Information Sciences 198 (2012) 1–23

Contents lists available at SciVerse ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Top-k retrieval for ontology mediated access to relational databases

Umberto Straccia

Istituto di Scienza e Tecnologie dell'Informazione (ISTI), Consiglio Nazionale delle Ricerche (CNR), Pisa, Italy

ARTICLE INFO

Article history:
Received 5 February 2010
Received in revised form 11 October 2010
Accepted 18 February 2012
Available online 1 March 2012

Keywords:
Description logic
Top-k retrieval
Ontology

ABSTRACT

We address the problem of evaluating ranked top-k queries in the context of ontology mediated access over relational databases. An ontology layer is used to define the relevant abstract concepts and relations of the application domain, while facts with associated score are stored into a relational database. Queries are conjunctive queries with ranking aggregates and scoring functions. The results of a query may be ranked according to the score and the problem is to find efficiently the top-k ranked query answers.

1. Introduction

Description Logics (DLs) [4] provide popular features for the representation of structured knowledge. Nowadays, DLs have gained even more popularity due to their application in the context of the *Semantic Web*. DLs play a particular role as they are essentially the theoretical counterpart of state of the art languages to specify ontologies, such as *OWL 2*.[1]

It becomes also apparent that in these contexts, data are typically very large and dominate the intentional level of the ontologies. Hence, while in the above mentioned contexts one could still accept reasoning, specifically query answering, that is exponential on the intentional part, it is mandatory that reasoning and query answering is polynomial in the data size, *i.e.* in *data complexity* [35].

Recently, efficient management of large amounts of data and its computational complexity analysis has become a primary concern of research in DLs and in ontology reasoning systems [3,7,8,12,13,17,18,29]. This is especially important in case of ontology mediated access to relational databases for data intensive applications, where data is stored into a database and a data complexity tractable DL is used to define the relevant abstract concepts and relations of the application domain. Specifically, the adoption of such a DL allowed the development of an efficient query answering method that can roughly described as follows: given a query and an ontology, rewrite the query using the ontology as several queries that can then be submitted as SQL queries over the database. The effectiveness of this method is based on the fact that the rewriting of the query can be computed efficiently and SQL query answering may take advantage of the performance of state of the art database engines.

In this paper, we address a relatively novel issue for DLs with a huge data repository, namely the problem of *evaluating ranked top-k queries*. Usually, an answer to a query is a set of tuples that satisfy a query. Each tuple does or does not satisfy the predicates in the query. However, very often the information need of a user involves so-called *scoring predicates* [14]. For instance, a user may need: "Find *cheap* hotels *near* to the conference location". Here, *cheap* and *near* are scoring predicates. Unlike the classical case, tuples satisfy now these predicates to a score. In the former case the score may depend, *e.g.*, on the price, while in the latter case it may depend *e.g.*, on the distance between the hotel location and the conference location.

E-mail address: straccia@isti.cnr.it

[1] http://www.w3.org/TR/2009/REC-owl2-overview-20091027.

0020-0255/$ - see front matter © 2012 Elsevier Inc. All rights reserved.
doi:10.1016/j.ins.2012.02.026

Therefore, a major problem we have to face with in such cases is that now an answer is a set of tuples *ranked* according to their *score*. This poses a non-negligible challenge in case we have to deal with a huge amount of instances. Indeed, virtually every tuple may satisfy a query with a non-zero score and, thus, has to be ranked. Computing all these scores, ranking them and then selecting the top-k ones is not feasible in practice for large size databases [14].

So far, there are very views works addressing specifically the top-k retrieval problem for logic-based languages in general and, specifically, in the DL context [15,24,26,28,32]. With respect to these works, the contribution of this paper can be summarised as follows.

- We consider a more general DL language than the one considered in top-k retrieval, such as described in [15,24,26,28]. Specifically,
  - at the ontology level, while the language is closely related to *OWL QL* (a so-called profile of the web ontology language *OWL 2* [2]), we further allow scoring functions to occur in the left hand side of an axiom to combine scores and assigning the result to the axiom's right hand side;
  - at the query level, we consider Datalog like conjunctive queries with ranking aggregates and scoring functions with scoring predicates in the same way as [14]. Our query language will subsume [15,16,22,24,26–28].
- We provide a reasoning algorithm for the specified language. Specifically,
  - we provide a generalised reformulation algorithm in the spirit of [8], that given a query and an ontology, rewrites the query as several queries;
  - we then provide a novel algorithm for efficient submission of these queries to the database. To this end we extend the so-called *Disjunctive Threshold Algorithm* provided in [23–26] to cope with ranking aggregates and main memory problems encountered in preliminary experiments.
- We conduct an experiment to asses both the effective SoftFacts system [1,33], and to illustrate some issues related to a naive approach to the top-k retrieval problem.

In the following, we proceed as follows. In the next section, we provide some basics on top-k retrieval for relational databases. Then, Section 3 describes our query and representation language, Section 4 describes the query answering algorithms, while Section 5 reports the experiments. Section 6 addresses related work and Section 7 concludes and provides an outlook to future work.

2. Top-k retrieval in relational databases

We recap here some salient notions related to the top-k retrieval problem for relational databases (see, *e.g.* [14]).

There are essentially three query models to specify the data objects to be scored, namely (i) top-k selection query; (ii) top-k join query; and (iii) top-k aggregate query, which we describe next. For illustrative purposes, we consider the following running example taken from [30].

Example 2.1. Consider a relational database excerpt (Fig. 1) about student's Curricula Vitæ, together with some basic information about the degrees they got.

2.1. Top-k selection query

In this model, the scores are assumed to be attached to base tuples. A top-k selection query is required to report the k tuples with the highest scores according to some user-defined scoring function.

Profile								
profID	firstName	lastName	height	age	cityOfBirth	address	city	...
2	Wayne	Hernandez	180	31	Berlin	Via Volta	Terni	...
34	Hillary 156	Gadducci	175	28	Bangalore	Church ST	New York	...
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	....

HasDegree		
profID	degID	mark
2	29	107
34	25	104
.	.	.
.	.	.
.	.	.

Degree	
degID	name
29	Civil_Structural_Engineering
25	Chemical_Engineering
.	.
.	.
.	.

Fig. 1. A relational database.

Definition 2.1 (*Top-k Selection Query*) Consider a relation R, where each tuple in R has n attributes. Consider m scoring predicates, $p_1, \ldots, p_m$ defined on these attributes. Let $f(t) = f(p_1(t), \ldots, p_m(t))$ be the overall score of tuple $t \in R$. A *top-k selection query* selects the k tuples in R with the largest f values.

A SQL template for top-k selection query is the following:

```
SELECT *
FROM R
WHERE selection condition
ORDER BY f(p1,...,pm)
LIMIT k
```

For instance, consider Example 3.1. Assume we are interested in finding the top-10 most young and tall students born in Berlin, where young is defined in terms of the student's age as some scoring predicate p_1, while tall is defined in terms of the student's height by means of some scoring predicate p_2. The global score has been defined by a user as the linear combination $0.75 \cdot p_1(t.age) + 0.25 \cdot p_2(t.height)$, giving more weight to the student's age. This query could be written as follows:

```
SELECT profID, lastName
FROM Profiles
WHERE cityOfBirth = 'Berlin'
ORDER BY 0.75 · p1(age) + 0.25 · p2(height)
LIMIT 10
```

Note that concerning scoring predicates, typically we may use the so-called left shoulder, right shoulder, triangular and trapezoidal functions (see Fig. 2). For instance, may define $p_1(\text{age}) = ls(\text{age}; 20, 30)$ and $p_2(\text{height}) = rs(\text{height}; 175, 200)$.

2.2. Top-k join query

In this model, scores are assumed to be attached to join results rather than base tuples. A top-k join query joins a set of relations based on some arbitrary join condition, assigns scores to join results based on some scoring function, and reports the top-k join results.

Definition 2.2. Top-k Join QueryConsider a set of relations $R_1, \ldots, R_n$. A *top-k join query* joins $R_1, \ldots, R_n$, and returns the k join results with the largest combined scores. The combined score of each join result is computed according to some function $f(p_1, \ldots, p_m)$, where $p_1, \ldots, p_m$ are scoring predicates defined over the join results.

A possible SQL template for a top-k join query is.

```
SELECT *
FROM R1, ..., Rn
WHERE join_condition(R1,...,Rn)
ORDER BY f(p1,...,pm)
LIMIT k
```

For instance, consider Example 3.1. Suppose we are looking for the top-10 young and good students, where good is defined in terms of the marks by means of a scoring predicate p_3. We may formulate such a query as.

```
SELECT P.profID, P.lastName, P.age, D.mark
FROM Profiles P, HasDegree D
WHERE P.profID = H.profID
ORDER BY 0.5 · p1(P.age) + 0.5 · p3(D.mark)
LIMIT 10
```

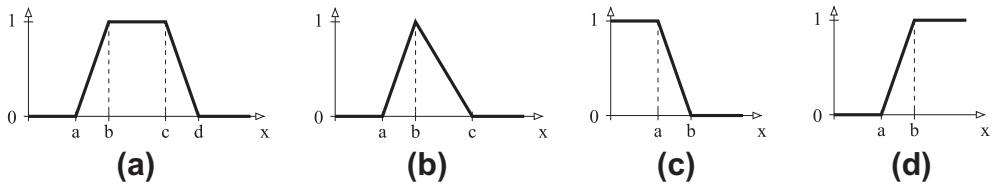

Fig. 2. (a) Trapezoidal function $trz(x; a, b, c, d)$, (b) triangular function $tri(x; a, b, c)$, (c) left shoulder function $ls(x; a, b)$, and (d) right shoulder function $rs(x; a, b)$.

2.3. Top-k aggregate query

In this model, scores are computed for tuple groups, rather than individual tuples. A top-k aggregate query reports the k groups with the largest scores. Group scores are computed using a group aggregate function such as sum, min and avg.

Definition 2.3. Top-k Aggregate QueryConsider a set of grouping attributes $G = \{g_1, \ldots, g_r\}$, and an aggregate function f that is evaluated on each group. A top-k aggregate query returns the k groups, based on G, with the highest f values.

A SQL formulation for a top-k aggregate query is.

```
SELECT g_1, ..., g_r, f AS score
FROM R_1, ..., R_n
WHERE join_condition(R_1,...,R_n)
GROUP BY g_1, ..., g_r
ORDER BY score
LIMIT k
```

An example top-k aggregate query is to score young and good students higher the more degrees they hold. Such a query may be formulated as.

```
SELECT P.profID, P.lastName, P.age, SUM(0.5 · p_1(P.age) + 0.5 · p_3(D.mark)) AS score
FROM Profiles P, HasDegree D
WHERE P.profID = H.profID
GROUP BY P.profID, P.lastName, P.age
ORDER BY score
LIMIT 10
```

which concludes this section.

3. Our ontology representation and query language

For computational reasons, the particular logic we adopt is based on a DLR-Lite [8] Description Logic (DL) [4] variant. DLR-Lite is an extension of DL-Lite, which is the logic behind *OWL QL* (a profile of the web ontology language *OWL 2* [2]), and supports n-ary relations whereas DLs support usual unary relations (called *concepts*) and binary relations (called *roles*) only.[2] Please note that we do not consider negation of atoms, as supported in DL-Lite, as these do not play any role at query answering time, the main task we are interested in. Negated atoms play a role only at knowledge base consistency checking time. In our setting, any knowledge base will be consistent. Therefore, to what concerns query answering, the drop of negation is harmless.

