# Modeling Interdependent Preferences over Incomplete Knowledge Graph Query Answers

Till Affeldt[1][0000−0001−6440−5654], Stephan Mennicke[2][0000−0002−3293−2940], and Wolf-Tilo Balke[1][0000−0002−5443−1215]

[1] Institute for Information Systems, TU Braunschweig, Braunschweig, Germany
[2] Knowledge-Based Systems Group, TU Dresden, Dresden, Germany

**Abstract.** We study the properties of optimal patterns, our novel extension of Sparql, that allows for a fine-grained control over incompleteness of query answers in knowledge graphs. Optimal patterns encode preferences over the completeness of query answers. In this paper, we add language constructs for expressing dependencies between conflicting preferences. Furthermore, we provide discussions and proofs of fundamental results concerning Sparql with optimal patterns.

**Keywords:** knowledge graphs · structural preferences · Sparql.

## 1 Introduction

One of the major difficulties in working with knowledge graphs is that entities may vastly differ in their structural representation, even if they are of the same kind. Indeed, due to limitations of the underlying sources and knowledge extraction processes it is quite common that in practical instances two entities (represented as nodes or resources) share a type, but may be characterized in totally different ways regarding their properties. The consequences are unexpectedly small or even empty result sets when querying basic graph patterns (conjunctive queries), usually followed by a lot of manual query refinements in the retrieval process. Thus, operators for handling such heterogeneity are mandatory when designing a robust query language for knowledge graphs. Sparql [11] already offers *optional patterns*, which handle incompleteness in RDF graphs by returning complete matches (i. e., mandatory plus optional parts) and incomplete matches. Optional patterns and well-behaving subclasses of them have been heavily discussed over the last decade [12,13,2,6,4]. However, the diversity of result types remains unchanged. Suppose we query for three patterns $A$, $B$, and $C$:

$$A \ \mathsf{OPTIONAL}\ B \ \mathsf{OPTIONAL}\ C.$$

Then we may get results satisfying $A$ alone, results that additionally satisfy $B$ or $C$, and results that satisfy all three patterns. In fact, the number of result schemas can be exponential in the number of OPTIONAL operators in a query. We recently proposed *optimal patterns* [1], which come with a built-in preference semantics [8], so that only maximal results are returned. In the example above, we would not return results satisfying $A$ alone when there are results that satisfy more patterns.

The new OPTIMAL operator for SPARQL enables the expression of structural as well as some value preferences. These preferences can be used to satisfy a user's information need if no perfect results can be found. For instance, a user may ask for a car with sunroof and air conditioner that costs less than 10,000€. If such configurations are not available in the data, cars with fewer features or higher prices are still relevant. Evaluating queries that use our OPTIMAL operator automatically retrieves the best possible answers. Thus, a user can receive useful information instead of having to manually tweak her search settings until a result can be found.

OPTIMAL positions itself between the rather restrictive query conjunction and the loose optional patterns of SPARQL. In our previous work [1] we have shown that the operator can be used in real-world scenarios to achieve fine-grained control over inherent incompleteness of RDF knowledge bases. It can also be used to model multiple preferences simultaneously, both of similar as well as prioritized relevance. Moreover, using optimal patterns results in readable and maintainable query representations. After we introduce some notational conventions in Sect. 2, we briefly recap syntax and semantics of optimal patterns in Sect. 3. In addition to what has been shown in our previous work, we discuss fundamental design decisions and properties of optimal patterns (Sections 3.1 and 3.2). In particular, we discuss how optimal patterns behave in open world knowledge bases like RDF.

Optimal patterns alone lack the ability to describe complex dependencies between preference patterns. As we will argue with concrete query examples in Sect. 4.1, we cannot express what we call *positive* and *negative dependencies*. For instance, the preferred color of a car may only be relevant for a certain type of car but irrelevant for others. Similarly, optimal patterns do not allow for modeling the next-best preference if one preference cannot be satisfied. Therefore, we extend our earlier work introducing structural preferences into SPARQL with a focus on retrieving *reasonably sized* result sets over incomplete data sets. We analyze how multiple preferences can affect each other leading to a characterization of different types of dependencies (Sect. 4). In order to cope with these different types, we then introduce a suitable set of operators to be used in conjunction with OPTIMAL.

Modeling preferences with SPARQL has been the goal of several language extensions [5,10,14,15]. However, all of these works focus on a preference model based on preferred data values, i. e., they allow for expressing certain preferences over the attribute values such as the lowest price for a car or a specific set of brands. But, while these extensions work quite well for value-based ranking and selection operations, users cannot express preferences with respect to the completeness of query results. Indeed, expressing a simple query such as *I am* *looking for cars, preferably including information on prices and/or brands* will result in rather cumbersome expressions. And even if a user's preferences are not completely satisfiable in some given knowledge graph, he/she is still looking for cars and expects that cars are returned whenever possible. Another difference is that value preferences do usually not consider structural preference dependencies. This is because no structural dependencies can exist if the returned results are guaranteed to be structurally complete. In this case also the need for non-structural dependencies is greatly reduced: The same results can often be achieved by defining a priority order for preferences, in particular by evaluating independent preferences first. For a detailed discussion of other related approaches see [1].

