SPARQL with Qualitative and Quantitative Preferences
(Extended Report)

Marina Gueroussova Axel Polleres* Sheila A. McIlraith
Department of Computer Science, University of Toronto, Toronto, Canada
* Vienna University of Economics and Business (WU Wien), Vienna, Austria

October 21, 2013

Abstract

The volume and diversity of data that is queriable via SPARQL and its increasing integration motivate the desire to query SPARQL information sources via the specification of preferred query outcomes. Such preference-based queries support the ordering of query outcomes with respect to a user's measure of the quality of the response. In this report we argue for the incorporation of preference queries into SPARQL. We propose an extension to the SPARQL query language that supports the specification of qualitative and quantitative preferences over query outcomes and examine the realization of the resulting preference-based queries via off-the-shelf SPARQL engines.

1 Introduction

Once the sole purview of IT departments, today's data is produced, shared, and consumed by a diversity of stakeholders – corporate and consumer. It is stored in a variety of formats, dynamically integrated, and queried by a variety of users, many of whom are largely unfamiliar with the content of those sources. As such, querying today's data sources using standard query languages is evolving into a manual/iterative search process, in which repeated queries are devised to hone in on outcomes that meet some criteria. Consider looking for a car to buy on the web – one might prefer a car with a powerful engine, but only if it's a hybrid, or failing that an electric car if it's under a certain price, and so on. Such a query requires not only the specification of the information to be returned, but also specification of an ordering or preference over what is returned. With these fundamental changes in the nature of data management and querying comes the need for query languages and engines that are better suited to the specification of and search for high-quality outcomes.

Preferences have been a long studied subject across many fields including economics, philosophy, and artificial intelligence. Within the database community there is a growing literature on preferences (e.g., (Stefanidis, Koutrika, & Pitoura, 2011)). Indeed, both Chomicki (Chomicki, 2002, 2003, 2011) and independently, Kießling and colleagues, including Köstler, Endres, and Wenzel (Kießling, 2002; Kießling & Köstler, 2002; Kießling, Endres, & Wenzel, 2011) have developed foundational theories relating preferences to database systems and have proposed extensions to SQL that support the specification of quantitative (e.g., top-k (Ilyas, Beskales, & Soliman, 2008)) and qualitative (e.g., skyline (Börzsönyi, Kossmann, & Stocker, 2001)) SQL queries. Top-k queries use a scoring function to determine an ordering over query results. In contrast, skyline queries filter a dataset with respect to a set of preference relations, returning a set of undominated tuples.

Our concern in this paper is with SPARQL (Harris & Seaborne, 2013) and with the provision of a means of succinctly specifying queries that will enable a user to search for data sources (SPARQL endpoints) and data content that is tailored to their individual preferences, and in turn for SPARQL query engines to return ordered outcomes that reflect those preferences. Within the semantic web community, there has been significant recent work on the computation of top-k queries (e.g., (Montoya, Vidal, Corcho, Ruckhaus, & Aranda, 2012; Bozzon, Valle, & Magliacane, 2012; Magliacane, Bozzon, & Valle, 2012; Wagner, Tran, Ladwig, Harth, & Studer, 2012)), but little on how to extend the expressiveness of SPARQL to address a broad spectrum of qualitative and quantitative preferences. As such, (qualitative) preference-based querying is often realized by multiple lengthy queries that stipulate different combinations of hard constraints, or via "standard tricks[1]" such as "stacking" OPTIONAL patterns, as illustrated in the following example which preferably returns the email address of my friends, and the homepage if there is no email.

```
SELECT ?Contact WHERE { me foaf:knows ?X
                        OPTIONAL {?X foaf:mbox ?Contact}
                        OPTIONAL {?X foaf:homepage ?Contact} }
```

In 2006 Siberski, Pan, and Thaden introduced the notion of qualitative preferences into SPARQL queries (Siberski, Pan, & Thaden, 2006). In that work they realized their preference queries through the development of solution modifiers. Interestingly, their work was done before the semantics of OPTIONAL was established, and it was our hypothesis that much of what they did in 2006 with solution modifiers could be done in native SPARQL 1.0 and SPARQL 1.1 through rewriting. Building on that work, we propose an extension of the SPARQL query language, PrefSPARQL that, like Siberski et. al.'s work, builds on the vetted work on SQL preference queries, extending it here to support the expression of conditional preferences. In Section 4 we focus on the realization of qualitative preferences, and in particular on skyline and conditional preferences, showing how they can be rewritten into SPARQL 1.1 (and also SPARQL 1.0) and thus realized by existing SPARQL query engines. The work presented here is only the first step of a larger endeavour that will see the extension of the presented query grammar with a number of interesting features, and the development of optimized query processing techniques that are tailored to the efficient computation of preference-based SPARQL queries.