On one hand, our DL will be used in order to define the relevant abstract concepts and relations of the application domain. On the other hand, conjunctive queries will be used to describe the information needs of a user and rank the answers according to a scoring function and are the logical counterpart of top-k aggregate queries we have seen in the previous section.

Scoring space. To start with, answers to a query are scored according to some scoring function. We will call the score also degree. Hence we have to fix a scoring space. We will consider as scoring-space the interval of rational numbers $[0, \top]$, where $\top \geqslant 1$. One might wonder why not $\top = 1$, as usually assumed. This is due to the fact that we may use aggregation functions (*e.g.*, sum of scores) that may generate typically a score above 1.0. In principle, any such score may be mapped to $[0,1]$ via the function $h(x) = x/\top$, but for convenience we prefer to consider $[0, \top]$. What is important is not the score of an answer *per se*, but rather the ranking of answers induced by the scores.

3.1. The ontology representation language

Knowledge base. A *knowledge base* $\mathcal{K} = \langle \mathcal{F}, \mathcal{O}, \mathcal{M} \rangle$ consists of a *facts component* $\mathcal{F}$, an *Ontology component* $\mathcal{O}$ and an *mapping component* $\mathcal{M}$, which are defined below.

Facts component. We allow to store so-called graded ground facts such as "the Audi TT is to some degree (*e.g.*, 0.85) a sports car". Indeed, $\mathcal{F}$ is a finite set of expressions of the form

$$R(c_1, \ldots, c_n)[s],$$

where R is an n-ary relation, every c_i is a constant, and s is a degree in $[0, \top]$.

[2] Most *OWL QL* constructs are supported in our language. We refer the reader for more details on this to Appendix A.

For each relation R, we represent the facts $R(c_1,\ldots,c_n)[s]$ in $\mathcal{F}$ by means of a relational $n+1$-ary table T_R, containing the records $\langle c_1,\ldots,c_n,s\rangle$. We assume that there cannot be two records $\langle c_1,\ldots,c_n,s_1\rangle$ and $\langle c_1,\ldots,c_n,s_2\rangle$ in T_R with $s_1 \neq s_2$ (if there are, then we remove the one with the lower degree). Each table is sorted in descending order with respect to the degrees. For ease, we may omit the degree component and in such case the value $\top$ is assumed.

Ontology Component. The ontology component is used to define the relevant abstract concepts and relations of the application domain by means of axioms. Specifically, $\mathcal{O}$ is a finite set of *axioms* having the form

$$t(Rl_1,\ldots,Rl_m) \sqsubseteq Rr,$$

where

1. $m \geqslant 1$, all Rl_i and Rr have the same arity and where each Rl_i is a so-called *left hand relation* and Rr is a *right hand relation*;
2. t is a *scoring* function symbol that has a fixed interpretation $\bar{t} : [0,\top]^m \to [0,\top]$. $\bar{t}$ will combine the scores of the left hand relations into an overall *score* to be assigned to the right hand relation. We assume that $\bar{t}$ is *monotone*, that is, for each $\mathbf{v}$, $\mathbf{v}' \in [0,\top]^m$ such that $\mathbf{v} \leqslant \mathbf{v}'$, $\bar{t}(\mathbf{v}) \leqslant \bar{t}(\mathbf{v}')$ holds, where $(v_1,\ldots,v_m) \leqslant (v'_1,\ldots,v'_m)$ iff $v_i \leqslant v'_i$ for all i. We also assume that $\bar{t}$ is both *bounded*, *i.e.*, $\bar{t}(\ldots,v_i,\ldots) \leqslant v_i$ and computable, *i.e.* for any input, the value of $\bar{t}$ can be determined in finite time. As t will have fixed interpretation, for the sake of ease the presentation, we will use in the following $\bar{t}$ in place of t instead.

We also write

$$Rl_1 \sqcap \ldots \sqcap Rl_m \sqsubseteq Rr$$

in place of the axiom

$$\min(Rl_1,\ldots,Rl_m) \sqsubseteq Rr.$$

Axioms of the form $Rl_1 \sqcap \ldots \sqcap Rl_m \sqsubseteq Rr$ are those considered usually in crisp DL-Lite and DRL-Lite variants and, thus, are special cases (see Appendix A). Furthermore, we assume that relations occurring in $\mathcal{F}$ do not occur in axioms (so, we do not allow that database relation names occur in $\mathcal{O}$).

Example 3.1 Example 2.1 cont.). An excerpt of the domain ontology is described in Fig. 3 and partially encodes an ontology used to describe Curricula Vitæ. We assume that we have relations hasDegree (*profID*, *degID*, *marks*), knowsLanguage (*profID*, *lanID*) and an atomic concept (unary relation) Degree (*degID*). For instance, axiom (Fig. 3) states that the languages known by profile profID should be languages. Here, knowsLanguage[2] corresponds to the projection of relation knowsLanguage on the second column.

The exact syntax of the relations appearing on the left-hand and right hand side of ontology axioms is specified below (where $h \geqslant 1$):

$$Rr \to A | R[i_1,\ldots,i_k]$$
$$Rl \to A | R[i_1,\ldots,i_k] | R[i_1,\ldots,i_k].(Cond_1,\ldots,Cond_h)$$
$$Cond \to ([i] \leqslant v) | ([i] < v) | ([i] \geqslant v) | ([i] > v) | ([i] = v) | ([i] \neq v)$$

where A is an *atomic concept* (an unary predicate), R is an n-ary relation with $n \geqslant 2$, $1 \leqslant i_1, i_2, \ldots, i_k \leqslant n$, $1 \leqslant i \leqslant n$ and v is a string value or a rational number. Here $R[i_1,\ldots,i_k]$ is the projection of the relation R on the columns $i_1,\ldots,i_k$ (the order of the indexes matters). Hence, $R[i_1,\ldots,i_k]$ has arity k. On the other hand, $R[i_1,\ldots,i_k].(Cond_1,\ldots,Cond_l)$ further restricts the projection $R[i_1,\ldots,i_k]$ according to the conditions specified in $Cond_i$. For instance, $([i] \leqslant v)$ specifies that the values of the i-th column have to be less or equal than the value v and similarly, for the other conditions. We assume that the comparison occurs among values with a comparable type. For instance, Profile[1,5].(([5] $\geqslant$ 28)) corresponds to the set of tuples $\langle$profID,birthDate$\rangle$ such that the fifth column of the relation Profile, *i.e.* the person's age, is equal or greater than 28.

Mapping component. The mapping component is a set of "mapping statements" that allow to connect atomic concepts and relations to physical relational tables. Essentially, this component is used as a wrapper to the underlying database and, thus, prevents that relational table names occur in the ontology. Formally, a *mapping statement* is of the form

$$R \mapsto (c_1,\ldots,c_n)[c_{score}].sql,$$

where *sql* is a SQL statement returning n-ary tuples $\langle c_1,\ldots,c_n\rangle$ with score determined by the c_{score} column. The tuples have to be ranked in decreasing order of score and, as for the fact component, we assume that there cannot be two records $\langle \mathbf{c}, s_1\rangle$ and $\langle \mathbf{c}, s_2\rangle$ in the result set of *sql* with $s_1 \neq s_2$ (if there are, then we remove the one with the lower score). The score c_{score} may be

$$\text{Legislators} \sqsubseteq \text{Legal\_Support\_Workers} \quad (1)$$
$$\text{Auditors} \sqsubseteq \text{Accountants\_and\_Auditors} \quad (2)$$
$$\text{Bakers} \sqsubseteq \text{Food\_Processing\_Workers} \quad (3)$$
$$\text{knowsLanguage}[2] \sqsubseteq \text{Language} \quad (4)$$
$$\text{hasDegree}[2] \sqsubseteq \text{Degree} \quad (5)$$

Fig. 3. Excerpt of a CV ontology.

omitted and in that case the score $\top$ is assumed for the tuples. We assume that R occurs in $\mathcal{O}$, while all of the relational tables occurring in the SQL statement occur in $\mathcal{F}$.

As illustrative purpose, assume that we have a relation Jobs in a database with signature Jobs(jobID, name). Then, an example of mapping statement is

$$\mathsf{jobName} \mapsto (\mathit{jobID}, \mathit{name}).(\mathsf{SELECT}\ \mathit{jobID}, \mathit{name}\ \mathsf{FROM}\ \mathit{Jobs}),$$

by means of which we state that the relation jobName occurring in the ontology component, has arity two and has to be mapped into the relation Jobs occurring in the database. On the other hand, the mapping statement

$$\mathsf{BigCity} \mapsto (\mathit{id})[\mathit{score}].(\mathsf{SELECT}\ \mathit{id}, \mathit{ls}(\mathit{size}; 5 \times 10^5, 10^6)\ \mathsf{AS}\ \mathit{score}\quad \mathsf{FROM}\ \mathit{CityTable}\ \mathsf{ORDER\ BY}\ \mathit{score})$$

defines the abstract relation BigCity as the set of city IDs, where the score of being big is determined by left shoulder function $ls(size; 5 \times 10^5, 10^6)$, where *size* is an attribute of the *CityTable* relation recording the number of inhabitants of a city. The city IDs are ranked in decreasing order of score.

Finally, we assume that there is at most one abstract statement for each abstract relational symbol R.

3.2. The query language

Concerning queries, a *query* consists of a conjunctive query with a scoring function to rank the answers. A conjunctive query will be the logical counterpart of SQL top-k aggregate queries we have seen in Section 2.3.

Top-k conjunctive query. Let $@ \in \{\mathsf{SUM}, \mathsf{AVG}, \mathsf{MAX}, \mathsf{MIN}\}$ be ranking aggregate function. Then a *top-k conjunctive aggregate query*, or simply, *conjunctive query*, is of the form

$$\begin{aligned} q(\mathbf{x})[s] \leftarrow\ & \exists \mathbf{y}\ R_1(\mathbf{z}_1)[s_1], \ldots, R_l(\mathbf{z}_l)[s_l], \\ & \mathsf{GroupedBy}(\mathbf{w}), \mathsf{OrderBy}\big(s = @\big[f(s_1, \ldots, s_l, p_1)(\mathbf{z}'_1), \ldots, p_h(\mathbf{z}'_h)\big]\big) \end{aligned}$$

where

1. q is an n-ary relation, every R_i is an n_i-ary relation. $R_i(\mathbf{z_i})$ may also be a concrete unary predicate of the form $(z \leqslant v)$, $(z < v)$, $(z \geqslant v)$, $(z > v)$, $(z = v)$, $(z \neq v)$, where z is a variable, v is a value of the appropriate type;
2. $\mathbf{x}$ are the *distinguished variables*;
3. $\mathbf{y}$ are existentially quantified variables called the *non-distinguished variables*. We omit to write $\exists \mathbf{y}$ when $\mathbf{y}$ is clear from the context;
4. $\mathbf{z_i}$, $\mathbf{z'_j}$ are tuples of constants or variables in $\mathbf{x}$ or $\mathbf{y}$;
5. $\mathbf{w}$ is a list of variables according to which we want to group the tuples. We assume that $\mathbf{w}$ are variables in $\mathbf{x}$ or $\mathbf{y}$ such that each variable in $\mathbf{x}$ occurs in $\mathbf{w}$;
6. s, s_1, …, s_l are distinct variables and different from those in $\mathbf{x}$ and $\mathbf{y}$;
7. p_j is an n_j-ary *scoring predicate* assigning to each n_j-ary tuple $\mathbf{c}_j$ a *score* $p_j(\mathbf{c}_j) \in [0, \top]$. We require that an n-ary scoring predicate p is *safe*, that is, there is not an m-ary scoring predicate p' such that $m < n$ and $p = p'$. Informally, all parameters are needed in the definition of p;
8. f is a *scoring* function f: $([0, \top])^{l+h} \to [0, \top]$, which combines the scores of the l relations $R_i(\mathbf{c}'_i)$ and the n scoring predicates $p_j\left(\mathbf{c}''_j\right)$ into an overall *score*. We assume that f is *monotone*, that is, for each $\mathbf{v}$, $\mathbf{v}' \in ([0, \top])^{l+h}$ such that $\mathbf{v} \leqslant \mathbf{v}'$, it holds $f(\mathbf{v}) \leqslant f(\mathbf{v}')$, where $(v_1, \ldots, v_{l+h}) \leqslant (v'_1, \ldots, v'_{l+h})$ iff $v_i \leqslant v'_i$ for all i. We also assume that the computational cost of f and all scoring predicates p_i is bounded by a constant.