## 2 Preliminaries

Since SPARQL is the recommended query language for the *Resource Description Framework* (RDF) [11], we base our notions on that of RDF. Therefore, assume infinite and disjoint universes of IRIs $\mathbf{I}$ and literals $\mathbf{L}$. The core construct of RDF is that of an *RDF triple* $(s, p, o) \in \mathbf{I} \times \mathbf{I} \times \mathbf{IL}$[3] In such a triple, predicates $(p)$ connect resources $(s)$ to other resources or actual data values $(o)$. Sets of RDF triples form *RDF graphs* $\mathbf{G}$.

SPARQL offers expressions for querying information from RDF graphs. We restrict our presentation to pattern matching capabilities, e. g., leaving out path queries. Abstract syntax and semantics of SPARQL follow the standard notation [11,12]. Let $\mathbf{V}$ denote the universe of variables, e. g., $\mathsf{x}, \mathsf{y}, \mathsf{z} \in \mathbf{V}$. A *triple pattern* is a triple $(u, a, v) \in \mathbf{VI} \times \mathbf{VI} \times \mathbf{VIL}$, i. e., variables $\mathsf{x} \in \mathbf{V}$ may occur in every component of triple patterns. Sets of triple patterns $\mathbb{G}$ also relate to graph-like structures, then called *basic graph patterns* (BGPs).

Let $t = (u, a, v)$ be a triple pattern and $\mathbf{G}$ an RDF graph. A partial function $\mu : \mathbf{VIL} \to \mathbf{IL}$ is called a *match of $t$ in* $\mathbf{G}$ iff (a) $\mu(c) = c$ for all $c \in \mathbf{IL}$, (b) $\{u, a, v\} \subseteq dom(\mu)$[4], and (c) $(\mu(u), \mu(a), \mu(v)) \in \mathbf{G}$. By $\emptyset_{\mathbf{IL}}$ we denote the *empty match*, being the partial function that respects (a) but is otherwise undefined. The notion of a match naturally extends to BGPs $\mathbb{G}$ by requiring that $\mu$ is a match of all triple patterns $t \in \mathbb{G}$. We denote the set of all matches of $t$ ($\mathbb{G}$, resp.) in $\mathbf{G}$ by $[\![t]\!]_{\mathbf{G}}$ ($[\![\mathbb{G}]\!]_{\mathbf{G}}$, resp.).

SPARQL further supports complex queries in terms of *unions*, *query conjunctions*, *optional patterns*, and *filter conditions*. The semantics of such queries $\mathcal{Q}$ (w. r. t. RDF graphs $\mathbb{G}$) can be found in [12] and is denoted by $[\![\mathcal{Q}]\!]_{\mathbb{G}}$. The core notion of *compatibility* for query conjunctions and optional patterns is also required for a formal account of $\mathsf{OPTIMAL}$. In general, two partial functions $\mu_1, \mu_2 : \mathbf{VIL} \to \mathbf{IL}$ are *compatible*, denoted $\mu_1 \leftrightarrows \mu_2$, iff for all $x \in dom(\mu_1) \cap dom(\mu_2)$, $\mu_1(x) = \mu_2(x)$, i. e., $\mu_1$ and $\mu_2$ agree on the variables they share. In conjunctions and optional patterns, only compatible matches to subpatterns are joined in the result sets.

[3] We use $\mathbf{IL}$ as shorthand for $\mathbf{I} \cup \mathbf{L}$.

[4] $dom(\mu)$ refers to the set of all elements of $\mathbf{VIL}$ for which $\mu$ is defined.

## 3 SPARQL with OPTIMAL

Throughout this section, we present our extension of Sparql by *optimal patterns*. Additional examples, an encoding by standard Sparql, as well as an initial evaluation may be found in [1]. Here, we give a brief account of the syntax and semantics. Furthermore, we discuss the properties of optimal patterns formally (cf. Sections 3.1 and 3.2).

As with the other Sparql structures, OPTIMAL combines queries $\mathcal{Q}_1$ and $\mathcal{Q}_2$ to an *optimal pattern* $\mathcal{Q}_1$ OPTIMAL $\mathcal{Q}_2$. The intuition of optimal patterns is that we state a preference for matches that are complete w. r. t. $\mathcal{Q}_1$ and $\mathcal{Q}_2$, i. e., in the *optimal case*, we get the matches of $\mathcal{Q}_1$ AND $\mathcal{Q}_2$. Only if this goal is impossible to reach, i. e., there could be matches of $\mathcal{Q}_1$ but no compatible ones of $\mathcal{Q}_2$, we expect the matches of $\mathcal{Q}_1$. The semantics of optimal patterns will adhere to the following property:

$$[\![\mathcal{Q}_1 \ \mathsf{AND}\ \mathcal{Q}_2]\!]_{\mathbf{G}} \subseteq [\![\mathcal{Q}_1 \ \mathsf{OPTIMAL}\ \mathcal{Q}_2]\!]_{\mathbf{G}} \subseteq [\![\mathcal{Q}_1 \ \mathsf{OPTIONAL}\ \mathcal{Q}_2]\!]_{\mathbf{G}} \qquad (1)$$

