they are evaluated grows larger.

Perez et al. [31] introduced the *well-designed* fragment of SPARQL queries by imposing a syntactic restriction on the use of variables in OPT-expressions. Roughly speaking, each variable in the optional (i.e., right) argument of an OPT-expression should either appear in the mandatory (i.e., left) argument or be globally fresh for the query, i.e., appear nowhere outside of the argument. Well-designed queries have lower complexity of query evaluation – the problem is coNP-complete (provided all the variables in the query are selected). Moreover, such queries have a more intuitive behaviour than arbitrary SPARQL queries; in particular, they enjoy the monotonicity property that we observed for query (1): each partial answer over a graph can potentially be extended to undefined variables if the graph is completed with the missing information, and the more information we have the more specified are the answers. Well-designed queries can be efficiently transformed to an intuitive normal form allowing for a transparent graphical representation of queries as trees [27, 33]. Hence, many recent studies concentrate partially [27, 25, 23, 37, 38] or entirely [33] on well-designed queries.

Such a success of well-designed queries may lead to the impression that non-well-designed SPARQL queries are just a useless side effect of the early specification. But is this impression justified by the use of SPARQL in practice? To answer this question, a comprehensive analysis of real-life queries is required. We are aware of two works that analyse the distribution of operators in SPARQL queries asked over DBpedia [32, 9]. Both studies show that OPT is used in a non-negligible amount of practical queries. However, only Picalausa and

Vansummeren [32] go further and analyse how many of these queries are well-designed; and the result is quite interesting – well-designed queries make up only about half of all queries with OPT. In other words, well-designed queries are common, but by far not exclusive.

The main goal of this paper is to investigate SPARQL queries beyond the well-designed fragment. We wanted to see if the well-designedness condition could be extended so as to include most practical queries while preserving good computational properties. The main result of our study is very positive – we identified a new fragment of SPARQL queries, called *weakly well-designed* queries, that covers about 99% of queries over DBpedia and has the same complexity of query evaluation as the well-designed fragment. We also show that our fragment is in a sense maximal for this complexity.

We next describe our results and techniques in more detail. Our first step was to identify most typical real-life queries that are not well-designed. We analysed the USEWOD2013 [10] and USEWOD2014 [11] query logs for DBpedia 3.8 and 3.9 and found two interesting types of non-well-designed queries. The first type is exemplified by the following query:

$$\begin{array}{l}\mathsf{SELECT}\ ?i, ?n\ \mathsf{WHERE}\\ ((?i, \texttt{rdf:type}, \texttt{foaf:person})\ \mathsf{OPT}\ (?i, \texttt{foaf:name}, ?n))\ \mathsf{OPT}\ (?i, \texttt{v\_card:name}, ?n).\end{array} \tag{2}$$

This query is clearly not well-designed because variable $?n$, binding the name of a person, appears in two different unrelated optional parts. Let us analyse answers to this query over different graphs. On graph G in Figure 1(a) the result is exactly the same as for query (1), shown in Figure 1(b), simply because the IRI `v_card:name` is not present in G, and so cannot be matched against the second optional part of the query. Similarly, on graph G' in Figure 1(c), where the source of the name and the name itself are different, the result is as in Figure 1(d). In this case, the first optional part in the query does not match anything in the graph so the variable $?n$ is left unbound at this point; then the second optional is matched, and the variable is assigned with the name from `v_card`. More interestingly, query (2) evaluated over the graph $G \cup G'$ once again yields the result in Figure 1(b). Indeed, in this case, the first optional part has a match again and $?n$ is assigned the value *Ana*; then, this variable is already bound and there is no match for the second optional part that agrees with this value, meaning that the alternative `v_card` name is disregarded by the query. To summarise, query (2) is once again looking for person IDs and, optionally, their names. Now, however, names are collected from two different sources, `foaf` and `v_card`, where the first source is given preference over the second (maybe because it is considered more reliable or more informative, or for some other reason). In other words, if we know the `foaf` name of a person, it is returned as part of the answer regardless of his `v_card` name; however, if there is no `foaf` name, then the `v_card` name is also acceptable and should be returned; variable $?n$ is left unbound only if the name cannot be extracted from either source.

Of course, preference patterns encountered in real-life queries are often more complex. Still, in most cases they do not increase the complexity of query evaluation.

Our second example query is as follows:

$$\begin{array}{l}\mathsf{SELECT}\ ?i, ?n\ \mathsf{WHERE}\ ((?i, \texttt{rdf:type}, \texttt{foaf:person})\ \mathsf{OPT}\ (?i, \texttt{foaf:name}, ?n))\\ \qquad\qquad \mathsf{FILTER}\ (\neg \mathit{bound}(?n) \vee \neg(?n = \mathit{Ana})).\end{array} \tag{3}$$

The query uses FILTER, a standard SPARQL operator that admits only answers conforming to a specified constraint. Again, this query is not well-designed because the FILTER constraint mentions the variable $?n$, which occurs in the optional part of the query but not in the mandatory part. However, the intention of the query is quite clear: it searches for people whose names are not known to be *Ana*, including people whose names are unknown.

This use of FILTER is in fact very common in real-life queries. Moreover, it is intuitive as long as FILTER is essentially the outermost operator in the query, as it is in our example. In all such cases, however, FILTER cannot lead to an increase in complexity.

Having isolated these typical uses of non-well-designedness, we identify a new fragment of SPARQL that (a) includes all queries of the above two types, (b) subsumes well-designed queries, and (c) has the same complexity of query evaluation as well-designed queries. We call such queries *weakly well-designed*. They are the maximal fragment without structural restrictions on conjunctive blocks and filter conditions that has the above properties. Our analysis shows that about 99% of DBpedia queries with OPT are weakly well-designed.

Besides low complexity of query evaluation, we establish a few more useful properties of weakly well-designed queries, which are summarised in the following outline of the paper. After introducing the syntax and semantics of SPARQL in Section 2, we formally define our new fragment in Section 3. In Section 4, we show that, similarly to the well-designed case, weakly well-designed queries can be transformed to an intuitive normal form, which allows for a natural graphical representation as *constraint pattern trees*. Using this representation, in Section 5, we formally show that the step from well-designed to weakly well-designed queries does not increase complexity of query evaluation; minimal relaxations of weak well-designedness, however, already lead to a complexity jump. In Section 6, we compare the expressive power of our fragment (and its extensions with additional operators) with well-designed queries and unrestricted SPARQL queries; in particular, we show that the expressivity of weakly well-designed queries lies strictly in-between well-designed and unrestricted queries. In Section 7, we study static analysis problems for weakly well-designed queries and establish Π_2^p-completeness of equivalence, containment, and subsumption. Finally, in Section 8, we detail our analysis of DBpedia logs.

2 SPARQL Query Language

We begin by formally introducing the syntax and semantics of SPARQL that we adopt in this paper. Our formal setup mostly follows [31], which has some differences from the W3C specification [35, 20]; in particular, we use two-placed OPT and two-valued FILTER (conditional OPT and errors in FILTER evaluation as in the standard are expressible in our formalisation [5]), and adopt set semantics, leaving multiset answers for future work.

RDF Graphs. An RDF graph is a labeled graph where nodes can also serve as edge labels. Formally, let $\mathbf{I}$ be a set of *IRIs*. Then an *RDF triple* is a tuple (s, p, o) from $\mathbf{I} \times \mathbf{I} \times \mathbf{I}$, where s is called *subject*, p *predicate*, and o *object*. An *RDF graph* is a finite set of RDF triples.

SPARQL Syntax. Let $\mathbf{X}$ be an infinite set $\{?x, ?y, \ldots\}$ of *variables*, disjoint from $\mathbf{I}$. *Filter constraints* are conditions of the form

- $\top$, $?x = u$, $?x = ?y$, or $\mathit{bound}(?x)$ for $?x, ?y$ in $\mathbf{X}$ and $u \in \mathbf{I}$ (*atomic constraints*),
- $\neg R_1$, $R_1 \wedge R_2$, or $R_1 \vee R_2$ for filter constraints R_1 and R_2.