We call $q(\mathbf{x})[s]$ its *head*, $\exists \mathbf{y}.R_1(\mathbf{z}_1)[s_1], \ldots, R_l(\mathbf{z}_l)[s_l]$ its *body*, $\mathsf{OrderBy}(s = @[f(s_1, \ldots, s_l, p_1)(\mathbf{z}'_1), \ldots, p_h(\mathbf{z}'_h)])$ the *scoring atom* and $\mathsf{GroupBy}(\mathbf{w})$ the *grouping atom*. We also allow the scores $[s], [s_1], \ldots, [s_l]$ and the scoring and grouping atom to be omitted. In this case we assume the value $\top$ for s_i and s instead.

The informal meaning of such a query is: if $\mathbf{z}_i$ is an instance of R_i to degree at least or equal to s_i, then $\mathbf{x}$ is an instance of q to degree at least or equal to s, where s has been determined by the scoring atom and the tuples are grouped according to $\mathbf{w}$.

Below, few example queries:

$\mathsf{q(x)} \leftarrow \mathsf{SportsCar(x)}$
// find sports cars
$\mathsf{q(x)} \leftarrow \mathsf{SportsCar(x)}, \mathsf{hasSpeed(x,y)}, (\mathsf{y} \geqslant 240)$
// find sports cars whose speed exceed 240
$\mathsf{q(x)[s]} \leftarrow \mathsf{SportsCar(x)[s_1]}, \mathsf{hasSpeed(x,y)}, \mathsf{isFast(y)[s_2]}, \mathsf{OrderBy(s = s_1 \cdot s_2)}$
// find fast sports cars
$\mathsf{q(x)[s]} \leftarrow \mathsf{SportsCar(x)[s_1]}, \mathsf{hasPrice(x,p)}, \mathsf{OrderBy(s = 0.7 \cdot s_1 + 0.3 \cdot ls(p; 10000, 14000))}$
// find cheap sports cars

Example 3.2 Example 3.1 cont.. Assume that we have also the abstract mappings

hasName $\mapsto$ (*profID, lastName*). (SELECT *profID, lastName* FROM *Profile*),
hasDegree $\mapsto$ (*profID, degID*). (SELECT *profID, degID* FROM *HasDegree*)
hasMark $\mapsto$ (*profID, mark*). (SELECT *profID, mark* FROM *HasDegree*)
hasDegreeName $\mapsto$ (*degID, name*). (SELECT *degID, name* FROM *Degree*).

Then, a query searching for CV's with a degree with mark between 100 (minimum) and 110 (maximum) can be expressed as

$$\begin{aligned} \mathsf{q}(\mathsf{id},\mathsf{name},\mathsf{degree},\mathsf{mark})[\mathsf{s}] \leftarrow\ & \mathsf{CV}(\mathsf{id}), \mathsf{hasName}(\mathsf{id},\mathsf{name}), \mathsf{hasDegree}(\mathsf{id},\mathsf{y}), \\ & \mathsf{hasDegreeName}(\mathsf{y},\mathsf{degree}), \mathsf{hasMark}(\mathsf{id},\mathsf{mark}), \\ & \mathsf{OrderBy}(\mathsf{s} = \mathsf{rs}(\mathsf{mark};100,110)) \end{aligned}$$

Assume now that we additionally would like to sum-up the scores of the degrees of each person. Then, such a query may be expressed as

$$\begin{aligned} \mathsf{q}(\mathsf{id},\mathsf{name})[\mathsf{s}] \leftarrow\ & \mathsf{CV}(\mathsf{id}), \mathsf{hasName}(\mathsf{id},\mathsf{name}), \mathsf{hasMark}(\mathsf{id},\mathsf{mark}), \\ & \mathsf{GroupedBy}(\mathsf{id},\mathsf{name}), \mathsf{OrderBy}(\mathsf{s} = \mathsf{SUM}[\mathsf{rs}(\mathsf{mark};100,110)]) \end{aligned}$$

Intuitively, for the above query, we ask to group all tuples according to $\langle \mathsf{id} \rangle$, name and then for each group to sum-up the scores. That is, if $g = \{t_1,\ldots,t_n\}$ is a group of tuples with same id and name, where each tuple has score s_i computed as rs (mark; 100, 110) then the score s_g of the group g is $\sum_{t_i} s_i$. A group g is ranked then according to its score s_g and the top-k ranked groups are returned.

Top-k disjunctive query. We conclude by defining a *disjunctive query* **q** as usual as a finite set of conjunctive queries in which all the rules have the same head. Intuitively, the answers to a disjunctive query are the *union* of the answers of the conjunctive queries.

3.3. Semantics

Interpretation. An *interpretation* $\mathcal{I} = \langle \Delta, \cdot^{\mathcal{I}} \rangle$ consists of a *fixed infinite domain* Δ containing a domain Δ_D of strings and rational numbers, and an *interpretation function* $\cdot^{\mathcal{I}}$ that maps.

- every atom A to a partial function $A^{\mathcal{I}} : \Delta \to [0, \top]$
- maps an n-ary predicate R to a partial function $R^{\mathcal{I}} : \Delta^n \to [0, \top]$
- constants to elements of Δ such that $a^{\mathcal{I}} \neq b^{\mathcal{I}}$ if $a \neq b$ (unique name assumption).

We also assume to have one object for each constant, denoting exactly that object. In other words, we have standard names, and we do not distinguish between the alphabet of constants and the objects in Δ.

Furthermore, we assume that the relations have a typed signature and the interpretations have to agree on the relation's type. To the easy of presentation, we omit the formalisation of this aspect and leave it at the intuitive level.

Informally, for a relation R and tuple $\mathbf{c}$, an interpretation assigns a score to $R(\mathbf{c})$ (similarly for $A^{\mathcal{I}}(c)$).

Note that, since $R^{\mathcal{I}}$ (resp. $A^{\mathcal{I}}$) may be a partial function, some tuples may not have a score. Alternatively, we may assume $R^{\mathcal{I}}$ (resp. $A^{\mathcal{I}}$) to be a total function. We use the former formulation to distinguish the case where a tuple $\mathbf{c}$ may be retrieved, even though the score is 0, from the case where a tuple is not retrieved, since it does not satisfy the query. In particular, if a tuple does not belong to an extensional relation, then its score is assumed to be undefined, while if $R^{\mathcal{I}}$ (resp. $A^{\mathcal{I}}$) is total, then the score of this tuple would be 0.

In the following, we use $\mathbf{c}$ to denote an n-tuple of constants, and $\mathbf{c}_{|i_1,\ldots,i_k}$ to denote the $i_1, \ldots, i_k$-th components of $\mathbf{c}$. For instance, $(a,b,c,d)_{|3,1,4}$ is (c,a,d). Please note that the order matters.

Models of a knowledge base. Concerning facts, an interpretation $\mathcal{I}$ is a *model* of (or *satisfies*) a fact $R(\mathbf{c})[s]$, denoted $\mathcal{I} \models R(\mathbf{c})[s]$, iff $R^{\mathcal{I}}(\mathbf{c}) \geqslant s$ whenever $R^{\mathcal{I}}(\mathbf{c})$ is defined. Furthermore, an interpretation $\mathcal{I}$ *is a model of* (*satisfies*) a fact component $\mathcal{F}$ iff it satisfies each element in it.

Concerning comparison predicates, $\mathcal{I}$ is a *model* of (or *satisfies*) $([i] \leqslant v)(\mathbf{c})$, denoted $\mathcal{I} \models ([i] \leqslant v)(\mathbf{c})$, iff $c_i \leqslant v$, *i.e.*, the projection of tuple $\mathbf{c}$ on the i-th column is less or equal than the value v and similarly for the other comparison constructs, $([i] < v)$, $([i] \geqslant v)$, $([i] > v)$, $([i] = v)$ and $([i] \neq v)$. We assume in all of them that both the values v and c_i are of the appropriate type.

The interpretation function $\cdot^{\mathcal{I}}$ has to satisfy the following conditions: for all $\mathbf{c} \in \Delta^k$ and n-ary relation R:

$$\begin{aligned} R[i_1,\ldots,i_k]^{\mathcal{I}}(\mathbf{c}) &= \sup\left\{R^{\mathcal{I}}(\mathbf{c}') | \mathbf{c}' \in \Delta^n, \mathbf{c}'_{|i_1,\ldots,i_k} = \mathbf{c}\right\}(R[i_1,\ldots,i_k].(Cond_1,\ldots,Cond_l))^{\mathcal{I}}(\mathbf{c}) \\ &= \sup\left\{R^{\mathcal{I}}(\mathbf{c}') | \mathbf{c}' \in \Delta^n, \mathbf{c}'_{|i_1,\ldots,i_k} = \mathbf{c}, \mathcal{I} \models Cond_j(\mathbf{c}')\right\}. \end{aligned}$$

Some explanations are in place. Consider $R[i_1,\ldots,i_k]$. Informally, from a classical semantics point of view, $R[i_1,\ldots,i_k]$ is the projection of the relation R over the columns i_1, …, i_k and, thus, corresponds to the set of tuples

$$\{\mathbf{c}|\exists \mathbf{c}' \in R \text{ s.t. } \mathbf{c}'_{|i_1,\ldots,i_k} = \mathbf{c}\}.$$

Note that for a fixed tuple $\mathbf{c}$ there may be several tuples $\mathbf{c}' \in R$ such that $\mathbf{c}'_{|i_1,\ldots,i_k} = \mathbf{c}$. As now any such $\mathbf{c}'$ may have an attached score, we choose the maximal among these scores.

Now given an interpretation $\mathcal{I}$, the notion of $\mathcal{I}$ *is a model of* (*satisfies*) an axiom τ, denoted $\mathcal{I} \models \tau$, is defined as follows:

$$\mathcal{I} \models t(Rl_1,\ldots,Rl_m) \sqsubseteq Rr \text{ iff for all } \mathbf{c} \in \Delta^k, t\left(Rl_1^{\mathcal{I}}(\mathbf{c}),\ldots,Rl_l^{\mathcal{I}}(\mathbf{c})\right) \leqslant Rr^{\mathcal{I}}(\mathbf{c}),$$

where we assume that the arity of Rr and all Rl_i is k. An interpretation $\mathcal{I}$ *is a model of* (*satisfies*) an ontology $\mathcal{O}$ iff it satisfies each element in it.