More concretely, if there are matches of $\mathcal{Q}_1$ AND $\mathcal{Q}_2$ in $\mathbf{G}$, then $[\![\mathcal{Q}_1 \ \mathsf{OPTIMAL}\ \mathcal{Q}_2]\!]_{\mathbf{G}} = [\![\mathcal{Q}_1 \mathsf{AND} \mathcal{Q}_2]\!]_{\mathbf{G}}$. Unlike the respective optional pattern $\mathcal{Q}_1 \mathsf{OPTIONAL} \mathcal{Q}_2$, we do not include a single match of only $\mathcal{Q}_1$ in this case. This matching behavior allows us to express preferences over the completeness of the matches w. r. t. a given RDF graph. Only if $[\![\mathcal{Q}_1 \mathsf{AND} \mathcal{Q}_2]\!]_{\mathbf{G}} = \emptyset$, the respective semantics of optimal patterns and optional patterns align.

As long as $\mathcal{Q}_1$ and $\mathcal{Q}_2$ are free of optimal patterns, the previous paragraph tells us the whole story of the new operator. The combination of (several) optimal patterns with filter constraints allows for formulating meaningful preference patterns on data values and structure.

*Example 1.* OPTIMAL can be used to model preferences over attribute values. A simple price preference could be: *I prefer cheap cars under 15,000€ over other cars under 20,000€ over more expensive cars.* It can be adequately modeled in the form of: `?car rdf:typeof Car`

```
OPTIMAL { ?car ex:price ?price FILTER( ?price < 15000 ) }
OPTIMAL { ?car ex:price ?price FILTER( ?price < 20000 ) }
```

The query will return all cars that are cheaper than 15,000€ if any exist. If not, then it will return all cars that are cheaper than 20,000€ instead. If no such car exists either, then the query will return all cars in the data set that are more expensive or have no available price information.

*Example 2.* A user can pose structural preferences as well. Such a query could be *I prefer to see cars with available price and brand information.* Such a query could be adequately modeled as: `?car rdf:typeof Car`

```
OPTIMAL { ?car ex:price ?price }
OPTIMAL { ?car ex:brand ?brand }
```

The query will return all cars with information for price and brand. If no such cars can be retrieved, it will instead return all cars with price information. If that also results in an empty set, then the query will return all cars with brand information instead. Only if all of these attempts yield empty results, then the query will return all cars instead.

The last example shows how optimal patterns may be used to impose an ordering on the preferred structure. To allow for the formulation of preferences of equal importance, we extended the syntax of optimal patterns $\mathcal{Q}_1 \,\mathsf{OPTIMAL}\, \mathcal{Q}_2$, allowing $\mathcal{Q}_2$ to be a finite list of queries $\mathcal{Q}_2^1, \mathcal{Q}_2^2, \ldots, \mathcal{Q}_2^n$, i. e., the syntax of these optimal patterns is

$$\mathcal{Q}_1 \,\mathsf{OPTIMAL}\, (\mathcal{Q}_2^1, \mathcal{Q}_2^2, \ldots, \mathcal{Q}_2^n).$$

The intuition is still that $\mathcal{Q}_1$ is necessary to be matched. However, a simple correspondence as in Equation (1) cannot be derived. In the *optimal case*, we find matches of all of the queries in the optional pattern, i. e.,

$$[\![\mathcal{Q}_1 \,\mathsf{AND}\, \mathcal{Q}_2^1 \,\mathsf{AND}\, \mathcal{Q}_2^2 \,\mathsf{AND} \ldots \mathsf{AND}\, \mathcal{Q}_2^n]\!]_{\mathbf{G}} \subseteq [\![\mathcal{Q}_1 \,\mathsf{OPTIMAL}\, (\mathcal{Q}_2^1, \mathcal{Q}_2^2, \ldots, \mathcal{Q}_2^n)]\!]_{\mathbf{G}}.$$

Whenever we cannot reach the optimal case, we will obtain *dominant matches* as answers.

*Example 3.* Given the query from Example 2, a possible answer could be a car for a price of 17,000€ but of no known brand. This result contains more information than one with completely unknown attributes. Thus, the first match dominates the second one. A match concerning a third car with a price of 18,000€ and a known brand contains even more information on the other hand. Thus, the first match is dominated by the third one. If we consider price and brand to be of equal importance, then the first match would not dominate a fourth one that only has brand information. The two matches would be incomparable. If we consider the price to be more important instead, then the fourth match does not appear in the result set. The just sketched process is like asking for a skyline over the price (which does not include the fourth match) and then construct a skyline over the brand for the previous results.

Hence, the new construct requires a Pareto-style semantic, which entails a *skyline of matches*. Subsequently, we order candidate matches by *Pareto dominance*. Only the maximal candidates (w. r. t. Pareto dominance) are considered matches of optimal patterns.