2 PrefSPARQL Syntax

In this section we propose a core grammar for PrefSPARQL that addresses a selection of qualitative and quantitative preferences in support of specifying preferred SPARQL query outcomes. We illustrate our grammar with respect to skyline and conditional preference queries. Skyline queries for relational databases have been the subject of significant research (e.g., (Chomicki, 2011; Börzsönyi et al., 2001)), and refer to a set of results that are no worse than any other result across all dimensions of a set of independent boolean or numerical preferences (Börzsönyi et al., 2001). Skyline queries have been studied extensively in database systems, and explicitly in the SQL context by Chomicki (Chomicki, 2002, 2003, 2011), and by Kießling, Köstler, Endres, and Wenzel as an essential part of their *Preference SQL* language and system (Kießling, 2002; Kießling & Köstler, 2002; Kießling et al., 2011).

We build upon an earlier approach to adopt features of *Preference SQL* in SPARQL, by Siberski, Pan, and Thaden (Siberski et al., 2006). In particular, we extend the SPARQL query language in a similar fashion to the proposal in (Siberski et al., 2006); we also use 'AND' to separate independent dimensions of skyline queries. The key differences in our proposal are that, firstly, we add preferences at the level of filters

[1] e.g., `http://answers.semanticweb.com/questions/20682/preference-patterns-for-sparql-11`

(production 68 of the SPARQL grammar (Harris & Seaborne, 2013, Section 19)) rather than as solution modifiers, with the justification that preferences semantically *filter* the solution set rather than 'ordering' or 'slicing' it. This approach makes preferences usable inside any patterns in a nested fashion as opposed to just at the end of queries.[2] Secondly, we follow the latest version of *Preference SQL* (Kießling et al., 2011) in replacing 'CASCADE' with 'PRIOR TO'. Furthermore, we support additional atomic preference constructs such as 'BETWEEN', 'AROUND', 'MORE THAN', 'LESS THAN', 'HIGHEST', and 'LOWEST' ; 'x BETWEEN (*Low*,*High*)' differs from writing '$((x >= Low)$ && $(x <= High))$' in SPARQL in that – in the absence of a value in the chosen interval – 'BETWEEN' will return the closest value to *Low* (or *High*, resp.); AROUND(x), MORE THAN(x), and LESS THAN(x) are analogous to BETWEEN(x,x), BETWEEN(x,∞) and, BETWEEN($-\infty,x$) respectively; likewise, we note that'HIGHEST' and 'LOWEST' can be seen as syntactic sugar for AROUND(∞) and AROUND($-\infty$), respectively. In all these functions, one usually needs to assume a partially ordered domain, and also a distance function ($Abs(x-y)$). We note that in the general domain of SPARQL, an ordered domain cannot be assumed, since SPARQL expressions can return arbitrary RDF terms that are neither partially ordered, cf. (Harris & Seaborne, 2013, 15.1), nor is there a general distance. The present version of this report focuses on defining the syntax and overall semantics framework for PrefSPARQL, so we assume RDF terms to be partially ordered and a distance function being available for now, and leave a detailed treatment of these issues to future work.

Finally, we augment the grammar with conditional (IF-THEN-ELSE) preferences. We note that, given the availability of conditional preferences, BETWEEN (as well as AROUND, MORE THAN, and LESS THAN) come for free.[3]

```
Filter ::= 'FILTER' Constraint
           | 'PREFERRING' '(' MultidimensionalPref ')'
MultidimensionalPref ::= PrioritizedPref ('AND' PrioritizedPref)*
PrioritizedPref ::= ConditionalOrAtomicPref
                    ('PRIOR TO' ConditionalOrAtomicPref)*
ConditionalOrAtomicPref ::= ConditionalPref | AtomicPref
ConditionalPref ::= 'IF' Expression
                    'THEN' ConditionalOrAtomicPref
                    'ELSE' ConditionalOrAtomicPref
AtomicPref  ::= Expression | HighestPref | LowestPref | BetweenPref
                | AroundPref  | MoreThanPref | LessThanPref
HighestPref ::= 'HIGHEST' Expression
LowestPref  ::= 'LOWEST' Expression
BetweenPref ::= Expression 'BETWEEN' '(' Expression ',' Expression ')'
AroundPref  ::= Expression 'AROUND' Expression
MoreThanPref ::= Expression 'MORE THAN' Expression
LessThanPref ::= Expression 'LESS THAN' Expression
```