A *basic pattern* is a set of triples from $(\mathbf{I} \cup \mathbf{X}) \times (\mathbf{I} \cup \mathbf{X}) \times (\mathbf{I} \cup \mathbf{X})$. Then, SPARQL *(graph) patterns* P are defined by the grammar

$$P ::= B \mid (P \,\mathsf{AND}\, P) \mid (P \,\mathsf{OPT}\, P) \mid (P \,\mathsf{UNION}\, P) \mid (P \,\mathsf{FILTER}\, R),$$

where B ranges over basic patterns and R over filter constraints. Additionally, we require all filter constraints to be *safe*, that is, $\mathsf{vars}(R) \subseteq \mathsf{vars}(P)$ for every pattern $(P \,\mathsf{FILTER}\, R)$, where $\mathsf{vars}(S)$ is the set of all variables in S (which can be pattern, constraint, etc.) When

needed, we distinguish between patterns by their top-level operator (e.g., OPT-pattern or FILTER-pattern). The set of all triples in basic patterns of a pattern P is denoted $\mathsf{triples}(P)$.

We write $\mathcal{U}$ for the set of all patterns. We also distinguish the fragment $\mathcal{P}$ of $\mathcal{U}$ that consists of all UNION-*free* patterns, i.e., patterns that do not use the UNION operator.

Projection is realised in SPARQL by means of *queries with select result form*, or *queries* for short, which are expressions of the form

$$\mathsf{SELECT}\ X\ \mathsf{WHERE}\ P, \tag{4}$$

where X is a set of variables and P is a graph pattern. We write $\mathcal{S}$ for the set of all queries.

Note that every pattern P can be seen as a query of the form (4) where $X = \mathsf{vars}(P)$. Hence, all definitions that refer to "queries" implicitly extend to patterns in the obvious way.

SPARQL Semantics. The semantics of graph patterns is defined in terms of *mappings*, that is, partial functions from variables to IRIs. The *domain* $\mathsf{dom}(\mu)$ of a mapping μ is the set of variables on which μ is defined. Two mappings μ_1 and μ_2 are *compatible* (written $\mu_1 \sim \mu_2$) if $\mu_1(?x) = \mu_2(?x)$ for all variables $?x \in \mathsf{dom}(\mu_1) \cap \mathsf{dom}(\mu_2)$. If $\mu_1 \sim \mu_2$, then $\mu_1 \cup \mu_2$ constitutes a mapping that coincides with μ_1 on $\mathsf{dom}(\mu_1)$ and with μ_2 on $\mathsf{dom}(\mu_2)$. Given two sets of mappings Ω_1 and Ω_2, we define their *join*, *union* and *difference* as follows:

$$\begin{array}{rcl} \Omega_1 \bowtie \Omega_2 & = & \{\mu_1 \cup \mu_2 \mid \mu_1 \in \Omega_1, \mu_2 \in \Omega_2, \text{ and } \mu_1 \sim \mu_2\}, \\ \Omega_1 \cup \Omega_2 & = & \{\mu \mid \mu \in \Omega_1 \text{ or } \mu \in \Omega_2\}, \\ \Omega_1 \setminus \Omega_2 & = & \{\mu_1 \mid \mu_1 \in \Omega_1, \mu_1 \not\sim \mu_2 \text{ for all } \mu_2 \in \Omega_2\}. \end{array}$$

Based on these, the *left outer join* operation is defined as $\Omega_1 \mathbin{⟕} \Omega_2 = (\Omega_1 \bowtie \Omega_2) \cup (\Omega_1 \setminus \Omega_2)$. Given a graph G, the *evaluation* $[\![P]\!]_G$ of a graph pattern P over G is defined as follows:

1. if B is a basic pattern, then $[\![B]\!]_G = \{\mu : \mathsf{vars}(B) \to \mathbf{I} \mid \mu(B) \subseteq G\}$;
2. $[\![(P_1\ \mathsf{AND}\ P_2)]\!]_G = [\![P_1]\!]_G \bowtie [\![P_2]\!]_G$;
3. $[\![(P_1\ \mathsf{OPT}\ P_2)]\!]_G = [\![P_1]\!]_G \mathbin{⟕} [\![P_2]\!]_G$;
4. $[\![(P_1\ \mathsf{UNION}\ P_2)]\!]_G = [\![P_1]\!]_G \cup [\![P_2]\!]_G$;
5. $[\![(P'\ \mathsf{FILTER}\ R)]\!]_G = \{\mu \mid \mu \in [\![P']\!]_G \text{ and } \mu \models R\}$,
   where μ *satisfies* a filter constraint R, denoted by $\mu \models R$, if one of the following holds:
   - R is $\top$;
   - R is $?x = u$, $?x \in \mathsf{dom}(\mu)$, and $\mu(?x) = u$;
   - R is $?x = ?y$, $\{?x, ?y\} \subseteq \mathsf{dom}(\mu)$, and $\mu(?x) = \mu(?y)$;
   - R is $\mathit{bound}(?x)$ and $?x \in \mathsf{dom}(\mu)$;
   - R is a Boolean combination of filter constraints evaluating to *true* under the usual interpretation of $\neg, \wedge$, and $\vee$.

Let $\mu|_X$ be the *projection* of a mapping μ to variables X, that is, $\mu|_X(?x) = \mu(?x)$ if $?x \in X$ and $\mu|_X(?x)$ is undefined if $?x \notin X$. The *evaluation* $[\![Q]\!]_G$ of a query Q of the form (4) is the set of all mappings $\mu|_X$ such that $\mu \in [\![P]\!]_G$.

Finally, a *solution* to a query (or pattern) Q over G is a mapping μ such that $\mu \in [\![Q]\!]_G$.

3 Weakly Well-Designed Patterns

We begin by recalling the notion of well-designed patterns and then formulate our generalisation. For now, we focus on the AND-OPT-FILTER fragment $\mathcal{P}$, leaving the operators UNION and SELECT for later sections.

Note that a given pattern can occur more than once within a larger pattern. In what follows we will sometimes need to distinguish between a (sub-)pattern P as a possibly repeated building block of another pattern P' and its *occurrences* in P', that is, unique subtrees in the parse tree. Then, the *left (right) argument* of an occurrence i is the subtree rooted in the left (right) child of the root of i in the parse tree, and an occurrence i is *inside* an occurrence j if the root of i is a successor of the root of j.

▶ **Definition 1** (Pérez et al. [31]). A pattern P from $\mathcal{P}$ is *well-designed* (or *wd-pattern*, for short) if for every occurrence i of an OPT-pattern P_1 OPT P_2 in P the variables from $\mathsf{vars}(P_2) \setminus \mathsf{vars}(P_1)$ occur in P only inside (the labels of) i.

We write $\mathcal{P}_{\text{wd}}$ for the fragment of wd-patterns. Such patterns comply with the basic intuition for optional matching in SPARQL: *"do not reject (solutions) because some part of the query pattern does not match"* [20]; indeed, our canonical use case (1) is clearly well-designed. Evaluation of wd-patterns, that is, checking if $\mu \in [\![P]\!]_G$ for a mapping μ, graph G and pattern $P \in \mathcal{P}_{\text{wd}}$, is coNP-complete, as opposed to PSPACE-complete for $\mathcal{P}$ [31, 36]. The high complexity of unrestricted patterns is partially due to the fact that unrestricted combinations of OPT and FILTER allow to express nesting of the difference operator MINUS with semantics $[\![P_1 \text{ MINUS } P_2]\!]_G = [\![P_1]\!]_G \setminus [\![P_2]\!]_G$ (for non-empty P_1 and P_2):

$$P_1 \text{ MINUS } P_2 \equiv (P_1 \text{ OPT } (P_2 \text{ AND } (?x, ?y, ?z))) \text{ FILTER } \neg \mathit{bound}(?x). \tag{5}$$

This property is well-known [5, 31], and has been usually considered the main source of non-well-designed patterns in practice. We challenge this claim by answering differently the question on the prevalent structure of real-life queries beyond the well-designed fragment. This question is not just of theoretical interest: as previous studies [32] show (and our analysis confirms), about half of queries with OPT asked over DBpedia are not well-designed.

Next we discuss two sources of non-well-designedness in patterns as revealed by the example queries (2) and (3) in the introduction – one based on OPT and another on FILTER.