Concerning mapping statements, the notion of $\mathcal{I}$ *is a model of* (*satisfies*) a mapping statement σ, denoted $\mathcal{I} \models \sigma$, is defined as follows: let *sql* be a SQL statement and let use denote $sql^{\mathcal{I}} = \{\langle \mathbf{c}, s\rangle | (\mathbf{c}, s)$ is an answer to the SQL statement *sql*$\}$, then

$$\mathcal{I} \models R \mapsto (t_1,\ldots,t_n)[c_{score}].sql \text{ iff for all } \mathbf{c} \in \Delta^n, R^{\mathcal{I}}(\mathbf{c}) \geqslant s \text{ if } \langle \mathbf{c}, s\rangle \in sql^{\mathcal{I}}.$$

An interpretation $\mathcal{I}$ *is a model of* (*satisfies*) a mapping component $\mathcal{M}$ iff it satisfies each element in it and $\mathcal{I}$ *is a model of* (*satisfies*) a knowledge base if it satisfies each component.

Models of a query. Concerning queries, we may assume that they are of the form

$$q(\mathbf{x})[s] \leftarrow \exists \mathbf{y}\phi(\mathbf{x},\mathbf{y})[\mathbf{s}], \tag{6}$$

where $\phi(\mathbf{x},\mathbf{y})[\mathbf{s}]$ is

$$R_1(\mathbf{z}_1)[s_1],\ldots,R_l(\mathbf{z}_l)[s_l], \quad \mathsf{GroupBy}(\mathbf{w}), \mathsf{OrderBy}\left(s = @\left[f(s_1,\ldots,s_l,p_1)\left(\mathbf{z}'_1\right),\ldots,p_h\left(\mathbf{z}'_h\right)\right]\right).$$

Let R be an n-ary relation, $\mathbf{c}$ an n-ary tuple and s be a score. Then an interpretation $\mathcal{I}$ *is a model of* (*satisfies*) $R(\mathbf{c})[s]$, denoted $\mathcal{I} \models R(\mathbf{c})[s]$, iff $R^{\mathcal{I}}(\mathbf{c}) \geqslant s$. Note that if $R^{\mathcal{I}}(\mathbf{c})$ is not defined then $R(\mathbf{c})[n]$ is not satisfied.

We say that $\mathcal{I}$ *is a model of* (*satisfies*) a query of the form (6) iff for all $\mathbf{c} \in \Delta^n$, $q^{\mathcal{I}}(\mathbf{c}) \geqslant s$ whenever

$$s = \sup\{s' | g = \{\langle \mathbf{c}, \mathbf{c}'_1\rangle,\ldots,\langle \mathbf{c}, \mathbf{c}'_k\rangle\},$$

g is a group of k tuples with identical projection on the variables in $\mathbf{w}$,

$$s' = @[s^1,\ldots,s^k]\},$$

where for $1 \leqslant r \leqslant k$, each score s^r is computed as follows:

$\mathbf{c}'_r \in \Delta \times \cdots \times \Delta$ is a substitution of variables in $\mathbf{y}$,

$\mathcal{I} \models R_i(\mathbf{c}_i)[s_i^r], \mathbf{c}_i$ projection of $\langle \mathbf{c}, \mathbf{c}'_r\rangle$ on the variables $\mathbf{z}_i$,
$v_j^r = p_j(\bar{\mathbf{c}}_j)$, where $\bar{\mathbf{c}}_j$ is the projection of $\langle \mathbf{c}, \mathbf{c}'_r\rangle$ on the variables $\mathbf{z}'_j$,
$s^r = f(s_1^r,\ldots,s_l^r, v_1^r,\ldots,v_h^r)\}$.

In the above expression, for each group of tuples, $g = \{\langle \mathbf{c}, \mathbf{c}'_1\rangle,\ldots,\langle \mathbf{c}, \mathbf{c}'_k\rangle\}$, we compute the scores s^r each tuple belonging to that group and then apply the ranking aggregation function.

Entailment. We say $\mathcal{K}$ *entails* $q(\mathbf{c})$ to degree s, denoted $\mathcal{K} \models q(\mathbf{c})[s]$, iff for each model $\mathcal{I}$ of $\mathcal{K}$, it is true that $q^{\mathcal{I}}(\mathbf{c}) \geqslant s$ whenever $q^{\mathcal{I}}(\mathbf{c})$ is defined. We implicitly assume that the score s is as large as possible, *i.e.*, there is no $s' > s$ such that $q^{\mathcal{I}}(\mathbf{c}) \geqslant s'$. For a disjunctive query $\mathbf{q} = \{q_1,\ldots,q_m\}$, $\mathcal{K}$ *entails* $\mathbf{q}(\mathbf{c})$ to degree s, denoted $\mathcal{K} \models \mathbf{q}(\mathbf{c})[s]$, iff $\mathcal{K} \models q_i(\mathbf{c})[s]$ for some $q_i \in \mathbf{q}$,

Top-k answer set. The *answer set* of $\mathbf{q}$ w.r.t. $\mathcal{K}$ is

$$ans(\mathcal{K},\mathbf{q}) = \{\langle \mathbf{c}, s\rangle | \mathcal{K} \models \mathbf{q}(\mathbf{c})[s]\}.$$

As now each answer to a query has a score, the basic inference problem we are interest in is the top-k retrieval problem, formulated as follows. Given a knowledge base $\mathcal{K}$, and a disjunctive query $\mathbf{q}$, retrieve k tuples $\langle \mathbf{c}, s\rangle$ that instantiate the query relation q with maximal scores (if k such tuples exist), and rank them in decreasing order relative to the score s, denoted

$$ans_k(\mathcal{K}, \mathtt{q}) = \mathtt{Top}_k\ ans(\mathcal{K}, \mathtt{q}).$$

The following example, taken from [26], will be used to illustrate our top-k query answering procedure.

Example 3.3. Suppose the set of axioms is

$$\mathcal{O} = \{P_2[2] \sqsubseteq A,\ A \sqsubseteq P_1[1],\ B \sqsubseteq P_2[1]\}.$$

Let us assume that we have the mapping statements

$P_2 \mapsto (c,s).(\text{SELECT } c,s \text{ FROM } TabP2)$
$B \mapsto (c).(\text{SELECT } c \text{ FROM } TabB)$
$C \mapsto (c).(\text{SELECT } c \text{ FROM } TabC)$

and that the set of facts $\mathcal{F}$ is

$$TabP2 = \{\langle 0, \mathtt{s}\rangle, \langle 3, \mathtt{t}\rangle, \langle 4, \mathtt{q}\rangle, \langle 6, \mathtt{q}\rangle\},$$
$$TabB = \{\langle 1\rangle, \langle 2\rangle, \langle 5\rangle, \langle 7\rangle\},$$
$$TabC = \{\langle 5\rangle, \langle 3\rangle, \langle 2\rangle, \langle 4\rangle\}.$$

Assume our disjunctive query is $\mathbf{q} = \{q', q''\}$ where

$$q' := q(x)[s] \leftarrow \exists y \exists z.P_2(x,y), P_1(y,z), \mathsf{OrderBy}(s = f(\mathtt{p}(x)))$$
$$q'' := q(x)[s] \leftarrow C(x), \mathsf{OrderBy}(s = f(\mathtt{r}(x))),$$

the scoring function f is the identity $f(z) = z$ (f is monotone, of course), the scoring predicate $\mathtt{p}$ is $\mathtt{p}(x) = \max(0, 1 - x/10)$, and the scoring predicate $\mathtt{r}$ is $\mathtt{r}(x) = \max(0, 1 - (x/5)^2)$. Therefore, we can rewrite the query $\mathbf{q}$ as

$$q' := q(x)[s] \leftarrow \exists y \exists z.P_2(x,y), P_1(y,z), \mathsf{OrderBy}(s = \max(0, 1 - x/10))$$
$$q'' := q(x)[s] \leftarrow C(x), \mathsf{OrderBy}(s = \max(0, 1 - (x/5)^2)).$$

Now, it can be verified that

$$\mathcal{K} \models q(3)[0.7], \ \mathcal{K} \models q(2)[0.84]$$

and for any $v \in [0, \top]$, $\mathcal{K} \not\models q(9)[v]$. In the former case, any model $\mathcal{I}$ of $\mathcal{K}$ satisfies $P_2(3,\mathtt{t})$. But, $\mathcal{I}$ satisfies $\mathcal{O}$, so $\mathcal{I}$ satisfies $P_2[2] \sqsubseteq P_1[1]$. As $\mathcal{I}$ satisfies $P_2(3, \mathtt{t}), \mathcal{I}$ satisfies $P_2[2](\mathtt{t})$ and, thus, $P_1[1](\mathtt{t})$. As $0.7 = \max(0, 1 - 3/10)$, it follows that $\langle 3, 0.7\rangle$ evaluates the body of q' true in $\mathcal{I}$. On the other hand, $\langle 3, 0.64\rangle$ evaluates the body of q'' true in $\mathcal{I}$. Hence, under $\mathcal{I}$ the maximal score for 3 is 0.7, *i.e.*, $q^{\mathcal{I}}(3) \geqslant 0.7$. The other cases can be shown similarly. In summary, it can be shown that the top-4 answer set of $\mathbf{q}$ is $ans_4(\mathcal{K}, \mathtt{q}) = [\langle 0, 1.0\rangle, \langle 1, 0.9\rangle, \langle 2, 0.84\rangle, \langle 3, 0.7\rangle]$.

4. Query answering

In this section we will describe our top-k query answering procedure, which consists of four steps: (i) query reformulation; (ii) redundancy elimination; (iii) query evaluation; and (iv) result merging. Specifically, given a query $q(\mathbf{x})[s] \leftarrow \exists \mathbf{y} \phi(\mathbf{x}, \mathbf{y})[\mathbf{s}]$,

1. by considering $\mathcal{O}$, the user query $\mathbf{q}$ is *reformulated* into a set of conjunctive queries $r(\mathtt{q}, \mathcal{O})$. Informally, the basic idea is that the reformulation procedure closely resembles a top-down resolution procedure for logic programming, where each axiom is seen as a logic programming rule. For instance, given the query

$$q(x)[s] \leftarrow A(x)[s'], \mathsf{OrderBy}(s = f(s'))$$

and suppose that $\mathcal{O}$ contains the axioms

$$t_1(B_1, B_2) \sqsubseteq A$$
$$t_2(C_1, C_2) \sqsubseteq A,$$

then we can reformulate the query into two queries

$$q(x)[s] \leftarrow B_1(x)[s_1], B_2(x)[s_2], \mathsf{OrderBy}(s = f(t_1(s_1, s_2)))$$
$$q(x)[s] \leftarrow C_1(x)[s_1], C_2(x)[s_2], \mathsf{OrderBy}(s = f(t_2(s_1, s_2)));$$

Note that the reason for, *e.g.*, the expression $f(t_1(s_1,s_2))$ follows informally from the following argument. From $B_i(x) \geqslant s_i$, $t_1(B_1(x), B_2(x)) \leqslant A(x)$ and the monotonicity of t, we have $A(x) \geqslant t(s_1,s_2)$. So, for $s' = t(s_1,s_2)$, $q(x) \geqslant s = f(s') = f(t_1(s_1,s_2))$.
2. from the set of reformulated queries $r(\mathtt{q}, \mathcal{O})$ we remove redundant queries;
3. the reformulated queries $q' \in r(\mathtt{q}, \mathcal{O})$ are translated to top-k SQL queries and evaluated. The query evaluation of each top-k SQL query returns the top-k answer set for that query;
4. all the $n = |r(\mathtt{q}, \mathcal{O})|$ top-k answer sets have to be merged into the unique top-k answer set $ans_k(\mathcal{K}, \mathtt{q})$. To this purpose, we describe a method to merge all the answer sets without necessarily computing the union.