Note that in the grammar of the *Preference SQL* language (Kießling et al., 2011; Kießling & Köstler, 2002), which we base on, the nonterminal '<column>' is used in the productions for 'HIGHEST', 'LOWEST', 'BETWEEN', 'AROUND', 'MORE THAN', and 'LESS THAN' in the following manner: "<column> HIGHEST", "<column> LOWEST", "<column> BETWEEN ...", "<column> AROUND...", "<column>

[2] While with the addition of subqueries in SPARQL1.1 this does not add to language expressivity, it still allows one to write certain preference queries more concisely.

[3] x BETWEEN(*Low*,*High*) can be viewed as syntactic sugar for (IF $x >= Low$ && $x <= High$ THEN 0 ELSE IF $x < Low$ THEN $-Abs(Low-x)$ ELSE $-Abs(High-x)$). Note here that we want to prefer the smallest distance to the interval, which is why we use the *negated* absolute distance $-Abs(\cdot)$ in this expression.

MORE THAN...", and "<column> LESS THAN...". [4] While the use of columns in SQL would naturally correspond to 'Var' in SPARQL, we decided that we can allow arbitrary expressions in place of columns without adding expressive power: indeed, in SPARQL 1.1 (but not SPARQL 1.0) 'Var' can be substituted for 'Expression' in the above BNF without loss of expressivity, since any expression can be assigned to a variable via additional BIND ... AS clauses in a SPARQL pattern.

For example, in the SPARQL 1.1 flavour of PrefSPARQL:

```
PREFERRING IF(bound(?price_inEUR), ?price_inEUR, ?price_inUSD) MORE THAN 100
```

can instead be written as:

```
BIND( IF(bound(?price_inEUR), ?price_inEUR, ?price_inUSD) AS ?P)
PREFERRING ?P MORE THAN 100
```

In SQL, similar assignments of arbitrary expressions to columns are allowed with the keyword 'AS'. However, since BIND ... AS clauses are not available in SPARQL 1.0, we use 'Expression' rather than 'Var' in our grammar so as to retain the same expressivity also in the SPARQL1.0 flavour of PrefSPARQL.

To further illustrate PrefSPARQL, consider a modification of the example from (Siberski et al., 2006). The knowledge base contains therapist ratings and their appointment offerings (in Turtle syntax).

```
@prefix : <http://www.example.org/>.
:mary a :therapist ; :rated :excellent ;
        :offers :appointment1, :appointment2 .
:appointment1 :day "Tuesday"; :starts 1500; :ends  1555 .
:appointment2 :day "Sunday"; :starts 1600; :ends 1655 .
```

For the case of the skyline preference, query Q1 prefers excellent therapists, appointments around lunchtime (between 12:00 and 13:00) over those outside lunchtime (written as a BETWEEN preference), and later appointments over earlier ones provided that both are equal with respect to lunchtime.

```
SELECT ?A WHERE {
   ?T :rated ?R; :offers ?A. ?A :starts ?S; :ends ?E .
   PREFERRING ( ?R = excellent AND
                (?S BETWEEN(1200,1300) AND ?E BETWEEN(1200,1300)
                 PRIOR TO HIGHEST ?E)) }
```

For the case of conditional preferences, in query Q2 we prefer appointments before 6PM on the weekends, and appointments after 6PM on the weekdays.

```
SELECT ?A WHERE { ?A :day ?D; :starts ?S.
                PREFERRING ( IF (?D = "Saturday" || ?D = "Sunday")
                THEN ?S < 1800 ELSE ?S >= 1800 ) }
```

[4] Note that, having used Siberski et.al.'s (Siberski et al., 2006) grammar as a starting point, we follow them in placing 'Expression' after both 'HIGHEST' and 'LOWEST', slightly deviating from the Preference SQL grammar, cf. http://ursaminor.informatik.uni-augsburg.de/trac/wiki/Preference%20SQL%20Syntax.

3 PrefSPARQL Semantics

In keeping with prior work on preferences (e.g., (Chomicki, 2002, 2003, 2011; Kießling, 2002; Kießling & Köstler, 2002; Kießling et al., 2011; Siberski et al., 2006)), we define a preference relation among solutions inductively on the structure of $Pref$ as follows.