**Definition 1 (Candidate Matches).** *Let $\mathcal{Q}$ be an optimal pattern, i. e.,*

$$\mathcal{Q} = \mathcal{Q}_0 \,\mathsf{OPTIMAL}\, (\mathcal{Q}_1, \mathcal{Q}_2, \ldots, \mathcal{Q}_k)$$

*for some integer $k > 0$. A partial function $\mu : \mathbf{VIL} \to \mathbf{IL}$ is called a* candidate match of $\mathcal{Q}$ in $\mathbf{G}$ *iff there are $\mu_0, \mu_1, \mu_2, \ldots, \mu_k$, such that 1. $\mu_0 \in [\![\mathcal{Q}_0]\!]_{\mathbf{G}}$, 2. $\mu_i \in [\![\mathcal{Q}_i]\!]_{\mathbf{G}} \cup \{\emptyset_{\mathbf{IL}}\}$ $(0 < i \leq k)$, 3. $\mu_i \leftrightarrows \mu_j$ for all $i, j \in \{0, 1, 2, \ldots, k\}$, and 4. $\mu = \mu_0 \cup \mu_1 \cup \mu_2 \cup \ldots \cup \mu_k$.*

The first requirement accounts for the fact that $\mathcal{Q}_0$ is the *necessary pattern* to be matched. Note, every other part of $\mu$ may be the empty match $\emptyset_{\mathbf{IL}}$. Using the separation of candidate matches $\mu$ of $\mathcal{Q}$ (in $\mathbf{G}$) into $\mu_0, \mu_1, \mu_2, \ldots, \mu_k$, we say that $\mu$ *covers* $\mathcal{Q}_i$ *$(0 \leq i \leq k)$* iff $\mu_i \neq \emptyset_{\mathbf{IL}}$. Note, $\mathcal{Q}_0$ must be covered by all candidate matches. We denote by $\mathsf{cover}_{\mathcal{Q}}(\mu)$ the set of all sub-queries

$\mathcal{Q}_j \in \{\mathcal{Q}_0, \mathcal{Q}_1, \mathcal{Q}_2, \ldots, \mathcal{Q}_k\}$ covered by $\mu$. Then the semantics of optimal patterns boils down to the maximal matches w. r. t. inclusion of the sets of covered subqueries.

**Definition 2 (Semantics of optimal patterns).** *Let $\mathcal{Q}$ be an optimal pattern, $\mathbf{G}$ an RDF graph, and $\mu, \mu'$ two candidate matches of $\mathcal{Q}$ in $\mathbf{G}$. $\mu'$* dominates *$\mu$* w. r. t. *$\mathcal{Q}$, denoted by $\mu \prec_{\mathcal{Q}} \mu'$, iff $\mathsf{cover}_{\mathcal{Q}}(\mu) \subsetneq \mathsf{cover}_{\mathcal{Q}}(\mu')$.*

*Candidate match $\mu$ of $\mathcal{Q}$ in $\mathbf{G}$ is called a* match of $\mathcal{Q}$ in $\mathbf{G}$ *iff there is no candidate match $\mu'$ of $\mathcal{Q}$ in $\mathbf{G}$ with $\mu \prec_{\mathcal{Q}} \mu'$. The set of all matches of $\mathcal{Q}$ in $\mathbf{G}$ is denoted by $[\![\mathcal{Q}]\!]_{\mathbf{G}}$.*

This formalization allows us to derive correctness of (1) for the special case of optimal patterns with a single query as the right-hand side.

**Proposition 1.** *For all* Sparql *queries $\mathcal{Q}_1$ and $\mathcal{Q}_2$,* (1) *holds.*

*Proof.* Let $\mu$ be a match of $\mathcal{Q}_1$ $\mathsf{AND}$ $\mathcal{Q}_2$. Then there are matches $\mu_i \in [\![\mathcal{Q}_i]\!]_{\mathbf{G}}$ $(i = 1, 2)$ with $\mu = \mu_1 \cup \mu_2$ and $\mu_1 \leftrightarrows \mu_2$. Thus, $\mu$ is a candidate match of $\mathcal{Q}' = \mathcal{Q}_1$ $\mathsf{OPTIMAL}$ $\mathcal{Q}_2$. Furthermore, every match of $\mathcal{Q}_1$ $\mathsf{AND}$ $\mathcal{Q}_2$ covers $\mathcal{Q}_1$ and $\mathcal{Q}_2$ in $\mathcal{Q}'$. Hence, $\mu$ cannot be dominated by any other match of $\mathcal{Q}'$ in $\mathbf{G}$.