Source 1. There are two substantially different ways of nesting the OPT operator in patterns:

$$P_1 \text{ OPT } (P_2 \text{ OPT } P_3), \qquad (\texttt{Opt-R}) \qquad\qquad (P_1 \text{ OPT } P_2) \text{ OPT } P_3. \qquad (\texttt{Opt-L})$$

Non-well-designed nesting of type (`Opt-R`) is responsible for the PSPACE-hardness of query evaluation [31, 36]. Moreover, such nesting is not very intuitive. On the contrary, as we saw in the introduction, non-well-designed nesting of type (`Opt-L`) can be used for prioritising some parts of patterns to others, and is indeed used in real life. As we will see later, nesting of type (`Opt-L`) cannot lead to high complexity of evaluation.

Source 2. Well-designedness can be violated by using "dangerous" variables from the right side of OPT in filter constraints. In particular, patterns of the form $(P_1 \text{ OPT } P_2) \text{ FILTER } R$ with R using a variable from $\mathsf{vars}(P_2) \setminus \mathsf{vars}(P_1)$ are not well-designed, but rather frequent in practice. However, such patterns almost never occur inside the right argument of other OPT-patterns. We will see that if we restrict the usage of such filters to the "top level", we preserve the good computational properties of wd-patterns.

Motivated by these observations, we considerably generalise the notion of wd-patterns to allow for useful queries like (2) and (3) while retaining important properties of such patterns.

We start with two auxiliary notions. Given a pattern P, an occurrence i_1 in P *dominates* another occurrence i_2 if there exists an occurrence j of an OPT-pattern such that i_1 is inside the left argument of j and i_2 is inside the right argument. An occurrence i of a FILTER-pattern P' FILTER R in P is *top-level* if there is no occurrence j of an OPT-pattern such that i is inside the right argument of j.

▶ **Definition 2.** A pattern $P \in \mathcal{P}$ is *weakly well-designed* (*wwd-pattern*) if, for each occurrence i of an OPT-subpattern P_1 OPT P_2, the variables in $\mathsf{vars}(P_2) \setminus \mathsf{vars}(P_1)$ appear outside i only in

- subpatterns whose occurrences are dominated by i, and
- constraints of top-level occurrences of FILTER-patterns.

We write $\mathcal{P}_{\mathrm{wwd}}$ for the fragment of wwd-patterns. They extend wd-patterns by allowing variables from the right argument of an OPT-subpattern that are not "guarded" by the left argument to appear in certain positions outside of the subpattern. Note that the patterns of queries (4) and (3) are wwd-patterns. Also, patterns which allow only for OPT nesting of type (`Opt-L`) are always weakly well-designed, same as the pattern in the right hand side of (5), which expresses MINUS. However, patterns that have subpatterns of the latter form in the right argument of OPT are not weakly well-designed. Next we give a few more examples.

▶ **Example 3.** Consider the following patterns:

$$((?x, a, a) \text{ OPT } ((?x, b, ?y) \text{ OPT } (?y, c, ?z))) \text{ OPT } (?x, d, ?z), \tag{6}$$

$$((?x, a, a) \text{ OPT } (?x, d, ?z)) \text{ OPT } ((?x, b, ?y) \text{ OPT } (?y, c, ?z)), \tag{7}$$

$$(((?u, f, ?v) \text{ OPT } (?u, g, ?w)) \text{ FILTER } ?v \neq ?w) \text{ OPT } (?u, h, ?s), \tag{8}$$

$$(?u, h, ?s) \text{ OPT } (((?u, f, ?v) \text{ OPT } (?u, g, ?w)) \text{ FILTER } ?v \neq ?w). \tag{9}$$

Pattern (6) is not well-designed because of variable $?z$, but is weakly well-designed since the occurrence of $(?y, c, ?z)$ dominates $(?x, d, ?z)$. However, the similar pattern (7) is not weakly well-designed because the occurrence of the inner OPT-pattern with the second occurrence of $?z$ does not dominate the first. Pattern (8) is weakly well-designed since the FILTER-pattern is top-level (we write $?x \neq ?y$ for $\neg(?x = ?y)$), but pattern (9) is not, because of variable $?w$ in a non-top-level FILTER.

▶ **Proposition 4.** *Checking whether a pattern P belongs to the fragment $\mathcal{P}_{\mathrm{wwd}}$ can be done in time $O(|P|^2)$, where $|P|$ is the length of the string representation of P.*

4 OPT-FILTER-Normal Form and Constraint Pattern Trees

One of the key properties of wd-patterns is that they can always be converted to a so-called OPT-normal form, in which all AND- and FILTER-subpatterns are OPT-free [31]. Also, FILTER-free patterns in OPT-normal form can be naturally represented as trees [27, 33], which gives a good intuition for the evaluation and optimisation of such patterns. In this section, we show that these notions can be generalised to wwd-patterns.

▶ **Definition 5.** A pattern $P \in \mathcal{P}$ is in OPT-FILTER-*normal form* (or OF-*normal form* for short) if it adheres to the grammar

$$P ::= F \mid (P \text{ FILTER } R) \mid (P \text{ OPT } S), \qquad S ::= F \mid (S \text{ OPT } S), \qquad F ::= (B \text{ FILTER } R),$$

where B ranges over basic patterns and R over filter constraints.

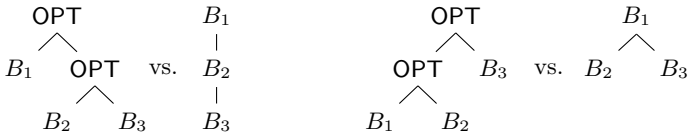

Figure 2 Parse trees vs. constraint pattern trees for patterns **(a)** B_1 OPT $(B_2$ OPT $B_3)$ and **(b)** $(B_1$ OPT $B_2)$ OPT B_3, with B_1, B_2, and B_3 basic patterns.

In other words, the parse tree of a pattern in OF-normal form can be stratified as follows:

1. (occurrences of) basic patterns as the bottom layer,
2. a FILTER on top of each basic pattern as the middle layer,
3. a combination of OPT and FILTER as the top layer;

moreover, each occurrence of a FILTER-pattern in the top layer is top-level. Note that our normal form is AND-free: all conjunctions are expressed via basic patterns.

▶ **Example 6.** None of the four patterns in Example 3 are in OF-normal form. However, the first three of them can be easily normalised by replacing each triple t with $t^\top$, where $P^\top$ is an abbreviation of P FILTER $\top$ for a pattern P. Also, compare the pattern

$$(((?x, a, a)^\top \text{ OPT } (?x, b, ?y)^\top) \text{ OPT } ((?x, b, ?z)^\top \text{ OPT } (?z, c, ?u)^\top)) \text{ FILTER } ?u \neq ?x, \quad (10)$$

which is in OF-normal form, with the very similar pattern

$$(((?x, a, a)^\top \text{ OPT } (?x, b, ?u)^\top) \text{ OPT } ((?x, b, ?z)^\top \text{ OPT } (?z, c, ?u)^\top) \text{ FILTER } ?u \neq ?z),$$

which is not, because the outer FILTER is in the right argument of the outermost OPT.

As shown by Letelier et al. [27], FILTER-free patterns in OPT-normal form can be represented by means of so-called pattern trees. We next show that this representation can be naturally extended to patterns in OF-normal form.

Let P be a pattern in OF-normal form. The *constraint pattern tree (CPT)* $\mathcal{T}(P)$ of P is the directed ordered labelled rooted tree recursively constructed as follows (in this definition we abuse notation and confuse patterns and their occurrences; strictly speaking, we create a fresh sub-tree for each occurrence, so the resulting object is indeed always a tree):

1. if B is a basic pattern then $\mathcal{T}(B \text{ FILTER } R)$ is a single node v labelled by the pair (B, R);
2. if P' is not a basic pattern then $\mathcal{T}(P' \text{ FILTER } R)$ is obtained by adding a *special* node labelled by R as the last child of the root of $\mathcal{T}(P')$;
3. $\mathcal{T}(P_1 \text{ OPT } P_2)$ is the tree obtained from $\mathcal{T}(P_1)$ and $\mathcal{T}(P_2)$ by adding the root of $\mathcal{T}(P_2)$ as the last child of the root of $\mathcal{T}(P_1)$.