Definition 1 *Let $CoAPref$ be a* `ConditionalOrAtomicPref`*, and let $\Omega = [[P]]$ be a set of solutions to a SPARQL pattern P as defined in (Pérez, Arenas, & Gutierrez, 2009a; Polleres & Wallner, 2013) and let $\mu, \mu' \in \Omega$. We say that μ is preferred to μ' according to preference $Pref$, written $\mu >^{Pref} \mu'$, if:*

- *for $Pref = CoAPref$:*[5]
$$\mu >^{Pref} \mu' \equiv simplify^{\mu}(CoAPref) > simplify^{\mu'}(CoAPref)$$
- *for $Pref = Pref_1$ `AND` $Pref_2$:*
$$\mu >^{Pref} \mu' \equiv (\mu >^{Pref_1} \mu' \wedge \mu \not<^{Pref_2} \mu') \vee (\mu >^{Pref_2} \mu' \wedge \mu \not<^{Pref_1} \mu')$$
- *for $Pref = Pref_1$ `PRIOR TO` $Pref_2$:*
$$\mu >^{Pref} \mu' \equiv (\mu >^{Pref1} \mu') \vee (\mu >^{Pref_2} \mu' \wedge \mu \not<^{Pref_1} \mu')$$

Let E be a SPARQL Expression, then the function $simplify^{\mu}(CoAPref)$ expands conditional or atomic preferences as follows:

- *for $CoAPref = E$, $simplify^{\mu}(CoAPref) = \mu(E)$*
- *for $CoAPref =$* `IF` E `THEN` $CoAPref_1$ `ELSE` $CoAPref_2$*,*
$$simplify^{\mu}(CoAPref) = \begin{cases} simplify^{\mu}(CoAPref_1) & \text{if } EBV(\mu(E)) = true \\ simplify^{\mu}(CoAPref_2) & \text{otherwise} \end{cases}$$

Here EBV stands for the effective boolean value of an expression, as per (Harris & Seaborne, 2013, Section 17.2.2).

As mentioned above in Section 2, `BETWEEN`, `AROUND`, `MORE THAN`, `LESS THAN`, `HIGHEST`, `LOWEST`, may be viewed as syntactic sugar (which could be captured in an extension of the $simplify^{\mu}(\cdot)$ function) accordingly.[6]
Finally, the `PREFERRING` keyword expresses the set of results that are no worse than any other result according to a preferences $Pref$, wherefore the semantics of PrefSPARQL patterns is defined as

$$[[P \texttt{ PREFERRING } Pref]] = \{\mu \in [[P]] \mid \neg\exists \mu' \in [[P]] : \mu' >^{Pref} \mu\}$$

We note that `AND`-preferences can generalized to an arbitrary number of AND-ed preferences, as follows:

[5] as usual, for boolean values we assume that $true > false$

[6] We recall though from Section 2 that we assume an ordered domain as well as the existence of a distance function, i.e. all these $CoAPref$ can be viewed as a "scoring function" to induce an ordering. As mentioned before, such ordering is not applicable to all RDF terms or expressions in SPARQL in general, as they are not all comparable. We plan to address this in the future, by for instance ordering such terms based on the partial order implied by the `ORDER BY` solution modifier in SPARQL (Harris & Seaborne, 2013, Section 17.2.2), which still though does not prescribe, e.g., an order on RDF blank nodes.

- for $Pref = Pref_1$ AND ... AND $Pref_n$:

$$\mu >^{Pref} \mu' \equiv (\bigwedge_i \mu \not<_{Pref_i} \mu') \wedge (\bigvee_i \mu >_{Pref_i} \mu')$$

This models a skyline preference (Chomicki, 2011), that is, for such a multi-dimensional AND preferences we obtain exactly the skyline, defined as the set of results that are no worse than any other result across all dimensions of a set of independent (boolean or numerical) preferences (Börzsönyi et al., 2001).

4 Realizing PrefSPARQL Through Query Rewriting

As opposed to a contrary conjecture in (Siberski et al., 2006), we argue that preferences such as the ones presented in Section 2 *can* be expressed in SPARQL 1.0 and 1.1 itself by the following high-level translation schemas, where P is a SPARQL pattern and $Pref$ is a 'PREFERRING' clause as defined in the grammar above.

SPARQL 1.1:

$$\frac{P \texttt{ PREFERRING } Pref}{P \texttt{ FILTER NOT EXISTS } \{P' \texttt{ FILTER } (tr^{>}(Pref))\}}$$

SPARQL 1.0:

$$\frac{P \texttt{ PREFERRING } Pref}{P \texttt{ OPTIONAL } \{P' \texttt{ FILTER } (tr^{>}(Pref)) \texttt{ [] ?check []}\} \texttt{ FILTER } (\texttt{!bound(?check)})}$$

Here, P' is a copy of P where all variables are renamed with fresh variables (primed versions).