We next address in more detail the above steps.

4.1. Query reformulation

The query reformulation step is a generalisation of [8,24,26] to our case and is as follows.

Bound, unbound variable. We say that a variable in a conjunctive query is *bound* if it corresponds to either a distinguished variable or a shared variable, *i.e.*, a variable occurring at least twice in the query body, or a constant, while we say that a variable is *unbound* if it corresponds to a non-distinguished non-shared variable (as usual, we use the symbol "_" to represent non-distinguished non-shared variables). Note that an expression $R[i_1,\ldots,i_k]$ can be seen as the relation $R(\mathbf{x})$, where the variables in position $i_1, \ldots, i_k$ are unbound. We write also $R(\_,\ldots,\_,x_{i_1},\ldots,x_{i_k},\_,\ldots,\_)$ to denote the relation $R(\mathbf{x})$ in which all variables except those in position $i_1, \ldots, i_k$ are unbound. Given a vector of variables $\mathbf{x}$, and a condition *Cond* occurring in the left hand side of an axiom then $Cond(\mathbf{x})$ is defined as follows: $([i] \leqslant v)(\mathbf{x}) = (x_i \leqslant v)$, $([i] < v)(\mathbf{x}) = (x_i < v)$, $([i] \geqslant v)(\mathbf{x}) = (x_i \geqslant v)$, $([i] > v)(\mathbf{x}) = (x_i > v)$, $([i] = v)(\mathbf{x}) = (x_i = v)$, and $([i] \neq v)(\mathbf{x}) = (x_i \neq v)$.

Applicable axiom. An axiom τ is *applicable* to an atom $A(x)[s]$ in a query body, if τ has A in its right hand side, while τ is applicable to an atom $R(\_,\ldots,\_,x_{i_1},\ldots,x_{i_k},\_,\ldots,\_)[s]$ in a query body, if the right hand side of τ is $R[i_1,\ldots,i_k]$. We indicate with $gr(g;\tau)$ the expression obtained from the atom or relation g by applying the inclusion axiom τ and with $\theta(g;\tau)$ the variable substitution obtained from the atom or relation g by applying the inclusion axiom τ. Specifically, we have the following.

- If g is $A(x)[s]$ and τ is $t(Rl_1,\ldots,Rl_m) \sqsubseteq A$ then

$$gr(g;\tau) = \{C_1(x)[s_1],\ldots,C_m(x)[s_m]\},$$

where for each $r \in \{1,\ldots,m\}$, s_r is a new scoring variable,

$$\theta(g;\tau) = \{s/t(s_1,\ldots,s_m)\},$$

and

- if $Rl_r = A_r$ then $C_r(x)[s_r]$ is $A_r(x)[s_r]$;
- if $Rl_r = R[j]$ then $C_r(x)[s_r]$ is $R_r(\_,\ \ldots,\_,x,\_,\ldots,\_)[s_r]$, where x is in the j-th position;
- if $Rl_r = R[j].(Cond_1,\ldots,Cond_h)$ then $C_r(x)[s_r]$ is

$$R_r(z)[s_r], Cond_1(z),\ldots,Cond_h(z),$$

where l is the arity of R, $\mathbf{z} = \langle z_1,\ldots,z_{j-1},x,z_{j+1},\ldots,z_l\rangle$ and all z_h are new variables.

- If g is $R(\_,\ldots,\_,x_{i_1},\ldots,x_{i_k},\_,\ldots,\_)[s]$ and τ is $t(Rl_1,\ldots,Rl_m) \sqsubseteq R[i_1,\ldots,i_k]$ then $gr(g;\tau)$ is

$$\{C_1(x_{i_1},\ldots,x_{i_k})[s_1],\ldots,C_m(x_{i_1},\ldots,x_{i_k})[s_m]\},$$

where for each $r \in \{1,\ldots,m\}$, s_r is a new scoring variable,

$$\theta(g;\tau) = \{s/t(s_1,\ldots,s_m)\},$$

and

- if $Rl_r = A_r$ then $k = 1$, and $C_r(x_{i_1})[s_r]$ is $A_r(x_{i_1})[s_r]$;
- if $Rl_r = R[j_1,\ldots,j_k]$ then $C_r(x_{i_1},\ldots,x_{i_k})[s_r]$ is

$$R_r(\_,\ldots,\_,x_{i_1},\ldots,x_{i_k},\_,\ldots,\_)[s_r],$$

where variables in position $j_1, \ldots, j_k$ are $x_{i_1},\ldots,x_{i_k}$;

- if $Rl_r = R[j_1,\ldots,j_k].(Cond_1,\ldots,Cond_h)$ then $C_r(x_{i_1},\ldots,x_{i_k})[s_r]$ is

$$R_r(z)[s_r], Cond_1(z),\ldots,Cond_h(z),$$

where l is the arity of R, $z = \langle z_1,\ldots,x_{i_1},\ldots,x_{i_k},\ldots,z_l\rangle$, all z_h are new variables and the variables in position $j_1, \ldots, j_k$ are $x_{i_1},\ldots,x_{i_k}$.

Example 4.1. Consider the query

$$q_0 := q(x)[s] \leftarrow A(x)[s_1], B(x)[s_2], \mathsf{OrderBy}(s = \min(s_1,s_2))$$

and suppose that $\mathcal{O}$ contains the axioms

$$\tau_1 := 0.8 \cdot B_1 \sqsubseteq A$$
$$\tau_2 := 0.7 \cdot B_2 \sqsubseteq A.$$

Then for $g = A(x)[s_1]$ we have

$$gr(g;\tau_1) := \{B_1(x)[s_3]\}, \theta(g;\tau_1) := \{s_1/0.8 \cdot s_3\}$$
$$gr(g;\tau_2) := \{B_2(x)[s_4]\}, \theta(g;\tau_2) := \{s_1/0.7 \cdot s_4\}.$$

Now, using $gr(g;\tau_i)$ and $\theta(g;\tau_i)$ we may reformulate the original query by replacing in the query q_0, the expression g with the elements in $gr(g;\tau_i)$ and then applying the score variable substitution $\theta(g;\tau_i)$ to the scoring atom. Therefore, we get two new queries

$$q_1 := q(x)[s] \leftarrow B_1[s_3], B(x)[s_2], \mathsf{OrderBy}(s = \min(0.8 \cdot s_3, s_2))$$
$$q_2 := q(x)[s] \leftarrow B_2[s_4], B(x)[s_2], \mathsf{OrderBy}(s = \min(0.7 \cdot s_4, s_2)).$$

Algorithm 1. *QueryRef*$(\mathtt{q}, \mathcal{O})$

Input: A disjunctive query **q**, axioms $\mathcal{O}$.
Output: Set of reformulated conjunctive queries $r(\mathtt{q}, \mathcal{O})$.
1: $r(\mathtt{q}, \mathcal{O}) := \mathtt{q}$
2: **repeat**
3: $S = r(\mathtt{q}, \mathcal{O})$
4: **for all** $q \in S$ **do**
5: **for all** $g \in q$ **do**
6: **if** $\tau \in \mathcal{O}$ is applicable to g **then**
7: $r(q, \mathcal{O}) := r(q, \mathcal{O}) \cup \{q[g/gr(g,\tau)]\theta(g,\tau)\}$
8: **for all** $g_1, g_2 \in q$ **do**
9: **if** g_1 and g_2 unify **then**
10: $r(q, \mathcal{O}) := r(q, \mathcal{O}) \cup \{\kappa(reduce(q, g_1, g_2))\}$
11: $r(\mathtt{q}, \mathcal{O}) := removeSubs(r(\mathtt{q}, \mathcal{O}))$
12: **until** $S = r(\mathtt{q}, \mathcal{O})$
13: **return** $r(\mathtt{q}, \mathcal{O})$

We are now ready to present the query reformulation algorithm (see Algorithm 1). Given a disjunctive query **q** and a set of axioms $\mathcal{O}$, the algorithm reformulates q in terms of a set of conjunctive queries $r(\mathtt{q}, \mathcal{O})$, which then can be evaluated over the facts $\mathcal{F}$ using the mappings in the mapping component $\mathcal{M}$.

In the algorithm, the expression $q[g/g']\theta(g,\tau)$ denotes the query obtained from q by replacing the atom g with a new atom g'. To the resulting query we apply the score variable substitution $\theta(g;\tau_i)$ to the scoring atom. At step 8, for each pair of atoms g_1, g_2 that unify, the algorithm computes the query $q' = reduce(q,g_1,g_2)$, by applying to q the most general unifier between g_1 and g_2.[3] Due to the unification, variables that were bound in q may become unbound in q'. Hence, inclusion axioms not applicable to atoms of q, may become applicable to atoms of q' (in the next executions of step (5)). Function κ applied to q' replaces with _ each unbound variable in q'. Finally, in step 11 we remove from the set of queries $r(\mathtt{q}, \mathcal{O})$, those which are already subsumed in $r(\mathtt{q}, \mathcal{O})$.

Query subsumption. The notion of query subsumption is similar as for the classical database theory [34]. Given two queries $q_i (i = 1, 2)$ with same head $q(\mathbf{x})[s]$ and $q_1 \neq q_2$, we say that q_1 *is subsumed by* q_2, denoted $q_1 \sqsubseteq q_2$, iff for any interpretation $\mathcal{I}$, for all tuples **c**, $q_1^{\mathcal{I}}(\mathtt{c}) \leqslant q_2^{\mathcal{I}}(\mathtt{c})$. Essentially, if $q_1 \sqsubseteq q_2$ and both q_1 and q_2 belong to $r(\mathtt{q}, \mathcal{O})$ then we can remove q_1 from $r(\mathtt{q}, \mathcal{O})$ as q_1 produces a lower ranked result than q_2 with respect to the same tuple **c**.

A condition for query subsumption is the following. Assume that q_1 and q_2 do not share any variable. This can be accomplished by renaming all variables in *e.g.* q_1. Then it can be shown in a similar way as in [26] that.

Proposition 4.2. *If q_1 and q_2 share the same score combination function, then $q_1 \sqsubseteq q_2$ iff there is a variable substitution θ such that for each relation $R(\mathbf{z}_2)$ occurring in the rule body of q_2 there is a relation $R(\mathbf{z}_1)$ occurring in the rule body of q_1 such that $R(\mathbf{z}_2) = R(\mathbf{z}_1)\theta$.* □

More complicated are cases in which q_1 and q_2 do not share the same score combination function.

Example 4.3 (Example 4.1 cont.). Suppose that the ontology component $\mathcal{O}$ has additionally the recursive axiom $\tau_3 := 0.9 \cdot A \cdot B_3 \sqsubseteq A$. Therefore, through the query reformulation procedure we get a new query

$$q_3 := q(x)[s] \leftarrow A(x)[s_5], B_3(x)[s_6], B(x)[s_2], \mathsf{OrderBy}(s = \min(0.9 \cdot s_5 \cdot s_6, s_2))$$

Now, let's compare q_0 with q_3. It turns out that $q_3 \sqsubseteq q_0$ as for any score n of the ground atom $A(c)$, $\min(0.9 \cdot n \cdot s_6, s_2) \leqslant \min(n, s_2)$, for any value for s_2 and s_6 (as the scoring function in the axioms are bounded).