By definition, there is a one-to-one correspondence between patterns in OF-normal form and CPTs. Hence, such trees can be seen as a convenient representation of patterns in OF-normal form.

Unlike parse trees, which represent the syntactic shape of patterns, CPTs show the semantic structure of OPT and FILTER nesting. Figure 2 shows how OPT nestings of types (`Opt-R`) and (`Opt-L`) are represented in both formats. Note that CPTs treat different FILTER-subpatterns differently: if the filter is over a basic pattern, the constraint of the FILTER is paired with this pattern; however, if the filter is over an OPT-subpattern, then the constraint is represented by a separate special node. Moreover, since in the second case

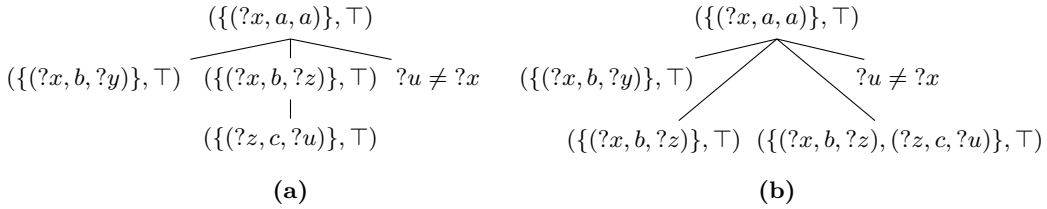

Figure 3 Constraint pattern trees of **(a)** $(((?x, a, a)^{\top}\ \mathsf{OPT}\ (?x, b, ?y)^{\top})\ \mathsf{OPT}\ ((?x, b, ?z)^{\top}\ \mathsf{OPT}\ (?z, c, ?u)^{\top}))\ \mathsf{FILTER}\ ?u \neq ?x$ (i.e., pattern (10)) and **(b)** equivalent pattern in "flat" form (13).

the FILTER-pattern must be top-level, special nodes can only occur in CPTs as children of the root. For instance, the CPT of the example pattern (10) is given in Figure 3(a).

Each wwd-pattern can be converted to OF-normal form and hence can be represented by a CPT. To prove this statement we make use of a number of equivalences. Formally, a pattern P_1 is *equivalent* to a pattern P_2 (written $P_1 \equiv P_2$) if $[\![P_1]\!]_G = [\![P_2]\!]_G$ holds for any graph G. There are several equivalences, such as associativity and commutativity of AND, as well as filter decompositions, such as $P\ \mathsf{FILTER}\ (R_1 \wedge R_2) \equiv (P\ \mathsf{FILTER}\ R_1)\ \mathsf{FILTER}\ R_2$, which hold for all patterns (see [36] for an extensive list). Moreover, the key equivalences used in [31] for normalising wd-patterns can easily be adapted to serve our needs.

▶ **Proposition 7.** *Let P_1, P_2, P_3 be patterns and R a filter constraint such that $\mathsf{vars}(P_2) \cap \mathsf{vars}(P_3) \subseteq \mathsf{vars}(P_1)$ and $\mathsf{vars}(P_2) \cap \mathsf{vars}(R) \subseteq \mathsf{vars}(P_1)$. Then the following equivalences hold:*

$$(P_1\ \mathsf{OPT}\ P_2)\ \mathsf{AND}\ P_3 \equiv (P_1\ \mathsf{AND}\ P_3)\ \mathsf{OPT}\ P_2,$$
$$(P_1\ \mathsf{OPT}\ P_2)\ \mathsf{FILTER}\ R \equiv (P_1\ \mathsf{FILTER}\ R)\ \mathsf{OPT}\ P_2.$$

Since all the equivalences preserve weak well-designedness, we obtain the desired result.

▶ **Proposition 8.** *Each wwd-pattern P is equivalent to a wwd-pattern in OF-normal form of size $O(|P|)$.*

Relying on this proposition, in the rest of the paper we silently assume that all wwd-patterns are in OF-normal form and hence can be represented by CPTs.

We next transfer the notion of weak well-designedness to CPTs. Let $\prec$ be the strict topological sorting of the nodes in $\mathcal{T}(P)$, computed by a depth first search traversal visiting the children of a node according to their ordering (i.e., $v \prec u$ holds if v is visited before u).

▶ **Proposition 9.** *A pattern P in OF-normal form is weakly well-designed if and only if, for each edge (v, u) in its CPT $\mathcal{T}(P)$, every variable $?x \in \mathsf{vars}(u) \setminus \mathsf{vars}(v)$ occurs only in nodes w such that $v \prec w$. The pattern is well-designed if and only if for every variable $?x$ in P the set of all nodes v in $\mathcal{T}(P)$ with $?x \in \mathsf{vars}(v)$ is connected.*

Note that if a pattern is FILTER-free, its OF-normal form coincides with the OPT-normal form in [31] (modulo tautological filters), and its CPT is the pattern tree from [27, 33]. In fact, the second part of Proposition 9 generalises an observation from [27] to the case with filters. An important difference to pattern trees is that in our case the order of children of a node is semantically relevant since wwd-patterns do not satisfy the equivalence

$$(P_1\ \mathsf{OPT}\ P_2)\ \mathsf{OPT}\ P_3 \equiv (P_1\ \mathsf{OPT}\ P_3)\ \mathsf{OPT}\ P_2. \tag{11}$$

This equivalence, established in [30], holds whenever $(\mathsf{vars}(P_2) \cap \mathsf{vars}(P_3)) \subseteq \mathsf{vars}(P_1)$, which is always the case for wd-patterns but not for wwd-patterns, as can be seen on query (2).

We conclude this section with a property that is unique to wwd-patterns: each wwd-pattern is equivalent to a pattern whose corresponding CPT has depth one.

▶ **Definition 10.** A pattern in $\mathcal{P}$ is in *depth-one normal form* if it has the structure

$$(\cdots((B\ \mathsf{op}_1\ S_1)\ \mathsf{op}_2\ S_2)\cdots)\ \mathsf{op}_n\ S_n, \tag{12}$$

where B is a basic pattern and each $\mathsf{op}_i\ S_i$, $1 \leq i \leq n$, is either $\mathsf{OPT}\,(B_i\ \mathsf{FILTER}\ R_i)$ with B_i a basic pattern and R_i a filter constraint, or just $\mathsf{FILTER}\,R_i$.

To show that each wwd-pattern can be brought to this form we use another equivalence.

▶ **Proposition 11.** *For patterns P_1, P_2, P_3 with $\mathsf{vars}(P_1) \cap \mathsf{vars}(P_3) \subseteq \mathsf{vars}(P_2)$ it holds that*

$$P_1\ \mathsf{OPT}\ (P_2\ \mathsf{OPT}\ P_3) \equiv (P_1\ \mathsf{OPT}\ P_2)\ \mathsf{OPT}\ (P_2\ \mathsf{AND}\ P_3). \tag{13}$$

Applied from left to right, equivalence (13) preserves weak well-designedness (but not well-designedness). Each such application transforms a weakly well-designed OPT nesting of type (`Opt-R`) to a nesting of type (`Opt-L`), decreasing the depth of the CPT.

▶ **Corollary 12.** *Every wwd-pattern is equivalent to a wwd-pattern in depth-one normal form.*

For instance, pattern (10) is equivalent to the pattern

$$((((?x, a, a)^{\top}\mathsf{OPT}(?x, b, ?y)^{\top})\mathsf{OPT}(?x, b, ?z)^{\top})\mathsf{OPT}\{(?x, b, ?z), (?z, c, ?u)\}^{\top})\mathsf{FILTER}\ ?u \neq ?x,$$

represented by the CPT in Figure 3(b). Such "flat" patterns are attractive in practice because of their regular structure. However, "flattening" a pattern can incur an exponential blowup in size. Hence, in the rest of the paper we consider arbitrary wwd-patterns in OF-normal form rather than restricting our attention to depth-one-normal patterns.

5 Evaluation of wwd-Patterns

In this section, we look at the query answering problem for wwd-patterns and their extensions with union and projection. We show that in all three cases, complexity remains the same as for wd-patterns. To obtain these results, we develop several new techniques.