We will denote the variables occurring in P as $vars(P)$; lastly, given a ConditionalOrAtomicPref $CoAPref$ over variables in $vars(P)$, we write $CoAPref'$ to denote the expression obtained from replacing all variables in $vars(P)$ within $Expr$ by their primed versions. The high-level idea here is that we reformulate PREFERRING clauses as FILTERs, asking for the non-existence of a solution to the pattern P' which dominates the solution of P according to a preference clause $Pref$. While non-existence can be expressed straightforwardly in SPARQL1.1, we use a common trick of combining OPTIONAL with !bound(?check) where ?check is a fresh variable that can only be bound inside the OPTIONAL thereby emulating non-existence in SPARQL1.0. The auxiliary triple pattern {[] ?check []} used here binds any predicate, so it should always create at least some auxiliary binding in case some dominating solution exists.

This simple recipe, along with a translation of the domination relation as a FILTER expression suffices to express the intended semantics of returning only dominating solutions. The translation function $tr^{>/</=}(Pref)$, which is applied within this translation recursively, implements the domination order defined in the previous section. Depending on a preference expression $Pref$, $tr^{>/</=}(Pref)$ is defined as follows:

- for $Pref = Pref_1$ AND $Pref_2$:

$$tr^{>}(Pref) = (tr^{>}(Pref_1)\ \&\&\ !(tr^{<}(Pref_2)))\ ||\ (!(tr^{<}(Pref_1))\ \&\&\ tr^{>}(Pref_2))$$

- for $Pref = Pref_1$ PRIOR TO $Pref_2$:

$$tr^{>}(Pref) = tr^{>}(Pref_1)\ ||\ (tr^{>}(Pref_2)\ \&\&\ tr^{=}(Pref_1))$$

- for $Pref = CoAPref$ and $\circ \in \{<, >, =\}$

$$tr^{\circ}(Pref) = simplify(CoAPref') \circ simplify(CoAPref)$$

Here, we assume that $simplify(\cdot)$ is as defined in section 3 above, with the difference that it is no longer dependent on μ, since (a) expressions are just left 'as is', i.e.

- for $CoAPref = E$, $simplify(CoAPref) = E$

and (b) conditional (IF-THEN-ELSE) preferences are – recursively – rewritten to SPARQL1.1. 'IF$(\cdot,\cdot,\cdot)$' expressions or basic SPARQL1.0 expressions as follows, implementing exactly the intended semantics:[7]

- for $CoAPref =$ IF E THEN $CoAPref_1$ ELSE $CoAPref_2$

  (SPARQL 1.1) $simplify(CoAPref) = \mathtt{IF}(E, simplify(CoAPref_1), simplify(CoAPref_2))$

  (SPARQL 1.0) $simplify(CoAPref) = (E\ \&\&\ simplify(CoAPref_1))\ ||\ (!(E)\ \&\&\ simplify(CoAPref_2))$

Strictly speaking, we note here that the translation for SPARQL 1.0 only serves as an *approximation* for IF-THEN-ELSE, since in case where E evaluates to an error, the expression $(E\ \&\&\ simplify(CoAPref_1))\ ||\ (!(E)\ \&\&\ simplify(CoAPref_2))$ will also evaluate to error, whereas the ELSE branch should indeed be considered. This is due to the 3-valued semantics of FILTER expressions (cf. (Harris & Seaborne, 2013, Section 17.2)); it is thus up to the user to take care of appropriate error handling within the SPARQL1.0 setting.

Let us illustrate the translation by means of some examples: for skyline queries, we take a simplified version of Q1 from above without BETWEEN,[8] where we prefer appointments outside rush hour, i.e., either starting after 18:00 or ending before 16:00, instead of over lunchtime.

```
SELECT ?A
WHERE {
   ?T :rated ?R; :offers ?A. ?A :starts ?S; :ends ?E .
   PREFERRING ( (?R = excellent) AND
                ((?S >= 1800) || ?E <= 1600) PRIOR TO
                HIGHEST ?S ))}
```

This query translates to the following SPARQL1.1 query:[9]

[7] Given that 'IF$(\cdot,\cdot,\cdot)$' is not available in SPARQL 1.0, different versions of the translation of the conditional (IF-THEN-ELSE) preference are provided for SPARQL 1.0 and SPARQL 1.1

[8] As mentioned before, BETWEEN can be translated using conditional preferences.