We can extend the query subsumption condition in Proposition 4.2 in the following way. Let q_1 and q_2 be two queries and let σ_1 and σ_2 be the scoring component of q_1 and q_2, respectively. Then as in [26], it can be shown that.

[3] We say that two atoms $g_1 = r(x_1,\ldots,x_n)$ and $g_2 = r(y_1,\ldots,y_n)$ *unify*, if for all i, either $x_i = y_i$ or $x_i = \_$ or $y_i = \_$. If g_1 and g_2 unify, then the unification of g_1 and g_2 is the atom $r(z_1,\ldots,z_n)$, where $z_i = x_i$ if $x_i = y_i$ or $y_i = \_$, otherwise $z_i = y_i$ [6].

Proposition 4.4. *$q_1 \sqsubseteq q_2$ iff there is a variable substitution θ such that for each relation $R(\mathbf{z}_2)$ occurring in the rule body of q_2 there is a relation $R(\mathbf{z}_1)$ occurring in the rule body of q_1 such that $R(\mathbf{z}_2) = R(\mathbf{z}_1)\theta$, and $\sigma_1\theta \leqslant \sigma_2$ for all variables occurring in $\sigma_1\theta$ and σ_2.* □

Example 4.5 (Example 4.3 cont.). Let $\theta = \{s_5/s_1\}$ then $\sigma_0 = \min(s_1, s_2)$, while $\sigma_3\theta = \min(0.9 \cdot s_1 \cdot s_6, s_2)$. It is easily verified that $\min(0.9 \cdot s_1 \cdot s_6, s_2) = \sigma_3\theta \leqslant \sigma_0 = \min(s_1, s_2)$ as the score combination function is monotone in its arguments and $0.9 \cdot s_1 \cdot s_6 \leqslant s_1$.

We use Proposition 4.4 to avoid the recursive application of the query reformulation steps (see, *e.g.*, Example 4.3). In such cases the check of the condition $\sigma_1\theta \leqslant \sigma_2$ is easy due to the monotonicity of the score combination function, the monotonicity and boundness of the scoring function in axioms. This concludes the query reformulation step.

The following is a complete example.

Example 4.6. Consider Example 3.3. At step 1, $r(\mathtt{q}, \mathcal{O})$ is initialized with $\{q', q''\}$. It is easily verified that both conditions in step 6 and step 9 fail for q''. So we proceed with q'. Let σ be $s = \max(0, 1 - x/10)$. Then at the first execution of step 7, the algorithm inserts query q_1,

$$q_1 := q(x)[s] \leftarrow P_2(x, y), A(y), \mathsf{OrderBy}(\sigma)$$

into $r(\mathtt{q}, \mathcal{O})$ using the axiom $A \sqsubseteq P_1[1]$.

At the second execution of step 7, the algorithm inserts query q_2,

$$q_2 := q(x)[s] \leftarrow P_2(x, y), P_2(\_, y), \mathsf{OrderBy}(\sigma)$$

using the axiom $P_2[2] \sqsubseteq A$. Since the two atoms of the second query unify, $\mathit{reduce}(q, g_1, g_2)$ returns

$$q(x)[s] \leftarrow P_2(x, y), \mathsf{OrderBy}(\sigma)$$

and since now y is unbound (y does not occur in σ), after application of κ, step 10 inserts the query q_3,

$$q_3 := q(x)[s] \leftarrow P_2(x, \_), \mathsf{OrderBy}(\sigma)$$

At the third execution of step 7, the algorithm inserts query q_4,

$$q_4 := q(x)[s] \leftarrow B(x), \mathsf{OrderBy}(\sigma)$$

using the axiom $B \sqsubseteq P_2[1]$ and stops. Note that we need not to evaluate all queries q_i. Indeed, it can be verified that for each q_i and all constants c, the scores of q_3 and q_4 are not lower than all the other queries q_i and q'. That is, we can restrict the evaluation of the set of reformulated queries to $r(\mathtt{q}, \mathcal{O}) = \{q'', q_3, q_4\}$ only. As a consequence, the top-4 answers to the original query are $[\langle 0, 1.0\rangle, \langle 1, 0.9\rangle, \langle 2, 0.84\rangle, \langle 3, 0.7\rangle]$, which are the top-4 ranked tuples of the union of the answer sets of q'', q_3 and q_4.

4.2. Computing top-k answers

The main property of the query reformulation algorithm is as follows:

$$\mathit{ans}_k(\mathcal{K}, \mathtt{q}) = \mathtt{Top}_k\{\langle \mathtt{c}, v\rangle | q_i \in r(\mathtt{q}, \mathcal{O}), \mathcal{F} \models q_i(\mathtt{c}, v)\}.$$

The above property dictates that the set of reformulated queries $q_i \in r(\mathtt{q}, \mathcal{O})$ can be used to find the top-k answers, by evaluating them over the set of instances $\mathcal{F}$ only, *i.e.*, over the database, without referring to the ontology $\mathcal{O}$ anymore. In the following, we show how to find the top-k answers of the union of the answer sets of conjunctive queries $q_i \in r(\mathtt{q}, \mathcal{O})$.

A naive solution would be: we compute for all $q_i \in r(\mathtt{q}, \mathcal{O})$ the whole answer set $\mathit{ans}(q_i, \mathcal{F}) = \{\langle \mathtt{c}, v\rangle | \mathcal{F} \models q_i(\mathtt{c}, v)\}$, then we compute the union, $\bigcup_{q_i \in r(\mathtt{q}, \mathcal{O})} \mathit{ans}(q_i, \mathcal{O})$, of these answer sets, order it in descending order of the scores and then we take the top-k tuples. We note that each conjunctive query $q_i \in r(\mathtt{q}, \mathcal{O})$ can easily be transformed into a top-k SQL query expressed over $\mathtt{DB}(\mathcal{F})$, *i.e.*, the database encoding $\mathcal{F}$. The transformation is conceptually simple.

A major drawback of this solution is the fact that there might be too many tuples with non-zero score and hence for any query $q_i \in r(\mathtt{q}, \mathcal{O})$, all these scores should be computed and the tuples should be retrieved. This may *not be feasible* in practice. Indeed, in our experiments (see Fig. 5, queries 1, 2, 7), in some cases this approach did not work at all due to main memory problems and motivated the algorithm described in the next sections, called *Disjunctive Thershold Algorithm*, initially proposed in [26], that does address the case for top-k join queries only. Therefore, in the following we first recap from [26] the case of top-k join queries (Section 4.2.1) and then describe our novel algorithm to handle queries with ranking aggregates too (Section 4.2.2).

4.2.1. The DTA for top-k join queries

An immediate method to compute $\mathit{ans}_k(\mathcal{K}, \mathtt{q})$ for top-k join queries is to compute for all $q_i \in r(\mathtt{q}, \mathcal{O})$, the top-$k$ answers $\mathit{ans}_k(\mathcal{F}, q_i)$. If both k and the number, $n_q = |r(\mathtt{q}, \mathcal{O})|$, of reformulated queries is reasonable, then we may compute the union,

$$U(q,\mathcal{K}) = \bigcup_{q_i \in r(\mathtt{q},\mathcal{O})} ans_k(\mathcal{F}, q_i),$$

of these top-k answer sets, order it in descending order w.r.t. score and then we take the top-k tuples.

As an alternative, we can avoid to compute the whole union $U(q,\mathcal{K})$, so further improving the answering procedure, by relying on a *disjunctive* variant [26] of the so-called *Threshold Algorithm* (TA) [11], called *Disjunctive TA* (DTA).

We recall that the TA has been developed to compute the top-k answers of a conjunctive query with monotone score combination function. In the following we show that we can use the same principles of the TA to compute the top-k answers of the union of conjunctive queries *without* ranking aggregates:

1. First, we compute for all $q_i \in r(\mathtt{q},\mathcal{O})$, the top-$k$ answers $ans_k(\mathcal{F}, q_i)$, using top-k rank-based relational database engine. Now, let us assume that the tuples in the top-k answer set $ans_k(\mathcal{F}, q_i)$ are sorted in decreasing order with respect to the score.
2. Then we process each top-k answer set $ans_k(\mathcal{F}, q_i)$ ($q_i \in r(\mathtt{q},\mathcal{O})$) according to some criteria (*e.g.*, in parallel, or alternating fashion, or by selecting the next tuple from the answer set with highest threshold θ_i defined below), and top-down (*i.e.* the higher scored tuples in $ans_k(\mathcal{F}, q_i)$ are processed before the lower scored tuples in $ans_k(\mathcal{F}, q_i)$).
   (a) For each processed tuple $\mathbf{c}$, if its score is one of the k highest we have already computed, then remember tuple $\mathbf{c}$ and its score $s_{\mathbf{c}}$ (ties are broken arbitrarily, so that only k tuples and their scores need to be remembered at any time).
   (b) For each answer set $ans_k(\mathcal{F}, q_i)$, let θ_i be the score of the last tuple processed in this set. Define the threshold value θ to be

   $$\theta = \max(\theta_1, \ldots, \theta_{n_q}).$$

   (c) As soon as at least k tuples have been processed whose score is at least equal to θ, then halt (indeed, any successive retrieved tuple will have score $\leqslant \theta$).
   (d) Let Y be the set containing the k tuples that have been processed with the highest scores. The output is then the set $\{\langle \mathbf{c}, s_{\mathbf{c}}\rangle | \mathbf{c} \in Y\}$. This set is $ans_k(\mathcal{K}, \mathtt{q})$.

It is not difficult to see that the DTA determines the top-k answers. Indeed, if at least k tuples have been processed whose score is at least equal to θ then any new not yet processed tuple $\mathbf{c}$ will have score bounded by θ and, thus, it cannot make it into the top-k. Hence, we can stop and the top-k tuples are among those already processed. The following example illustrates the DTA.

Example 4.7. Consider Example 4.6. Suppose we are interested in retrieving the top-3 answers of the disjunctive query $\mathbf{q} = \{q', q''\}$. We have seen that it suffices to find the top-3 answers of the union of the answers to q_3, q_4 and to q''. Let us show how the DTA works. First, we submit q_3, q_4 and q'' to a rank-based relational database engine, to compute the top-3 answers. It can be verified that

$$ans_3(\mathcal{F}, q_3) = [\langle 0, 1.0\rangle, \langle 3, 0.7\rangle, \langle 4, 0.6\rangle]$$
$$ans_3(\mathcal{F}, q_4) = [\langle 1, 0.9\rangle, \langle 2, 0.8\rangle, \langle 5, 0.5\rangle]$$
$$ans_3(\mathcal{F}, q'') = [\langle 2, 0.84\rangle, \langle 3, 0.64\rangle, \langle 4, 0.36\rangle].$$

The lists are in descending order w.r.t. the score from left to right. Now we process alternatively $ans_k(\mathcal{F}, q_3)$, then $ans_k(\mathcal{F}, q_4)$ and then $ans_k(\mathcal{F}, q'')$ in decreasing order of the score. The table below summaries the execution of our DTA algorithm. The ranked list column contains the list of tuples processed.