Formally, we look at the following decision problem for a given SPARQL fragment $\mathcal{L}$.

Eval($\mathcal{L}$)	**Input:**	Graph G, query $Q \in \mathcal{L}$, and mapping μ
	Question:	Does μ belong to $[\![Q]\!]_G$?

It is known that Eval($\mathcal{U}$) for general patterns $\mathcal{U}$ is PSPACE-complete [31], and the result easily propagates to queries with projection (i.e., $\mathcal{S}$) [27]. For wd-patterns, the evaluation problem is coNP-complete, and can be solved by exploiting the following idea [27].

Suppose we are given a wd-pattern P in OPT-normal form (for simplicity, suppose P is FILTER-free), a graph G, and a mapping μ. First, we look for a subtree of $\mathcal{T}(P)$ that includes the root of $\mathcal{T}(P)$, contains precisely the variables in $\mathsf{dom}(\mu)$, and "matches" G under μ (i.e., images of all its triples under μ are contained in G). This is doable in polynomial time. If such a subtree does not exist, then μ cannot be a solution. Otherwise, the subtree witnesses that μ is a part of a solution to P. Finally, to verify that μ is a complete solution, we need to check that the subtree is maximal, that is, cannot be extended to any more nodes in $\mathcal{T}(P)$ with a match in G. There are linearly many such nodes to check, and each check can be performed in coNP. So, the overall algorithm runs in coNP.

Inspired by this idea, we next show that the low evaluation complexity of wd-patterns transfers to wwd-patterns by developing a coNP algorithm for $\textsc{Eval}(\mathcal{P}_{\text{wwd}})$.

Let P be a wwd-pattern in OF-normal form. An *r-subtree* of $\mathcal{T}(P)$ is a subtree containing the root of $\mathcal{T}(P)$ and all its special children. Every r-subtree is also a CPT representing a wwd-pattern that can be obtained from P by dropping the right arguments of some OPT-subpatterns (i.e., a pattern P' with $P' \trianglelefteq P$ in the notation of [31]). A *child* of an r-subtree $\mathcal{T}(P')$ of $\mathcal{T}(P)$ is a node in $\mathcal{T}(P)$ that is not contained in $\mathcal{T}(P')$ but whose parent is.

▶ **Definition 13.** A mapping μ is a *potential partial solution* (or *pp-solution* for short) to a wwd-pattern P over a graph G if there is an r-subtree $\mathcal{T}(P')$ of $\mathcal{T}(P)$ such that $\mathsf{dom}(\mu) = \mathsf{vars}(P')$, $\mu(\mathsf{triples}(P')) \subseteq G$, and $\mu \models R$ for the constraint R of any ordinary node in $\mathcal{T}(P')$.

A pp-solution μ to P over G can be witnessed by several r-subtrees. However, the union of such r-subtrees is also a witness. Hence, there exists a unique maximal witnessing r-subtree, denoted $\mathcal{T}(P_\mu)$, with P_μ being the corresponding wwd-pattern.

Potential partial solutions generalise "partial solutions" as defined in [31] for wd-patterns. There, every "partial solution" is either a solution or can be extended to one. This is not the case for wwd-patterns. While every solution is clearly a pp-solution, not every pp-solution can be extended to a real one. Real solutions may not just extend pp-solutions by assigning previously undefined variables but can also override variable bindings established in some node v of $\mathcal{T}(P_\mu)$ by extending $\mathcal{T}(P_\mu)$ to a child that precedes v according to the order $\prec$.

An additional complication is the presence of non-well-designed top-level filters. Note that pp-solutions are only required to satisfy the constraints of ordinary nodes in the corresponding CPT, thus ignoring top-level filters. Indeed, requiring pp-solutions to satisfy constraints of top-level filters would be too strong since real solutions do not generally satisfy this property, as demonstrated by the following example.

▶ **Example 14.** Consider the graph $G = \{(1, a, 1), (3, a, 3)\}$ and wwd-pattern

$$P = (((?x, a, 1)\ \mathsf{OPT}\ (?y, a, 2))\ \mathsf{FILTER}\ \neg \mathit{bound}(?y))\ \mathsf{OPT}\ (?y, a, 3).$$

The mapping $\mu = \{?x \mapsto 1, ?y \mapsto 3\}$ is a solution to P over G, but $\mu \not\models \neg \mathit{bound}(?y)$.

We now present a characterisation of solutions for wwd-patterns in terms of pp-solutions that (a)takes into account that not every pp-solution can be extended to a real solution and (b) ensures correct treatment of non-well-designed top-level filters. For this we need some more notation. Given a wwd-pattern P, a node v in $\mathcal{T}(P)$, a graph G, and a pp-solution μ to P over G, let $\mu|_v$ be the projection $\mu|_X$ of μ to the set X of all variables appearing in nodes u of $\mathcal{T}(P_\mu)$ such that $u \prec v$. A mapping μ_1 is *subsumed* by a mapping μ_2 (written $\mu_1 \sqsubseteq \mu_2$) if $\mu_1 \sim \mu_2$ and $\mathsf{dom}(\mu_1) \subseteq \mathsf{dom}(\mu_2)$ (this notion is from [31, 8]).

▶ **Lemma 15.** *A mapping μ is a solution to a wwd-pattern P over a graph G if and only if*

1. *μ is a pp-solution to P over G;*
2. *for any child v of $\mathcal{T}(P_\mu)$ labelled with (B, R) there is no μ' such that $\mu|_v \sqsubseteq \mu'$, $\mu' \models R$, and $\mu'(B) \subseteq G$;*
3. *$\mu|_s \models R$ for any special node s in $\mathcal{T}(P)$ labelled with R.*

Intuitively, a pp-solution μ needs to satisfy two conditions to be a real solution to a wwd-pattern P. First, $\mu|_v$ (as opposed to μ for wd-patterns) must be non-extendable to v for any child v of $\mathcal{T}(P_\mu)$. Indeed, if such an extension exists, then it is either possible to provide

bindings for some variables that are undefined in μ, or some variables from $\mathsf{dom}(\mu)$ can be assigned different values of higher "priority" than the corresponding values in μ. Second, every top-level filter R labelling a node s needs to be satisfied by $\mu|_s$, which is precisely the part of μ bound by the subpattern of P that is paired with R in the FILTER-pattern.

Checking whether a mapping μ satisfies this characterisation is feasible in coNP: testing whether μ is a pp-solution takes polynomial time, same as computing the maximal witnessing tree $\mathcal{T}(P_\mu)$; to check that (the relevant part of) $\mathcal{T}(P_\mu)$ is not extendable to any of its children we need to consider linearly many children, and each check is in coNP; finally, the checks for top-level filters are again polynomial. Hence, we obtain the following theorem, where the hardness part follows from the coNP-hardness for wd-patterns [31].

▶ **Theorem 16.** $\textsc{Eval}(\mathcal{P}_{\text{wwd}})$ *is* coNP*-complete.*

Pérez et al. [31] extended wd-patterns to UNION by considering *unions of wd-patterns*, that is, patterns of the form P_1 UNION ... UNION P_n with all $P_i \in \mathcal{P}_{\text{wd}}$. We denote the resulting fragment by $\mathcal{U}_{\text{wd}}$. This syntactic restriction on the use of UNION in $\mathcal{U}_{\text{wd}}$ is motivated by the fact that any pattern in $\mathcal{U}$ can be equivalently expressed as a union of UNION-free patterns [31]. We denote the fragment of all queries over patterns in $\mathcal{U}_{\text{wd}}$ as $\mathcal{S}_{\text{wd}}$. Similarly, we write $\mathcal{U}_{\text{wwd}}$ for unions of wwd-patterns and $\mathcal{S}_{\text{wwd}}$ for queries over unions of wwd-patterns.

Analogously to the well-designed case, Theorem 16 extends to fragments $\mathcal{U}_{\text{wwd}}$ and $\mathcal{S}_{\text{wwd}}$.