Let $\mu'$ be a match of $\mathcal{Q}' = \mathcal{Q}_1$ $\mathsf{OPTIMAL}$ $\mathcal{Q}_2$. We need to show that $\mu'$ is a match of $\mathcal{Q}_1$ $\mathsf{OPTIONAL}$ $\mathcal{Q}_2$. Towards a contradiction assume, $\mu'$ is not a match of $\mathcal{Q}_1$ $\mathsf{OPTIONAL}$ $\mathcal{Q}_2$. Then $\mu' \in [\![\mathcal{Q}_1]\!]_{\mathbf{G}}$ but there is a match $\mu'' \in [\![\mathcal{Q}_2]\!]_{\mathbf{G}}$, such that $\mu' \leftrightarrows \mu''$. But then $\mu' \cup \mu''$ is a candidate match of $\mathcal{Q}'$ and

$$\mathsf{cover}_{\mathcal{Q}'}(\mu' \cup \mu'') = \{\mathcal{Q}_1, \mathcal{Q}_2\} \supsetneq \{\mathcal{Q}_1\} = \mathsf{cover}_{\mathcal{Q}'}(\mu'),$$

which contradicts the assumption that $\mu'$ is a match of $\mathcal{Q}'$, i. e., $\mu'$ is not dominated by any other candidate match. $\square$

Subsequently, we justify the syntax style of query lists in optional patterns and we consider RDF's *open world assumption* in the context of optimal patterns.

### 3.1 Justifying Optimal Patterns

Our divergence from the standard binary operator structure of Sparql may seem peculiar at first. However, this was a necessary step to allow for expressing preferences of equal importance. Suppose, we want to express a preference of query $\mathcal{Q}$ over $\mathcal{Q}_1$ and $\mathcal{Q}_2$, but $\mathcal{Q}_1$ and $\mathcal{Q}_2$ are equally important. Without the query list notation, we have several candidates using optimal patterns and standard Sparql operators:

- $\mathcal{Q}$ $\mathsf{OPTIMAL}$ $(\mathcal{Q}_1$ $\mathsf{UNION}$ $\mathcal{Q}_2)$: This query returns all the matches of $\mathcal{Q}$, preferably joined with matches of $\mathcal{Q}_1$ or $\mathcal{Q}_2$. However, as soon as $\mathcal{Q}_1$ and $\mathcal{Q}_2$ have a shared variable, we will hardly see any matches fulfilling all three sub-queries.
- $\mathcal{Q}$ $\mathsf{OPTIMAL}$ $(\mathcal{Q}_1$ $\mathsf{AND}$ $\mathcal{Q}_2)$: From this query we may expect only matches that fulfill all three sub-queries or only $\mathcal{Q}$. Hence, if $\mathcal{Q}_1$ cannot be matched, we will also not see matches that fulfill $\mathcal{Q}$ and $\mathcal{Q}_2$.

– ($\mathcal{Q}$ OPTIMAL $\mathcal{Q}_1$) AND ($\mathcal{Q}$ OPTIMAL $\mathcal{Q}_2$): This query is quite close to the desired Pareto query. Here, $\mathcal{Q}_1$ and $\mathcal{Q}_2$ are seen as independent through the use of query conjunction. For queries of that shape we may actually observe matches fulfilling all three sub-queries or only a subset of these. However, this query also accommodates some weird matching behavior. Suppose, $\mathcal{Q}_1$ and $\mathcal{Q}_2$ share a variable $\mathsf{x}$, that does not occur in $\mathcal{Q}$, but $\mathcal{Q}_1$ and $\mathcal{Q}_2$ cannot simultaneously be matched in some RDF graph $\mathbf{G}$. Then the query above may easily yield no results at all, although $\mathcal{Q}$ may be matched together with $\mathcal{Q}_1$ or $\mathcal{Q}_2$ separately.

Besides the possibilities above, it is always possible to unfold the Pareto query exponentially. However, such queries are hardly readable and maintainable. Therefore, we decided to expand SPARQL's operator set in this respect.

### 3.2 Optimal Patterns and Certain Answers

How well do optimal patterns cope with RDF's *open world assumption* (OWA). The OWA influences the way the query semantics is actually executed. We often see query semantics following the *certain answers* perspective [2]. A match is a *certain answer of a query $\mathcal{Q}$ in some OWA database* iff the match is a match in every possible interpretation of the OWA database. Regarding RDF, every superset RDF graph $\mathbf{H}$ of $\mathbf{G}$ is a possible interpretation of $\mathbf{G}$. According to Arenas and Pèrez, the certain answers perspective for SPARQL is manifested by

$$\textsc{CertainAnswers}(\mathcal{Q},\mathbf{G}) := \textstyle\bigcap_{\mathbf{H}\supseteq\mathbf{G}} [\![\mathcal{Q}]\!]_{\mathbf{H}}.$$

Iterating over all infinitely many extensions $\mathbf{H}$ of $\mathbf{G}$ is infeasible in practice. Therefore, handy characterizations can be proven. One such characterization is concerned with the *monotonicity* of the given query $\mathcal{Q}$. A query $\mathcal{Q}$ is *monotone* iff for all RDF graph $\mathbf{G} \subseteq \mathbf{H}$, $[\![\mathcal{Q}]\!]_{\mathbf{G}} \subseteq [\![\mathcal{Q}]\!]_{\mathbf{H}}$, i. e., adding more information may only lead to more query results. Since more information may lead to different matches that cover more sub-queries, Arenas and Pèrez introduced the notion of *weak monotonicity.* A query $\mathcal{Q}$ is *weakly monotone* iff for all RDF graphs $\mathbf{G} \subseteq \mathbf{H}$. $\mu \in [\![\mathcal{Q}]\!]_{\mathbf{G}}$ implies the existence of a $\mu' \in [\![\mathcal{Q}]\!]_{\mathbf{H}}$ with $\mu \subseteq \mu'$. It can easily be shown that in the case of (weakly) monotone queries $\mathcal{Q}$, $\textsc{CertainAnswers}(\mathcal{Q},\mathbf{G})$ coincides with $[\![\mathcal{Q}]\!]_{\mathbf{G}}$.