Step	Tuple	θ_{q_3}	θ_{q_4}	$\theta_{q''}$	θ	Ranked list
1	$\langle 0,1.0\rangle$	1.0	–	–	–	$\langle 0,1.0\rangle$
2	$\langle 1,0.9\rangle$	1.0	0.9	–	–	$\langle 0,1.0\rangle$, $\langle 1,0.9\rangle$
3	$\langle 2,0.84\rangle$	1.0	0.9	0.84	1.0	$\langle 0,1.0\rangle$, $\langle 1,0.9\rangle$, $\langle 2,0.84\rangle$
4	$\langle 3,0.7\rangle$	0.7	0.9	0.84	0.9	$\langle 0,1.0\rangle$, $\langle 1,0.9\rangle$, $\langle 2,0.84\rangle\langle 3,0.7\rangle$
5	$\langle 2,0.8\rangle$	0.7	0.8	0.84	0.84	$\langle 0,1.0\rangle$, $\langle 1,0.9\rangle$, $\langle 2,0.84\rangle\langle 3,0.7\rangle$

At step 5 we stop as the ranked list already contains three tuples above the threshold $\theta = 0.84$. So, the final output is $ans_k(\mathcal{F}, q_3) = [\langle 0, 1.0\rangle, \langle 1, 0.9\rangle, \langle 2, 0.84\rangle]$. Note that not all tuples have been processed.

As computing the top-k answers of each query $q_i \in r(\mathtt{q},\mathcal{O})$ requires (sub) linear time w.r.t. the database size, it is easily verified that the disjunctive TA algorithm is linear in data complexity.

Proposition 4.8 [26]. *Given $\mathcal{K} = \langle \mathcal{F}, \mathcal{O}, \mathcal{M}\rangle$ and a disjunctive query $\mathbf{q}$ without ranking aggregates, then the DTA determines $ans_k(\mathcal{K}, \mathtt{q})$ in (sub) linear time w.r.t. the size of $\mathcal{F}$.* □

Furthermore, the above method has the non-negligible advantage to be based on existing technology for answering top-k queries over relational databases, improves significantly the naive solution to the top-k retrieval problem, and is not difficult to implement.

4.2.2. The DTA with ranking aggregates

Unfortunately, in case where a ranking aggregate occurs in a query, the DTA has to be modified as the stopping condition does not work anymore, as illustrated in the following example.

Example 4.9. Suppose that $r(\mathtt{q},\mathcal{O}) = \{q_1, q_2\}$, where

$$q_1 := q(x)[s] \leftarrow R(x,y)[s_1], \mathsf{GroupBy}(x,y), \mathsf{OrderBy}(s = \mathsf{Min}[s_1])$$
$$q_2 := q(x)[s] \leftarrow P(x,y)[s_1], \mathsf{GroupBy}(x,y), \mathsf{OrderBy}(s = \mathsf{Min}[s_1])$$

and that

$$ans_3(\mathcal{F}, q_1) = [\langle a, 1.0\rangle, \langle b, 0.7\rangle, \langle d, 0.4\rangle]$$
$$ans_3(\mathcal{F}, q_2) = [\langle d, 0.9\rangle, \langle e, 0.6\rangle, \langle f, 0.5\rangle].$$

By applying the DTA, we would get

Step	Tuple	θ_{q_1}	θ_{q_2}	θ	Ranked list
1	$\langle a, 1.0\rangle$	1.0	–	–	$\langle a, 1.0\rangle$
2	$\langle d, 0.9\rangle$	1.0	0.9	1.0	$\langle a, 1.0\rangle$, $\langle d, 0.9\rangle$
3	$\langle b, 0.7\rangle$	0.7	0.9	0.9	$\langle a, 1.0\rangle$, $\langle d, 0.9\rangle$, $\langle b, 0.7\rangle$
4	$\langle e, 0.6\rangle$	0.7	0.6	0.7	$\langle a, 1.0\rangle$, $\langle d, 0.9\rangle$, $\langle b, 0.7\rangle$, $\langle e, 0.6\rangle$

We stop at Step 4, as we have three answers above the threshold $\theta = 0.8$. However, this is not correct as $\langle d, 0.9\rangle$ is not the score for d. In fact, as there is still $\langle d, 0.4\rangle \in ans_3(\mathcal{F}, q_1)$ and, by applying the ranking aggregate function, $\min(0.9, 0.4) = 0.4$, $\langle d, 0.4\rangle$ is the score for d and, thus, $\langle d, 0.4\rangle$ is not in the top-3.

The next example shows an even more serious problem.

Example 4.10. Suppose that $r(\mathtt{q},\mathcal{O}) = \{q_1, q_2\}$, where

$$q_1 := q(x)[s] \leftarrow R(x,y)[s_1], \mathsf{GroupBy}(x,y), \mathsf{OrderBy}(s = \mathsf{SUM}[s_1])$$
$$q_2 := q(x)[s] \leftarrow P(x,y)[s_1], \mathsf{GroupBy}(x,y), \mathsf{OrderBy}(s = \mathsf{SUM}[s_1])$$

and that we are looking for the top-1 answer. Assume that the answers are

$$ans(\mathcal{F}, q_1) = [\langle a, 1.0\rangle, \langle b, 0.4\rangle, \langle e, 0.3\rangle]$$
$$ans(\mathcal{F}, q_2) = [\langle b, 0.9\rangle, \langle e, 0.2\rangle, \langle a, 0.1\rangle,].$$

Hence, the score for tuple a should be 1.0 + 0.1 = 1.1, the score for b should be 0.9 + 0.4 = 1.3, and the score for e should be 0.3 + 0.2 = 0.5. Therefore, the top-1 answer is b with score 1.3. Now, if we would rely on submitting just a top-1 query to the underlying database engine, as we do for the current DTA, we get $ans_1(\mathcal{F}, q_1) = [\langle a, 1.0\rangle]$ and $ans_1(\mathcal{F}, q_2) = [\langle b, 0.9\rangle]$. Then the score for tuple a would be 1.0, while the score for b would be 0.9 and, thus, not only we get incorrect scores, but also more importantly we get a wrong ranking result (tuple a would be top-1).

As shown in Examples 4.9 and 4.10, in presence of ranking aggregates, not yet processed tuples in top-k answers, may affect the final score. This is the case for aggregation functions $@ \in \{\mathsf{MIN}, \mathsf{AVG}, \mathsf{SUM}\}$, but not for $@ = \mathsf{MAX}$. In fact, we have the following.

Case $@ = \mathsf{MAX}$. In this case, no modification to the DTA is required as the DTA implicitly retrieves the maximal score for each tuple, *i.e.*

Proposition 4.11. *Given $\mathcal{K} = \langle \mathcal{F}, \mathcal{O}, \mathcal{M}\rangle$ and a disjunctive query* $\mathbf{q}$ *with ranking aggregate* MAX *only, then the DTA computes $ans_k(\mathcal{K}, \mathtt{q})$ in (sub) linear time w.r.t. the size of $\mathcal{F}$.* □

It remains to address the cases $@ \in \{\mathsf{MIN}, \mathsf{AVG}, \mathsf{SUM}\}$.

In the following, for all $q_j \in r(\mathtt{q},\mathcal{O})$, consider the answer set $L_j = ans(\mathcal{F}, q_j)(1 \leqslant j \leqslant n = |r(\mathtt{q},\mathcal{O})|)$. Let $\mathbf{c}_i$ be a tuple and assume that it occurs in the $r \leqslant n$ lists $L_{i_1}, \ldots, L_{i_r}$ with score $s_{i_1}, \ldots, s_{i_r}$. We will say that the *definitive score* of $\mathbf{c}_i$ is $@[s_{i_1}, \ldots, s_{i_r}]$. Note that, $\langle \mathtt{c}, s\rangle \in ans(\mathcal{K}, \mathtt{q})$ is an answer for $\mathbf{q}$ iff s is the definitive score of $\mathbf{c}$.

In case we are interested in the top-k answers to a query $\mathbf{q}$, Example 4.10 points out that if we consider $L_j^k = ans_k(\mathcal{F}, q_j)$ in place of $L_j = ans(\mathcal{F}, q_j)$, we may not necessarily be able to compute from the top-k ranked lists L_j^k the definitive score of a

tuple $\mathbf{c}$ occurring in some L_j^k. Of course, if $\mathbf{c}$ occurs in *all* top-k ranked lists $L_1^k, \ldots, L_n^k$ with score $s_1, \ldots, s_n$ then $s = @[s_1, \ldots, s_n]$ is the definitive score of $\mathbf{c}$.

In the following, let gw a non-negative integer denoting a value called *group window*. For positive integer $i \geqslant 1$, with $ans_{i,gw}(\mathcal{F}, q)$ we will denote the ranked list of gw answers of q, ranked between position $(i-1)\cdot gw + 1$ and $i \cdot gw$, *i.e.*

$$ans_{i,gw}(\mathcal{F}, q) = ans_{i\cdot gw}(\mathcal{F}, q) \setminus ans_{(i-1)\cdot gw}(\mathcal{F}, q),$$

where $ans_0(\mathcal{F}, q) = \emptyset$. For instance, for $gw = 10, ans_{1,10}(\mathcal{F}, q)$ are the top-10 answers to q, while $ans_{2,10}(\mathcal{F}, q)$ are the answers to q ranked in position [11,...,20]. We will use $ans_{i,gw}(\mathcal{F}, q)$ to incrementally retrieve a successive ordered group of gw answers to q.[4]

Case $@ \in \{\mathsf{MIN}, \mathsf{AVG}, \mathsf{SUM}\}$.

1. Initialization: let $n = |r(\mathfrak{q}, \mathcal{O})|$, for all $q \in r(\mathfrak{q}, \mathcal{O})$, let $i_q = 1$, $L_q = \emptyset$ and let $\theta = \top$. Compute for all $q \in r(\mathfrak{q}, \mathcal{O})$, the top-$gw$ answers and let $L_q = ans_{gw}(\mathcal{F}, q)$, using top-$gw$ rank-based relational database engine. We will assume that the tuples in the top-gw answer set $ans_{gw}(\mathcal{F}, q_i)$ are sorted in decreasing order with respect to the score.
2. Then we process each answer set L_q according to some criteria (*e.g.*, in parallel, or alternating fashion, or by selecting the next tuple from the answer set with highest threshold θ_q defined below), and top-down (*i.e.* the higher scored tuples in L_q are processed before the lower scored tuples in L_q).
   (a) For each answer set L_q, let θ_{L_q} be the score of the last tuple processed in this set.
   (b) If for a list L_q we have processed the last tuple in it and we did not stop, then update $i_q := i_q + 1$ and append $ans_{i_q,gw}(\mathcal{F}, q)$ to L_q (*i.e.*, retrieve the next gw answers of q). If $ans_{i_q\cdot gw}(\mathcal{F}, q) = \emptyset$ do not process further L_q and set $\theta_q = 0$. Such a list L_q is called *completed*.
   (c) For each tuple $\mathbf{c}$ processed so far, let $\overline{\mathcal{L}}_{\mathfrak{c}}$ be the set of (non-completed) ranked lists in which $\mathbf{c}$ has not yet been processed so far;
   (d) For each processed tuple $\mathbf{c}$, let $l_{\mathfrak{c}} = [s_{i_1}, \ldots, s_{i_r}]$ be the actual list of r scores computed so far for $\mathbf{c}$ and let $s_{\mathbf{c}} = @l_{\mathbf{c}}$ be the actual score of $\mathbf{c}$.
   (e) If $\mathbf{c}$ has been processed in all lists $L_{q_1}, \ldots, L_{q_n}$, or all lists in which $\mathbf{c}$ have not be processed so far are completed, then $s_{\mathbf{c}}$ is the definitive score of $\mathbf{c}$;
   (f) For each tuple $\mathbf{c}$ processed so far, which has not yet a definitive score, we define the *upper score* of $\mathbf{c}$, $\bar{s}_{\mathfrak{c}}$, as follows ($\bar{s}_{\mathfrak{c}}$ is the highest possible score $\mathbf{c}$ may eventually achieve):
      - if $@ = \mathsf{Min}$ then $\bar{s}_{\mathfrak{c}}$ is $s_{\mathbf{c}}$
      - if $@ = \mathsf{AVG}$ then $\bar{s}_{\mathfrak{c}}$ is defined as follows. If there is no $L \in \overline{\mathcal{L}}_{\mathfrak{c}}$ with $\theta_L > s_{\mathbf{c}}$ then $\bar{s}_{\mathfrak{c}} := s_{\mathfrak{c}}$. Otherwise, $\bar{s}_{\mathfrak{c}} := \left(r \cdot s_{\mathfrak{c}} + \max_{L\in\overline{\mathcal{L}}_{\mathfrak{c}}} \theta_L\right)/(r+1)$. In summary,

$$\bar{s}_{\mathfrak{c}} := \max\left(s_{\mathfrak{c}}, (r \cdot s_{\mathfrak{c}} + \max_{L\in\overline{\mathcal{L}}_{\mathfrak{c}}} \theta_L)/(r+1)\right).$$