▶ **Corollary 17.** $\textsc{Eval}(\mathcal{U}_{\text{wwd}})$ *is* coNP*-complete and* $\textsc{Eval}(\mathcal{S}_{\text{wwd}})$ *is* Σ_2^p*-complete.*

The coNP-algorithm for $\mathcal{U}_{\text{wwd}}$ is obtained simply by applying the algorithm for $\mathcal{P}_{\text{wwd}}$ to each pattern in the union. Hardness for $\mathcal{S}_{\text{wwd}}$ follows from the hardness of the well-designed case [27], while for membership we just guess the values of the existential variables and then call a coNP-oracle for $\mathcal{U}_{\text{wwd}}$ on the resulting mapping and the normalised body of the query.

Hence, the complexity of evaluation for wwd-patterns is the same as for wd-patterns. We next show that wwd-patterns are, in a certain sense, a maximal extension of wd-patterns that preserves coNP evaluation complexity (under the usual complexity-theoretic assumptions).

There are two possible minimal relaxations of weak well-designedness that allow for basic patterns and filter constraints of arbitrary shape. We show that both lead to Π_2^p-hardness.

The first such relaxation is to allow for at least some non-well-designed OPT-nesting of type (`Opt-R`). However, even a minimal extension of this sort increases complexity. To see this, consider the fragment $\mathcal{P}_{\text{opt-r}}$ of patterns of the form B_1 OPT $(B_2$ OPT $B_3)$, where B_1, B_2 and B_3 are basic patterns. Intuitively, $\mathcal{P}_{\text{opt-r}}$ allows for the most simple form of non-well-designed nesting of type (`Opt-R`).

The other syntactic relaxation is to allow for some non-well-designed non-top-level filters. However, while requiring special nodes to be children of the root may look somewhat ad-hoc, it cannot be substantially relaxed. Consider the fragment $\mathcal{P}_{\text{filter-2}}$ of patterns of the form B_1 OPT $((B_2$ OPT $B_3)$ FILTER $R)$, where B_1, B_2 and B_3 are basic patterns such that $\mathsf{vars}(B_3) \cap \mathsf{vars}(B_1) \subseteq \mathsf{vars}(B_2)$, and R is a filter constraint. Intuitively, $\mathcal{P}_{\text{filter-2}}$ allows for the simplest form of "second-level" filters.

▶ **Proposition 18.** *The problems* $\textsc{Eval}(\mathcal{P}_{\text{opt-r}})$ *and* $\textsc{Eval}(\mathcal{P}_{\text{filter-2}})$ *are* Π_2^p*-complete.*

Proposition 18 implies that $\mathcal{P}_{\text{wwd}}$ is a maximal fragment of $\mathcal{P}$ that does not impose structural restrictions on basic patterns or filter constraints and has a coNP evaluation algorithm (assuming coNP $\neq \Pi_2^p$). Hence, going beyond wwd-patterns while preserving good computational properties requires more refined restrictions, as done, for example, in [27, Section 4].

6 Expressivity of wwd-Patterns and their Extensions

In this section, we analyse the expressive power of our fragments. Formally, a language $\mathcal{L}_1$ has the *same expressive power* as a language $\mathcal{L}_2$ (written $\mathcal{L}_1 \sim \mathcal{L}_2$) if for every query Q_2 in $\mathcal{L}_2$ there is a query Q_1 in $\mathcal{L}_1$ such that $Q_2 \equiv Q_1$ and vice versa; $\mathcal{L}_1$ is *strictly more expressive* than $\mathcal{L}_2$ (written $\mathcal{L}_2 < \mathcal{L}_1$) if the property holds in the forward but not in the backward direction. We begin by establishing $\mathcal{P}_{\mathrm{wd}} < \mathcal{P}_{\mathrm{wwd}} < \mathcal{P}$. Then we proceed to unions, showing that $\mathcal{U}_{\mathrm{wd}} < \mathcal{U}_{\mathrm{wwd}} < \mathcal{U}$. Finally, we establish $\mathcal{S}_{\mathrm{wwd}} \sim \mathcal{S}$, i.e., wwd-patterns with union and projection have the full expressive power of SPARQL (whereas it is known that $\mathcal{S}_{\mathrm{wd}} < \mathcal{S}$ [31], which then implies $\mathcal{S}_{\mathrm{wd}} < \mathcal{S}_{\mathrm{wwd}}$).

Following [31, 8], a set of mappings Ω_1 is *subsumed* by a set of mappings Ω_2 (written $\Omega_1 \sqsubseteq \Omega_2$) if for every $\mu_1 \in \Omega_1$ there exists a mapping $\mu_2 \in \Omega_2$ such that $\mu_1 \sqsubseteq \mu_2$. A query Q is *weakly monotone* if $[\![Q]\!]_{G_1} \sqsubseteq [\![Q]\!]_{G_2}$ for any two graphs G_1 and G_2 with $G_1 \subseteq G_2$, and a fragment $\mathcal{L}$ is *weakly monotone* if it contains only weakly monotone queries. Arenas and Pérez [8] showed that, unlike $\mathcal{P}$, the fragment $\mathcal{P}_{\mathrm{wd}}$ is weakly monotone, and hence $\mathcal{P}_{\mathrm{wd}} < \mathcal{P}$.

▶ **Example 19** (Pérez et al. [31])**.** Consider the non-well-designed pattern

$$P = (?x, a, 1)\ \mathsf{OPT}\ ((?y, a, 2)\ \mathsf{OPT}\ (?x, a, 3))$$

as well as graphs $G_1 = \{(1, a, 1), (2, a, 2)\}$ and $G_2 = G_1 \cup \{(3, a, 3)\}$. Then $\mu_1 = \{?x \mapsto 1, ?y \mapsto 2\}$ is the only mapping in $[\![P]\!]_{G_1}$ while $\mu_2 = \{?x \mapsto 1\}$ is the only mapping in $[\![P]\!]_{G_2}$. Hence $[\![P]\!]_{G_1} \not\sqsubseteq [\![P]\!]_{G_2}$, meaning P is not weakly monotone.

Analogously, we show that $\mathcal{P}_{\mathrm{wd}} < \mathcal{P}_{\mathrm{wwd}}$ by observing that $\mathcal{P}_{\mathrm{wwd}}$ is not weakly monotone. Indeed, the pattern in example query (2) violates weak monotonicity: if a graph G contains the triple (`P1`, `v_card:name`, *Anastasia*) but no triple of the form (`P1`, `foaf:name`, u) for any IRI u, then extending G with (`P1`, `foaf:name`, *Ana*), that is, adding more reliable information about the name of `P1`, does not extend the original solution $\{?i \mapsto$ `P1`$, ?n \mapsto$ *Anastasia*$\}$ but modifies it by overriding the value of $?n$. Since $\mathcal{P}_{\mathrm{wd}} \subseteq \mathcal{P}_{\mathrm{wwd}}$, we conclude that $\mathcal{P}_{\mathrm{wd}} < \mathcal{P}_{\mathrm{wwd}}$.

To distinguish $\mathcal{P}_{\mathrm{wwd}}$ from $\mathcal{P}$ we need a different property.

▶ **Definition 20.** A query Q is *non-reducing* if for any two graphs G_1, G_2 such that $G_1 \subseteq G_2$ and any mapping $\mu_1 \in [\![Q]\!]_{G_1}$ there is no $\mu_2 \in [\![Q]\!]_{G_2}$ such that $\mu_2 \sqsubset \mu_1$ (i.e., $\mu_2 \sqsubseteq \mu_1$ and $\mu_2 \neq \mu_1$). A fragment $\mathcal{L}$ is *non-reducing* if it contains only non-reducing queries.

Intuitively, for a non-reducing query extending a graph cannot result in a previously bound answer variable becoming unbound. Weakly monotone queries are non-reducing but not vice versa. Moreover, it is easily seen that wwd-patterns are non-reducing.

This property is not generally satisfied by patterns that are not weakly well-designed. For instance, consider again pattern P, graphs G_1, G_2, and mappings μ_1, μ_2 from Example 19. Pattern P is not non-reducing since $\mu_1 \in [\![P]\!]_{G_1}$ and $\mu_2 \in [\![P]\!]_{G_2}$ but $\mu_2 \sqsubset \mu_1$.

▶ **Theorem 21.** *It holds that* $\mathcal{P}_{\mathrm{wd}} < \mathcal{P}_{\mathrm{wwd}} < \mathcal{P}$.