Unfortunately, not all queries containing optimal patterns are weakly monotone. For instance, the third query shape in Sect. 3.1 is not monotone. Consider $\mathcal{Q}_1$ and $\mathcal{Q}_2$ to share a variable $\mathsf{x}$, that is not shared with $\mathcal{Q}$. Now construct $\mathbf{G}$ in such a way that it fulfills $\mathcal{Q}$ and $\mathcal{Q}_1$ but not $\mathcal{Q}_2$. Disjointly add a minimal structure to $\mathbf{G}$ that fulfills $\mathcal{Q}$ and $\mathcal{Q}_2$ but not $\mathcal{Q}_1$. Then we simultaneously match $\mathcal{Q}$ and $\mathcal{Q}_1$ as well as $\mathcal{Q}$ and $\mathcal{Q}_2$, but the respective matches are necessarily in conflict in $\mathsf{x}$. Therefore, the result set will easily be empty, where it used to be non-empty when $\mathcal{Q}_2$ was violated.

Hence, optimal patterns are neither better nor worse than optional patterns, at least w. r. t. certain answers. It is simply two different styles of querying and we have to let the user decide which style is appropriate in which scenarios.

## 4 Adding Dependencies over Preferences

In this section we propose a solution for modeling dependencies by introducing an additional set of operators that complement OPTIMAL. Preferences may exist in various different types and combinations. Being able to model such dependencies is necessary to adequately model complex user requests.

Sometimes a subquery preference simply cannot return useful results for a given preference unless another part of the query also returns valid matches. This is the case when a query asks for a possibly missing node that is only indirectly related to an enforced match. For example, a query may ask for a person's car as well as their car's model. The person's car can be trivially retrieved. However, if that person does not own a car, it is impossible to give an adequate answer regarding the car's model. We call this kind of dependency a *structural dependency* because these directly depend on the (in-)completeness in the knowledge graph structure. In contrast, we call all dependencies that relate to data values within the modeled preference as *non-structural dependencies*, e. g., a user may prefer the color red for sports cars but not necessarily for other cars as well.

If one preference cannot be answered unless the found answers satisfy a specified condition, i. e., also cover a more relevant preference, we speak of a *positive dependency*. The previously given examples fall into this category. We speak of negative dependencies if one preference should only be answered if a specified condition remains unsatisfied, i. e., a specific and more relevant preference is not covered. For example, a user may prefer the color blue for any car that is not a sports car with either no color preference, or a different one for sports cars. Negative dependencies also model different alternatives of unequal importance, e. g., a user interested in buying a car needs to know about the price of the object. Most likely, that user will prefer offers made in his own currency. If no suitable matches can be found, offers in foreign currencies may still be relevant.

Dependencies are not guaranteed to be 1:1 relations. Especially structurally dependent preferences often rely on the same condition, implying a 1:m relation. For example, asking for a person's car and its properties will result in a structural dependency of all properties towards the retrieval of a matching car. When modeling multiple alternatives of unequal importance, the opposite behavior is true. For example, a car manual should only be displayed in an unknown language if it is not available in any language the user is fluent in. This yields an n:1 relationship between preferences. Mixed types of questions can also lead to general n:m relations between preferences which need to be accounted for.

### 4.1 Limitations of OPTIMAL

Expressing interdependent preferences is already possible using existing SPARQL operators which requires significant effort, even more so than individual preferences. By the new OPTIMAL operator we have developed a tool to ease this process. We have also shown how it can be used for individual preferences. However, it is still not ideal for modeling dependencies.

Let us assume a user is looking for a car, preferably a sports car. If it happens to be a sports car, then the user prefers it to be red – otherwise any color is acceptable. Simultaneously, the user also prefers cars with available price information. Using multiple optimal patterns in a naive way (as shown in Example 4) does not represent these requirements.

*Example 4.* The following query will look for all cars, preferably sports cars. Among those, it will look for red ones. This works well if the database contains any sports car.

```
?car rdf:typeof Car
OPTIMAL { ?car rdf:typeof SportsCar, ?car ex:hasPrice ?price }
OPTIMAL { ?car ex:hasColor "red" }
```

Additionally, the query also describes a dominance of red cars over non-red cars.

A more accurate model can be achieved by nesting one optimal pattern into another one, being a technique also commonly used when modeling interdependent optional patterns.