[9] We use BIND...AS statements for the sake of clarity, to "collect" subexpressions subject to the TR $simplify(\cdot)$ function. While we haven't mentioned this explicitly in our translation this could be considered an optimization, since it also avoids re-computation of the same subexpressions at several places of the resulting FILTER. For an expanded translation without BIND...AS statements we refer to the Appendix.

```
1 SELECT ?A WHERE { ?T :rated ?R; :offers ?A. ?A :starts ?S; :ends ?E .
2   BIND ( (?R = :excellent)        AS ?Pref1 )
3   BIND ( (?S >= 1800 || ?E <= 1600) AS ?Pref2 )
4   BIND ( ?S                       AS ?Pref3 )
5   FILTER NOT EXISTS {
6      ?T_ :rated ?R_; :offers ?A_. ?A_ :starts ?S_; :ends ?E_ .
7      BIND ( (?R_ = :excellent)          AS ?Pref1_ )
8      BIND ( (?S_ >= 1600 || ?E_ <= 1600) AS ?Pref2_ )
9      BIND ( ?S_                         AS ?Pref3_ )
10     FILTER(
11        (  (?Pref1_ > ?Pref1) &&
12          !((?Pref2_ < ?Pref2) || (?Pref3_ < ?Pref3 && ?Pref2 = ?Pref2_ )))
13        ||
14        ( !(?Pref1_ < ?Pref1) &&
15           ((?Pref2_ > ?Pref2) || (?Pref3_ > ?Pref3 && ?Pref2 = ?Pref2_ ))))}}
```

In the copied pattern P' we replace every variable by a 'copy' appending '_' to the variable name. In both P and P' we bind every atomic expression in $Pref$ to a new variable (lines 2-4 and 7-9); then we use these variables to build up a FILTER expression that filters out 'dominating' witnesses for P; if no such witness exists, P survives. Let us explain briefly the rationale behind the translation of preference dominance in lines 11-15: the two dimensions (AND) of the skyline preference – i.e., that a dominating solution is better in at least one dimension and no worse in the others – are encoded in the two branches in lines 11-12 and lines 14-15, whereas the nested cascading (PRIOR TO) preference between $Pref2$ and $Pref3$ is encoded in the boolean expressions in lines 12 and 15, respectively.

Let us emphasize that the translation would also work without BINDs; the respective expressions could just be copied, although this would make the translated query more confusing.

As for the SPARQL1.0 version of the same query, note again that in SPARQL 1.0 `NOT EXISTS` is replaced by the well-known combination of `OPTIONAL` and `FILTER(!bound)` (cf. (Prud'hommeaux & Seaborne, 2008, Section 11.4.1)) to emulate set difference, plus there are no `BIND`s:

```
1 SELECT ?A WHERE { ?T :rated ?R; :offers ?A. ?A :starts ?S; :ends ?E .
2   OPTIONAL {
3     ?T_ :rated ?R_; :offers ?A_. ?A_ :starts ?S_; :ends ?E_ .
4     FILTER(
5        ( ((?R_ = :excellent) > (?R = :excellent)) &&
6          !(((?S_ >= 1600 || ?E_ <= 1600) < (?S >= 1800 || ?E <= 1600)) ||
7            (?S_ < ?S && (?S >= 1800 || ?E <= 1600) = (?S_ >= 1600 || ?E_ <= 1600))))
8        ||
9        ( !((?R_ = :excellent) < (?R = :excellent)) &&
10         (((?S_ >= 1600 || ?E_ <= 1600) > (?S >= 1800 || ?E <= 1600)) ||
11          (?S_ > ?S && (?S >= 1800 || ?E <= 1600) = (?S_ >= 1600 || ?E_ <= 1600)))))}
12   FILTER( !bound(?R_))}
```

As a further example, the translation of query `Q2` from above shows how conditional preferences can be encoded in SPARQL1.1 conveniently using the `IF(`$\cdot,\cdot,\cdot$`)` function.

```
1 SELECT ?A WHERE {
2   ?A :day ?D; :starts ?S.
3   BIND ( IF( (?D = "Saturday" || ?D = "Sunday"), ?S < 1800, ?S >= 1800)
4          AS ?Pref1)
5   FILTER NOT EXISTS { ?A_ :day ?D_; :starts ?S_.
6      BIND ( IF( (?D_ = "Saturday" || ?D_ = "Sunday"), ?S_ < 1800, ?S_ >= 1800)
```

```
7           AS ?Pref1_)
8      FILTER (?Pref1_ > ?Pref1)}}
```