We next compare $\mathcal{U}_{\mathrm{wwd}}$ to $\mathcal{U}_{\mathrm{wd}}$ and $\mathcal{U}$, and $\mathcal{S}_{\mathrm{wwd}}$ to $\mathcal{S}_{\mathrm{wd}}$ and $\mathcal{S}$ (note that neither UNION nor projection via SELECT can be expressed by means of the other operators [37], so adding either construct makes each fragment strictly more expressive). It is easily seen that $\mathcal{U}_{\mathrm{wd}}$ and $\mathcal{S}_{\mathrm{wd}}$ inherit weak monotonicity from $\mathcal{P}_{\mathrm{wd}}$ [31, 27], and hence $\mathcal{U}_{\mathrm{wd}} < \mathcal{U}_{\mathrm{wwd}}$ and $\mathcal{S}_{\mathrm{wd}} < \mathcal{S}_{\mathrm{wwd}}$. Non-reducibility, however, propagates neither to unions nor to projection.

▶ **Example 22.** Consider the following $\mathcal{U}_{\text{wd}}$-pattern with G_1, G_2 and μ_1, μ_2 from Example 19:

$$P = ((?x, a, 1)\ \mathsf{OPT}\ (?y, a, 2))\ \mathsf{UNION}\ (?x, a, 1).$$

We have $\mu_1 \in [\![P]\!]_{G_1}$ and $\mu_2 \in [\![P]\!]_{G_2}$ but $\mu_2 \sqsubset \mu_1$, which is due to the fact that μ_2 is already contained in $[\![P]\!]_{G_1}$ along with μ_1. This is only possible in the presence of UNION since all mappings in the evaluation of a UNION-free pattern are mutually non-subsuming [31].

Thus, to account for UNION, we introduce the following, more delicate property.

▶ **Definition 23.** A query Q is *extension-witnessing* (*e-witnessing*) if for any two graphs $G_1 \subseteq G_2$ and mapping $\mu \in [\![Q]\!]_{G_2}$ such that $\mu \notin [\![Q]\!]_{G_1}$ there is a triple t in Q such that $\mathsf{vars}(t) \subseteq \mathsf{dom}(\mu)$ and $\mu(t) \in G_2 \setminus G_1$. A fragment is *e-witnessing* if so are all of its queries.

Informally, a query Q is e-witnessing if whenever an extension of a graph leads to a new answer, this answer is justified by a triple pattern in Q which maps to the extension. Unions of wwd-patterns can be shown e-witnessing. On the other hand, $\mathcal{U}$ is not e-witnessing, as can be seen on the pattern and graphs in Example 19. Hence, we obtain the following theorem.

▶ **Theorem 24.** *It holds that* $\mathcal{U}_{\text{wd}} < \mathcal{U}_{\text{wwd}} < \mathcal{U}$.

In contrast, queries over unions of wwd-patterns are as expressive as full SPARQL.

▶ **Theorem 25.** *It holds that* $\mathcal{S}_{\text{wwd}} \sim \mathcal{S}$.

As a consequence, every SPARQL query can be rewritten to a query over a union of "flat" patterns in depth-one normal form (Definition 10), albeit at the expense of a worst-case exponential blow-up in size.

7 Static Analysis of wwd-Patterns

In this section, we look at the general static analysis problems of query equivalence, containment, and subsumption. Formally, equivalence for a language $\mathcal{L}$ is defined as follows.

EQUIVALENCE($\mathcal{L}$)	**Input:**	Queries Q and Q' from $\mathcal{L}$
	Question:	Is $Q \equiv Q'$?

This problem is commonly generalised to CONTAINMENT($\mathcal{L}$), in which one checks whether Q is *contained* in Q', that is, whether $[\![Q]\!]_G \subseteq [\![Q']\!]_G$ holds for every graph G. We have $Q \equiv Q'$ if and only if Q and Q' contain each other. Furthermore, Letelier et al. [27] proposed the problem SUBSUMPTION($\mathcal{L}$), where one checks whether Q is *subsumed* by Q', that is, whether $[\![Q]\!]_G \sqsubseteq [\![Q']\!]_G$ holds for every G.

These problems have been studied for FILTER-free wd-patterns in [27, 33], establishing NP-completeness of equivalence and containment, and Π_2^p-completeness of subsumption. Moreover, all three problems are Π_2^p-complete for unions of FILTER-free wd-patterns, and undecidable for fragments with projection. Finally, from the results in [38] it follows that containment and subsumption are undecidable for $\mathcal{U}$. On the other hand, nothing seems to be known so far for well-designed patterns with FILTER.

We next show that equivalence, containment, and subsumption are all Π_2^p-complete for $\mathcal{P}_{\text{wwd}}$ and $\mathcal{U}_{\text{wwd}}$ (whereas $\mathcal{S}_{\text{wwd}}$ is undecidable by the results in [33]). The upper bound for containment follows from a small counterexample property: if $P \not\subseteq P'$ for some P and P' from $\mathcal{U}_{\text{wwd}}$, then there is a witnessing mapping of size $O(|P| + |P'|)$. Given this property, a

Table 1 Structure of query patterns in DBpedia logs.

	DBpedia 3.8			DBpedia 3.9		
	unique patterns	fraction of total	fraction of OPT	unique patterns	fraction of total	fraction of OPT
total	7 014 249	100%		27 854	100%	
patterns with OPT	742 002	10.58%	100%	1 639	5.83%	100%
unions of wd-patterns	238 995	3.41%	32.32%	972	3.49%	59.31%
unions of wwd-patterns	736 051	10.49%	99.19%	1 620	5.82%	98.84%

Π_2^p algorithm for containment is straightforward – we guess a mapping μ and a graph G of linear size, check that $\mu \notin [\![P']\!]_G$, and then call a coNP oracle for checking $\mu \in [\![P]\!]_G$. As a corollary, Equivalence($\mathcal{U}_{\text{wwd}}$) is also in Π_2^p. The argument for subsumption is analogous.

Hardness of subsumption and equivalence is established by a reduction from $\forall\exists$3SAT, while containment is Π_2^p-hard by the results in [33].

▶ **Theorem 26.** *Problems* Equivalence($\mathcal{L}$), Containment($\mathcal{L}$) *and* Subsumption($\mathcal{L}$) *are* Π_2^p*-complete for any* $\mathcal{L} \in \{\mathcal{P}_{\text{wwd}}, \mathcal{U}_{\text{wwd}}\}$.

Hence, for UNION- and FILTER-free patterns the step from well-designed to weakly well-designed OPT incurs a complexity jump for containment and equivalence. However, for the fragments with UNION or projection complexity remains the same in all three cases. As far as we are aware, these are the first decidability results on query equivalence and related problems for SPARQL fragments with OPT and FILTER.

8 Analysis of DBpedia Logs

In this section, we present a preliminary analysis of query logs over DBpedia, which suggests that the step from wd- to wwd-patterns makes a dramatic difference in real life: while only about half of the queries with OPT have well-designed patterns, almost all of these patterns fall into the weakly well-designed fragment.

DBpedia [26] is a project providing access to RDF data extracted from Wikipedia via a SPARQL endpoint. DBpedia query logs are well suited for analysing the structure of real-life SPARQL queries as they contain a large amount of general-purpose knowledge base queries, generated both manually and automatically. DBpedia query logs have been analysed by Picalausa and Vansummeren [32], who reported that, over a period in 2010, about 46.38% of a total of 1344K distinct DBpedia queries used OPT. However, only 47.80% of the queries with OPT had well-designed patterns. Another analysis of DBpedia logs from the USEWOD2011 data set performed by Arias Gallego et al. [9] concluded that 16.61% of about 5166K queries contain OPT; however, detailed structure of queries was not analysed.