*Example 5.* The following query looks for sports cars with red color first and then combines it with other preferences. As a result, red sports cars will be preferred over other sports cars, and sports cars in general will be preferred over other cars.

```
?car rdf:typeof Car OPTIMAL {
   ?car rdf:typeof SportsCar OPTIMAL { ?car ex:hasColor "red" },
   ?car ex:hasPrice ?price }
```

This time, red cars that are no sports cars will no longer be preferred over other cars. So we managed to fix the underlying problem in our model. However, the model is still not correct. Because the color preference is evaluated first, red sports cars will always be ranked better than non-red sports cars. This also means that a red sports car without price will be ranked higher than a non-red sports car with a price. In our use case, these two preferences are meant to be incomparable.

Modeling preferences with optimal patterns alone causes side effects by introducing unwanted ranking of actually incomparable query answers. For sure, it is always possible to model them correctly using complex filter conditions (cf. encodings in [1]). However, our goal is to ease the modeling process. The required use of unintuitive filters is very similar to the original motivation. Thus, we would like to have a simple set of operators that lets a user define dependencies between preferences directly.

## 4.2 Syntax and Semantics of THEN and OTHERWISE

In order to enable an easy way of modeling dependencies, we introduce two new operators called THEN and OTHERWISE. Both operators can be used on the right-hand side of an OPTIMAL operator in order to fine-tune requirements of a preference's matching behavior. `A THEN B` defines a positive dependency of

B towards A, meaning that preference B should only be answered if preference A also yields a non-empty set of answers. A OTHERWISE B, on the other hand, defines a negative dependency of B towards A, meaning that preference B is only answered if preference A does return an empty set of answers.

For our OPTIMAL operator we have already introduced lists of preferences for simultaneous evaluation (cf. Sect. 3). We apply the same concept to both sides of the new operators. For a preference A and a set of dependent preferences B towards A, A THEN $(b_1, b_2, ..., b_m)$ applies a matching restriction for all dependent preferences $b_j \in B$. They will only be answered if preference A is covered as well. Similarly, A OTHERWISE $(b_1, b_2, ..., b_m)$ applies a negative dependency restriction to all $b_j \in B$. For a set of independent preferences A and dependent preference B towards A, $(a_1, a_2, ..., a_n)$ THEN B applies a matching restriction on B. It will only be answered if at least one $a_i \in A$ is covered. When modeling a restriction towards all preferences in A (instead of any member) a simple conjunction can be used in order to turn the conditions into a single one. This method is best suited for handling alternative routes that lead to similar objects. Likewise,$(a_1, a_2, ..., a_n)$ OTHERWISE B applies a matching restriction on B. It will only be answered if none of the preferences $a_i \in A$ are covered. If both A and B are a set of preferences then both the respective restrictions apply for all $b_j \in B$ towards all $a_i \in A$.

Just like the OPTIMAL operator, we define THEN and OTHERWISE as left-associative. That way, the syntax for all following preferences $B_i \in \{B_1, ..., B_n\}$ of the form $A \odot_1 B_1 \odot_2 ... \odot_n B_n$ (with $\odot_i \in \{\mathsf{THEN}, \mathsf{OTHERWISE}\}$) will always imply a relation towards A, making the query easier to read. Query groups and preference list separators can be used to model the order of execution explicitly.

*Example 6.* The following query adequately models the query we were looking for in Sect. 4.1 and thereby resolves our issues with optimal patterns.

```
?car rdf:typeof Car OPTIMAL
({ ?car rdf:typeof SportsCar } THEN { ?car ex:color "red" },
?car ex:price ?price )
```

*Example 7.* In the following query (with negative dependencies), we consider the price in US-Dollar as a fallback if no offers in Euros can be found. Finding both currencies is not better than only Euros, though.

```
?car rdf:typeof Car OPTIMAL
({ ?car ex:price_EUR ?p_eur } OTHERWISE
{ ?car ex:price_USD ?p_usd })
```

### 4.3 Encoding Dependencies

In our previous work [1], we have demonstrated that optimal patterns can be encoded using existing query operators in order to work under standard SPARQL 1.0 semantics. The same holds for the new set of operators. We have found two different styles of encodings – one using UNION and one using OPTIONAL. Since the use of optional patterns follows more or less standard procedure, we restrict our presentation to the union-style encoding.

The encoding using UNION has proven to be more stable regarding performance. For this style we first construct a superset of desired results by combining all possible preference combinations. For every such combination we bind a fresh variable in the respective subquery to mark which combination an answer candidate is generated from. As a next step, we retrieve the results for every subquery a second time (but with changed variable names) inside an `EXISTS`-operator. That way, we can determine which combinations yield at least one candidate match. Lastly, we use `FILTER`-conditions to remove any answers that are dominated by at least one combination with a non-empty set of matches.

This method is easily changed to adapt for the new operators. When constructing the superset, we have to redefine what a *possible combination* is. Every THEN or OTHERWISE operation results in additional constraints on this set. A preference `A` that is dependent on `B` is not supposed to be matched unless a mapping for `B` also appears in the answer. Thus, any combination including mappings for `A` but not `B` has to be removed from the superset. Likewise, a preference `A` that has a negative dependency on `B` is only supposed to match if no mapping for `B` appears in the answer. Thus, any combination including mappings for both `A` and `B` has to be removed from the superset. Our prototypical implementation[5] can easily be extended.