Similarly, applying the alternative translation scheme for SPARQL1.0 on Q2 we obtain the following query:

```
1 SELECT ?A WHERE {
2   ?A :day ?D; :starts ?S.
3   OPTIONAL {
4       ?A_ :day ?D_; :starts ?S_.
5       FILTER((((?D_ = "Saturday" || ?D_ = "Sunday") && ?S_ < 1800) ||
6         (!(?D_ = "Saturday" || ?D_ = "Sunday") && (?S_ >= 1800)))
7         >
8         (((?D = "Saturday" || ?D = "Sunday") && ?S < 1800) ||
9         (!(?D = "Saturday" || ?D = "Sunday") && (?S >= 1800))))}
10  FILTER ( !bound(?S_))}
```

5 Summary and Discussion

In this report we argued for (re-)considering preferences in SPARQL queries. Given the vast amount of data being subjected to SPARQL queries, we can conceive many examples where complex preferences would be needed to find the "needle in the haystack", with the user specifying preferences not only over query results but also over the sources (SPARQL endpoints) and provenance of data used to produce those results. Here we proposed a core grammar for PrefSPARQL, an extension to SPARQL 1.1 that supports the expression of preferred query results. Our language builds on established work on SQL preferences and on an earlier effort by Siberski et al. (Siberski et al., 2006), extending it with conditional preferences. We further argued, contrary to the conjecture of Siberski et al., that these preference queries can be directly expressed in both SPARQL1.0 and SPARQL1.1 using OPTIONAL queries or novel features of SPARQL1.1, such as NOT EXISTS. We illustrated such a realization with respect to skyline and conditional preference queries. We acknowledge that, at the time Siberski and colleagues' work was performed, the semantics of SPARQL1.0 was not yet fully defined and SPARQL1.1 was still far off on the horizon.

Nevertheless, we argue that this topic needs further attention, since preference queries implemented naively by rewriting in SPARQL might become prohibitively costly. In particular, we emphasize that all the examples we gave in this paper, even those expressing simple preferences by "stacking OPTIONALs", as mentioned in Section 1, or, respectively all our example translations would produce so called non-well-designed patterns (Pérez, Arenas, & Gutierrez, 2009b; Letelier, Pérez, Pichler, & Skritek, 2012) in SPARQL. We therefore plan to further investigate relaxations of the well-designedness restriction, which still enable efficiently evaluable preference queries.

The specification and efficient realization of preference-based SPARQL queries is an important topic that is worthy of further exploration. As this work is ongoing, considerations for expanding it include the addition of quantitative preferences in the form of top-k queries, ranking within a skyline, preferences over endpoints in the context of the SPARQL1.1 Federation extension (Prud'hommeaux & Buil-Aranda, 2013), SPARQL endpoint discovery (e.g. by preferences over Service descriptions (Williams, 2013)) as well as interaction of preferences with Entailment Regimes (Glimm & Ogbuji, 2013).

Acknowledgements

We wish to thank Wolf Siberski, Jeff Pan, and Florian Wenzel for their assistance. This work has been partially supported by the Natural Sciences and Engineering Research Council of Canada (NSERC), an

Ontario Graduate Scholarship (OGS), and by the Vienna Science and Technology Fund (WWTF) through project ICT12-015.

References

Börzsönyi, S., Kossmann, D., & Stocker, K. (2001). The skyline operator. In *Proc. of the 17th Int'l Conference on Data Engineering (ICDE)*, pp. 421–430.

Bozzon, A., Valle, E. D., & Magliacane, S. (2012). Extending SPARQL algebra to support efficient evaluation of top-k SPARQL queries. In Ceri, S., & Brambilla, M. (Eds.), *Search Computing – Broadening Web Search*, pp. 143–156. Springer Lecture Notes in Computer Science.

Chomicki, J. (2002). Querying with intrinsic preferences. In *Proc. of the 8th Int'l Conference on Extending Database Technology (EDBT)*, pp. 34–51.

Chomicki, J. (2003). Preference formulas in relational queries. *ACM Trans. on Database Systems (TODS)*, *28*(4), 427–466.

Chomicki, J. (2011). Logical foundations of preference queries. *IEEE Data Engineering Bulletin*, *34*(2), 3–10.

Glimm, B., & Ogbuji, C. (2013). SPARQL 1.1 Entailment Regimes.. W3C Recommendation.

Harris, S., & Seaborne, A. (2013). SPARQL 1.1 Query Language.. W3C Recommendation.