We considered query logs over DBpedia 3.8 from USEWOD2013 [10] and DBpedia 3.9 logs from USEWOD2014 [11]. The DBpedia 3.8 set is a random selection of almost 12M queries from 2012 while the DBpedia 3.9 set contains only 253K queries, from 2013 and beginning of 2014. We removed syntactically incorrect queries as well as queries outside of $\mathcal{S}$ (in particular, queries using operators specific to SPARQL 1.1). Also, we rewrote the patterns of the remaining queries to unions of UNION-free patterns as proposed in [31] and eliminated duplicates, which left us with just over 7M queries over DBpedia 3.8 and 28K queries over DBpedia 3.9 (the decrease from 253K to 28K for DBpedia 3.9 is mostly due to duplicate elimination – with duplicates, we still have 197K queries). Finally, we isolated queries involving OPT and counted how many of their patterns were in $\mathcal{U}_{\text{wwd}}$ and in $\mathcal{U}_{\text{wd}}$.

The results are given in Table 1. They confirm that a non-negligible number of DBpedia queries use OPT; the exact fraction, however, varies considerably between the logs. In both cases, however, by far not all queries with OPT are well-designed (only 32% for DBpedia 3.8 and 59% for DBpedia 3.9), which is consistent with the results in [32]. On the other hand, almost all of the patterns with OPT (around 99% in both cases) are weakly well-designed, which we consider as the main practical justification for wwd-patterns.

9 Conclusion and Future Work

In this paper, we introduced a new fragment of SPARQL patterns called weakly well-designed patterns. This fragment extends the widely studied well-designed fragment by allowing variables from the optional side of an OPT-subpattern that are not "guarded" by the mandatory side to occur in certain positions outside of the subpattern. We showed that queries with wwd-patterns enjoy the same low complexity of evaluation as well-designed queries but cover almost all real-life queries. Moreover, our fragment is the maximal coNP fragment that does not impose structural restrictions on basic patterns and filter conditions. We studied the expressive power of the fragment and the complexity of its query optimisation problems.

For future work, we want to extend wwd-patterns to allow for non-top-level occurrences of UNION and projection. Also, we want to take into account features of SPARQL 1.1 [20] such as GRAPH, NOT EXISTS and property paths. Finally, we would like to implement our ideas in a prototype and compare its performance with existing SPARQL engines.

References

1 AllegroGraph. URL: `http://franz.com/agraph/allegrograph/`.
2 Apache Jena. URL: `http://jena.apache.org`.
3 RDF4J. URL: `http://rdf4j.org`.
4 Virtuoso Universal Server. URL: `http://virtuoso.openlinksw.com`.
5 Renzo Angles and Claudio Gutierrez. The expressive power of SPARQL. In *ISWC*, pages 114–129, 2008.
6 Marcelo Arenas, Sebastián Conca, and Jorge Pérez. Counting beyond a yottabyte, or how SPARQL 1.1 property paths will prevent adoption of the standard. In *WWW*, pages 629–638, 2012.
7 Marcelo Arenas, Georg Gottlob, and Andreas Pieris. Expressive languages for querying the semantic web. In *PODS*, pages 14–26, 2014.
8 Marcelo Arenas and Jorge Pérez. Querying semantic web data with SPARQL. In *PODS*, pages 305–316, 2011.
9 Mario Arias Gallego, Javier D. Fernández, Miguel A. Martínez-Prieto, and Pablo de la Fuente. An empirical study of real-world SPARQL queries. In *USEWOD*, 2011. arXiv:1103.5043.
10 Bettina Berendt, Laura Hollink, Markus Luczak-Rösch, Knud Möller, and David Vallet. USEWOD2013: 3rd international workshop on usage analysis and the web of data. In *ESWC*, 2013.
11 Bettina Berendt, Laura Hollink, Markus Luczak-Rösch, Knud Möller, and David Vallet. USEWOD2014: 4th international workshop on usage analysis and the web of data. In *ESWC*, 2014.
12 Stefan Bischof, Markus Krötzsch, Axel Polleres, and Sebastian Rudolph. Schema-agnostic query rewriting in SPARQL 1.1. In *ISWC*, pages 584–600, 2014.

13 Carlos Buil-Aranda, Marcelo Arenas, and Oscar Corcho. Semantics and optimization of the SPARQL 1.1 federation extension. In *ESWC*, pages 1–15. Springer, 2011.

14 Carlos Buil Aranda, Axel Polleres, and Jürgen Umbrich. Strategies for executing federated queries in SPARQL1.1. In *ISWC*, pages 390–405, 2014.

15 Melisachew Wudage Chekol, Jérôme Euzenat, Pierre Genevès, and Nabil Layaïda. SPARQL query containment under RDFS entailment regime. In *IJCAR*, pages 134–148, 2012.

16 Melisachew Wudage Chekol, Jérôme Euzenat, Pierre Genevès, and Nabil Layaïda. SPARQL query containment under SHI axioms. In *AAAI*, pages 10–16, 2012.

17 Richard Cyganiak, David Wood, and Markus Lanthaler. RDF 1.1 concepts and abstract syntax. W3C recommendation, W3C, February 2014. URL: `http://www.w3.org/TR/rdf11-concepts/`.

18 Floris Geerts, Grigoris Karvounarakis, Vassilis Christophides, and Irini Fundulaki. Algebraic structures for capturing the provenance of SPARQL queries. In *ICDT*, pages 153–164, 2013.

19 Harry Halpin and James Cheney. Dynamic provenance for SPARQL updates. In *ISWC*, pages 425–440, 2014.

20 Steve Harris and Andy Seaborne. SPARQL 1.1 query language. W3C recommendation, W3C, March 2013. URL: `http://www.w3.org/TR/sparql11-query/`.

21 Patrick J. Hayes and Peter F. Patel-Schneider. RDF 1.1 semantics. W3C recommendation, W3C, February 2014. URL: `http://www.w3.org/TR/rdf11-mt/`.

22 Roman Kontchakov, Martin Rezk, Mariano Rodriguez-Muro, Guohui Xiao, and Michael Zakharyaschev. Answering SPARQL queries over databases under OWL 2 QL entailment regime. In *ISWC*, pages 552–567, 2014.

23 Egor V. Kostylev and Bernardo Cuenca Grau. On the semantics of SPARQL queries with optional matching under entailment regimes. In *ISWC*, pages 374–389, 2014.

24 Egor V. Kostylev, Juan L. Reutter, Miguel Romero, and Domagoj Vrgoc. SPARQL with property paths. In *ICWC*, pages 3–18, 2015.

25 Egor V. Kostylev, Juan L. Reutter, and Martín Ugarte. CONSTRUCT queries in SPARQL. In *ICDT*, pages 212–229, 2015.

26 Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N. Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick van Kleef, Sören Auer, and Christian Bizer. DBpedia – A large-scale, multilingual knowledge base extracted from Wikipedia. *Semantic Web*, 6(2):167–195, 2015.

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

28 Katja Losemann and Wim Martens. The complexity of evaluating path expressions in SPARQL. In *PODS*, pages 101–112, 2012.

29 Frank Manola, Eric Miller, and Brian McBride. RDF 1.1 primer. W3C working group note, W3C, June 2014. URL: `http://www.w3.org/TR/rdf11-primer/`.

30 Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. Semantics and complexity of SPARQL. In *ISWC*, pages 30–43, 2006.

31 Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. Semantics and complexity of SPARQL. *ACM Transactions on Database Systems*, 34(3), 2009.

32 François Picalausa and Stijn Vansummeren. What are real SPARQL queries like? In *SWIM*, 2011.

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

34 Axel Polleres and Johannes Peter Wallner. On the relation between SPARQL1.1 and answer set programming. *Journal of Applied Non-Classical Logics*, 23(1-2):159–212, 2013.

35 Eric Prud'hommeaux and Andy Seaborne. SPARQL query language for RDF. W3C recommendation, W3C, January 2008. URL: `http://www.w3.org/TR/rdf-sparql-query/`.

36 Michael Schmidt, Michael Meier, and Georg Lausen. Foundations of SPARQL query optimization. In *ICDT*, pages 4–33, 2010.

37 Xiaowang Zhang and Jan Van den Bussche. On the primitivity of operators in SPARQL. *Information Processing Letters*, 114(9):480–485, 2014.

38 Xiaowang Zhang and Jan Van den Bussche. On the satisfiability problem for SPARQL patterns, 2014. arXiv:1406.1404.

39 Xiaowang Zhang and Jan Van den Bussche. On the power of SPARQL in expressing navigational queries. *The Computer Journal*, 58(11):2841–2851, 2015.