In terms of performance, we leave out an extensive analysis here. Usage of the new operators only results in removal of answer candidates. Thus, any encoded query using THEN or OTHERWISE will be shorter and most likely slightly faster than an independent preference.

## 5 Conclusion

We have presented a new set of operators for expressing structural preferences over the completeness of knowledge graph query results. The primary extension consists of so-called *optimal patterns*, which we introduced in [1]. Here, we added the possibility of stating semantic dependencies between conflicting preferences. Furthermore, we discussed fundamental design decisions.

Although our discussion regarding certain answers (cf. Sect. 3.2) attests optimal patterns to have a similar behavior as optional patterns, a detailed discussion about the expressive power/complexity may still reveal important differences. For future work, we plan to overcome the limitations we observed in [1] with our encoding approach of evaluating optimal patterns by implementing custom evaluation strategies that may even use Pareto-specific optimizations [3,9,7].

*Acknowledgements.* This work is partly supported by Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) in project number 389792660 (TRR 248, Center for Perspicuous Systems) and Emmy Noether grant KR 4381/1-1 (DIAMOND).

[5] available at Github: `https://github.com/ifis-tu-bs/optisparql`

## References

1. Affeldt, T., Mennicke, S., Balke, W.T.: Preference-driven Control over Incompleteness of Knowledge Graph Query Answers. In: 12th ACM Web Science Conference 2020. WebSci'20, Southampton, United Kingdom, Association for Computing Machinery, New York, NY, USA (July 2020)
2. Arenas, M., Pérez, J.: Querying Semantic Web Data with SPARQL. In: Proceedings of the Thirtieth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems. pp. 305–316. PODS '11, ACM, New York, NY, USA (2011), event-place: Athens, Greece
3. Balke, W.T., Güntzer, U., Zheng, J.X.: Efficient Distributed Skylining for Web Information Systems. In: Bertino, E., Christodoulakis, S., Plexousakis, D., Christophides, V., Koubarakis, M., Böhm, K., Ferrari, E. (eds.) Advances in Database Technology - EDBT 2004. pp. 256–273. Lecture Notes in Computer Science, Springer, Berlin, Heidelberg (2004)
4. Cheng, S., Hartig, O.: OPT+: A Monotonic Alternative to OPTIONAL in SPARQL. Journal of Web Engineering **18**(1), 169–206 (2019)
5. Gueroussova, M., Polleres, A., McIlraith, S.A.: Sparql with qualitative and quantitative preferences. In: OrdRing@ ISWC. pp. 2–8 (October 2013)
6. Kaminski, M., Kostylev, E.V.: Beyond Well-designed SPARQL. In: Martens, W., Zeume, T. (eds.) 19th International Conference on Database Theory (ICDT 2016). Leibniz International Proceedings in Informatics (LIPIcs), vol. 48, pp. 5:1–5:18. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany (2016)
7. Keles, I., Hose, K.: Skyline Queries over Knowledge Graphs. In: Ghidini, C., Hartig, O., Maleshkova, M., Svátek, V., Cruz, I., Hogan, A., Song, J., Lefrançois, M., Gandon, F. (eds.) The Semantic Web – ISWC 2019. pp. 293–310. Lecture Notes in Computer Science, Springer International Publishing, Cham (2019)
8. Kießling, W.: Foundations of preferences in database systems. In: Proceedings of the 28th international conference on Very Large Data Bases. pp. 311–322. VLDB '02, VLDB Endowment, Hong Kong, China (2002)
9. Morse, M., Patel, J.M., Jagadish, H.V.: Efficient skyline computation over low-cardinality domains. In: Proceedings of the 33rd international conference on Very large data bases. pp. 267–278. VLDB '07, VLDB Endowment, Vienna, Austria (Sep 2007)
10. Pivert, O., Slama, O., Thion, V.: Sparql extensions with preferences: A survey. In: Proceedings of the 31st Annual ACM Symposium on Applied Computing. p. 1015–1020. SAC '16, Association for Computing Machinery, New York, NY, USA (2016). https://doi.org/10.1145/2851613.2851690
11. Prud'hommeaux, E., Seaborne, Andy: SPARQL Query Language for RDF. Tech. rep., W3C (2008), `https://www.w3.org/TR/rdf-sparql-query/`
12. Pérez, J., Arenas, M., Gutierrez, C.: Semantics and Complexity of SPARQL. ACM Trans. Database Syst. **34**(3), 16:1–16:45 (Sep 2009)
13. Schmidt, M., Meier, M., Lausen, G.: Foundations of SPARQL Query Optimization. In: Proceedings of the 13th International Conference on Database Theory. pp. 4–33. ICDT '10, ACM, New York, NY, USA (2010), event-place: Lausanne, Switzerland
14. Siberski, W., Pan, J.Z., Thaden, U.: Querying the semantic web with preferences. In: International Semantic Web Conference. pp. 612–624. Springer (2006)
15. Troumpoukis, A., Konstantopoulos, S., Charalambidis, A.: An extension of sparql for expressing qualitative preferences. In: International Semantic Web Conference. pp. 711–727. Springer (July 2017)