Ilyas, I. F., Beskales, G., & Soliman, M. A. (2008). A survey of top-k query processing techniques in relational database systems. *ACM Computing Surveys*, *40*(4), 11:1–11:58.

Kießling, W. (2002). Foundations of preferences in database systems. In *Proc. of 28th Int'l Conference on Very Large Data Bases (VLDB)*, pp. 311–322.

Kießling, W., Endres, M., & Wenzel, F. (2011). The preference SQL system - an overview. *IEEE Data Engineering Bulletin*, *34*(2), 11–18.

Kießling, W., & Köstler, G. (2002). Preference SQL - design, implementation, experiences. In *Proc. of 28th Int'l Conference on Very Large Data Bases (VLDB)*, pp. 990–1001.

Letelier, A., Pérez, J., Pichler, R., & Skritek, S. (2012). Static analysis and optimization of semantic web queries. In *Proc. of the 31st Symposium on Principles of Database Systems (PODS)*, pp. 89–100.

Magliacane, S., Bozzon, A., & Valle, E. D. (2012). Efficient execution of top-k SPARQL queries. In *Proc. of the 11th Int'l Semantic Web Conference (ISWC)*, pp. 344–360.

Montoya, G., Vidal, M.-E., Corcho, Ó., Ruckhaus, E., & Aranda, C. B. (2012). Benchmarking federated SPARQL query engines: Are existing testbeds enough?. In *Proc. of the 11th Int'l Semantic Web Conference (ISWC)*, pp. 313–324.

Pérez, J., Arenas, M., & Gutierrez, C. (2009a). Semantics and complexity of SPARQL. *ACM Trans. on Database Systems (TODS)*, *34*(3), Article 16 (45 pages).

Pérez, J., Arenas, M., & Gutierrez, C. (2009b). Semantics and complexity of SPARQL. *ACM Trans. on Database Systems (TODS)*, *34*(3).

Polleres, A., & Wallner, J. (2013). On the relation between sparql1.1 and answer set programming. *Journal of Applied Non-Classical Logics (JANCL)*, *23*(1–2), 159–212. Special issue on Equilibrium Logic and Answer Set Programming.

Prud'hommeaux, E., & Buil-Aranda, C. (2013). SPARQL 1.1 Federated Query.. W3C Recommendation.

Prud'hommeaux, E., & Seaborne, A. (2008). SPARQL Query Language for RDF.. W3C Recommendation.

Siberski, W., Pan, J. Z., & Thaden, U. (2006). Querying the semantic web with preferences. In *International Semantic Web Conference*, pp. 612–624.

Stefanidis, K., Koutrika, G., & Pitoura, E. (2011). A survey on representation, composition and application of preferences in database systems. *ACM Trans. on Database Systems (TODS)*, *36*(3), 19.

Wagner, A., Tran, D. T., Ladwig, G., Harth, A., & Studer, R. (2012). Top-k linked data query processing. In *Proceedings of the 9th Extended Semantic Web Conference (ESWC)*, pp. 56–71.

Williams, G. (2013). SPARQL 1.1 Service Description.. W3C Recommendation.

A Appendix

Returning to the example from Section 4:

```
SELECT ?A
WHERE {
   ?T :rated ?R; :offers ?A. ?A :starts ?S; :ends ?E .
   PREFERRING ( (?R = excellent) AND
                ((?S >= 1800) || ?E <= 1600) PRIOR TO
                HIGHEST ?S ))}
```

Without the use of BIND...AS statements, the above example from Section 4 translates into the following SPARQL 1.1 query:

```
1 SELECT ?A WHERE { ?T :rated ?R; :offers ?A. ?A :starts ?S; :ends ?E .
2   FILTER NOT EXISTS {
3     ?T_ :rated ?R_; :offers ?A_. ?A_ :starts ?S_; :ends ?E_ .
4     FILTER(
5       ( ((?R_ = :excellent) > (?R = :excellent)) &&
6         !(((?S_ >= 1600 || ?E_ <= 1600) <  (?S >= 1800 || ?E <= 1600)) ||
7           (?S_ < ?S &&  (?S >= 1800 || ?E <= 1600) = (?S_ >= 1600 || ?E_ <= 1600))))
8       ||
9       ( !((?R_ = :excellent) < (?R = :excellent)) &&
10         (((?S_ >= 1600 || ?E_ <= 1600) >  (?S >= 1800 || ?E <= 1600)) ||
11          (?S_ > ?S &&  (?S >= 1800 || ?E <= 1600) = (?S_ >= 1600 || ?E_ <= 1600)))))}}
```