      - if $@ = \mathsf{SUM}$ then $\bar{s}_{\mathfrak{c}}$ is

$$\bar{s}_{\mathfrak{c}} := s_{\mathfrak{c}} + \sum_{L\in\overline{\mathcal{L}}_{\mathfrak{c}}} \theta_L;$$

   (g) As soon as at least k tuples have been processed whose actual score is *definitive* and is at least equal to the definitive score or upper score of all other processed tuples, then halt.
   (h) Let Y be the set containing the k tuples that have been processed with the highest definitive score. The output is then the set $\{\langle \mathbf{c}, s_{\mathbf{c}}\rangle | \mathbf{c} \in Y\}$. This set is $ans_k(\mathcal{K}, \mathfrak{q})$.

Let us show that the stopping criteria works correctly and that we get the top-k ranked tuples. Consider the top-k ranked tuples $\langle \mathbf{c}_1, s_1\rangle, \ldots, \langle \mathbf{c}_k, s_k\rangle$, returned by the DTA above. Hence, k tuples have been processed whose actual score is *definitive* and is at least equal to the definitive score or upper score of all other processed tuples. Now, consider a processed tuple $\langle \mathbf{c}, s_{\mathbf{c}}\rangle$, which did not make it into the top-k. If the score $s_{\mathbf{c}}$ is definitive, it will not change any longer and, thus, the score of $\mathbf{c}$ is below s_k. If $s_{\mathbf{c}}$ is not definitive, then $\mathbf{c}$ may possibly occur in some of the ranked lists $L \in \overline{\mathcal{L}}_{\mathfrak{c}}$. Suppose that $\mathbf{c}$ will not eventually occur in some of the ranked lists $L \in \overline{\mathcal{L}}_{\mathfrak{c}}$. Then, the score $s_{\mathbf{c}}$ of $\mathbf{c}$ will be definitive and below s_k. Otherwise, if $\mathbf{c}$ will eventually occur with score s_L in a ranked list $L \in \overline{\mathcal{L}}_{\mathfrak{c}}$, then we have to consider all three cases $@ \in \{\mathsf{MIN}, \mathsf{AVG}, \mathsf{SUM}\}$. By the stopping criteria $s_k \geqslant \bar{s}_{\mathfrak{c}}$ holds.

Case $@ = \mathsf{MIN}$. As $\min(s_{\mathbf{c}}, s_L) \leqslant s_{\mathbf{c}}$, the definitive score of $\mathbf{c}$ will be below s_k.
Case $@ = \mathsf{AVG}$. We known that $s_L \leqslant \max_{L\in\overline{\mathcal{L}}_{\mathfrak{c}}} \theta_L$. Therefore, the definitive score of $\mathbf{c}$ cannot be greater than $\max_{L\in\overline{\mathcal{L}}_{\mathfrak{c}}} (r \cdot s_{\mathfrak{c}} + s_L)/(r+1) \leqslant (r \cdot s_{\mathfrak{c}} + \max_{L\in\overline{\mathcal{L}}_{\mathfrak{c}}} \theta_L)/(r+1) = \bar{s}_{\mathfrak{c}} \leqslant s_k$.

[4] We point out that $ans_{i,gw}(\mathcal{F}, q)$ can be computed, in *e.g.* RankSQL, by means of a statement of the form SELECT ... FROM ... GROUP BY ... ORDER *BY* ... DESCLIMIT *gw* OFFSET$(i-1)\cdot gw$.

Case @ = SUM. We known that $s_L \leqslant \max_{L\in\overline{\mathcal{L}}_c}\theta_L$. Therefore, the definitive score of **c** is $s_c + \sum_{L\in\overline{\mathcal{L}}_c} s_L \leqslant s_c + \sum_{L\in\overline{\mathcal{L}}_c}\theta_L = \bar{s}_c \leqslant s_k$, which concludes.

In the following, we call the DTA dealing with ranking aggregates DTA@. Hence, from the discussion above we have.

Proposition 4.12. *Given $\mathcal{K} = \langle \mathcal{F}, \mathcal{O}, \mathcal{M}\rangle$ and a disjunctive query* **q** *with ranking aggregates, then DTA@ computes $ans_k(\mathcal{K}, \mathtt{q})$ in (sub) linear time w.r.t. the size of $\mathcal{F}$.* □

Example 4.13. Consider Example 4.10. The execution of the DTA@ is shown below ($gw = 1$). For each tuple we hold the actual score and the upper score. The actual score is in bold if it is definitive and, in this, case we omit the upper score.

Step	Tuple	θ_{q_1}	θ_{q_2}	Ranked list
1	$\langle a, 1.0\rangle$	1.0	–	$\langle a, 1.0, 2.0\rangle$
2	$\langle b, 0.9\rangle$	1.0	0.9	$\langle a, 1.0, 1.9\rangle$, $\langle b, 0.9, 1.9\rangle$
3	$\langle b, 0.4\rangle$	0.4	0.9	$\langle b, \mathbf{1.3}\rangle$, $\langle a, 1.0, 1.9\rangle$
4	$\langle e, 0.2\rangle$	0.4	0.2	$\langle b, \mathbf{1.3}\rangle$, $\langle a, 1.0, 1.2\rangle$, $\langle e, 0.2, 0.6\rangle$

We stop at Step 4, as we have tuple *b* with definitive score 1.3 that is at least equal to the upper score of all other processed tuples. Note that none of the tuples *a*, *e* may make it into the top-1.

5. Experiments

We conducted an experiment to evaluate some aspects of our query answering procedure. To this end we considered our top-k retrieval system SoftFacts [1,33], that incorporates the proposed algorithm (however, in the SoftFacts system axioms are currently limited to the form $Rl_1 \sqcap \ldots \sqcap Rl_m \sqsubseteq Rr$). To this end, we considered an ontology for describing Curriculum Vitæ. It consists of 5115 axioms, 22 mapping statements and 2550 relations. The ontology has been used for "competence search" and is described in more details in [30,31].

We considered size $d = 500{,}000$ for the fact component consisting in automatically generated CVs and stored them into a database having 17 relational tables. We build several queries (see Appendix B), with/without scoring atom and with/without ranking aggregates. The queries have been submitted to the system for different values for *k* in case of top-k retrieval ($k \in \{1, 10\}$), and we measured for each query (time is measured in milliseconds).

1. the number of queries generated after the reformulation process ($|r(\mathtt{q}, \mathcal{O})|$);
2. the number of reformulated queries after redundancy elimination (q_{DB});
3. the time of the reformulation process (t_{ref});
4. the time of the query redundancy elimination process (t_{red});
5. the query answering time of the database, $t_{DB_{all}}$, t_{DB_1}, $t_{DB_{10}}$, *i.e.* time to retrieve all, top-1 and top-10 answers from the database.

Note that the measures 1–4 do neither depend on the number *d* of CVs nor on the number *k* for top-k retrieval.

We run the experiments using our top-k retrieval system, with indexes on the keys of the relational database tables. The tests have been performed on a MacPro machine with Mac OS X 10.6.2, 2 × 3 GHz Dual-Core processor and 9 GB or RAM. The database is Postgres,[5] which has been modified by RankSQL.

Fig. 4 reports the measures 1–4, which are independent of the relational database. We may note that the size of the set of reformulated queries $|r(\mathtt{q}, \mathcal{O})|$ may be non negligible, that the redundancy elimination may remove almost one third of them and the time of the reduction phase is negligible, except for query 9, which generates a large set of queries.

In Fig. 5 we report the retrieval times statistics. We also fixed the group window to $gw = 100$.

Let us consider few comments about the results:

- overall, the response time is reasonable (almost fraction of second, except for query 9) taking into account the non negligible size of the ontology and the number of CVs;
- if the answer set is large, *e.g.*, query 1, then there is a significant drop in response time, for the top-k case;
- for each query, the response time is increasing while we increase the number of retrieved records;
- the answering time of the database is increasing with the number of top-*k* results to be retrieved.

[5] http://www.postgres.org.

Query	$\|r(\mathbf{q},\mathcal{O})\|$	q_{DB}	t_{ref}	t_{red}
1	1599	1066	1.723	0.010
2	69	46	0.016	0.001
3	18	12	0.006	0.001
4	1242	552	2.072	0.015
5	75	50	0.027	0.001
6	18	12	0.005	0.001
7	69	46	0.013	0.001
8	1242	552	1.242	0.013
9	5175	2300	17.850	0.058
10	108	48	0.229	0.003
11	18	12	0.004	0.001
12	828	552	0.794	0.005
Average	961.5	468.4	2.318	0.01
Median	91.5	49	0.128	0.002

Fig. 4. Query reformulation statistics.

Size 500000							
Query	All	top-1	top-10	$\|ans(\mathcal{K},q)\|$	$t_{DB_{all}}$	t_{DB_1}	$t_{DB_{10}}$
1	-	1,852	1,907	$>$ 297195	-	0,142	0,174
2	-	0,083	0,180	$>$ 260470	-	0,030	0,106
3	18,005	0,136	0,198	12222	17,947	0,038	0,128
4	35,879	2,149	2,377	2259	33,930	0,078	0,232
5	23,086	0,060	0,064	13323	23,032	0,022	0,024
6	554,209	0,531	0,570	56293	554,185	0,513	0,537
7	-	1,558	1,707	$>$ 260470	-	1,534	1,551
8	51,050	2,283	4,279	10432	49,230	0,412	2,201
9	568,452	167,625	168,235	43462	550,792	150,038	150,370
10	260,607	3,072	3,082	37471	260,446	2,911	2,928
11	70,093	2,998	3,243	22247	70,072	2,971	3,227
12	56,535	47,189	47,677	8556	55,446	46,115	46,494
AVG	215,898	17,935	18,260	25066	212,795	15,572	15,825
Median	51,050	1,705	1,807	13323	49,230	0,277	0,385

Fig. 5. Retrieval statistics d = 500000.

The important aspect we wanted to highlight with this experiment is that for some queries *(1,2,7)* we were not able to compute all answers as the answer set becomes too big to be held in main memory (within the current prototypical implementation), while the relative top-k problems have been solved easily. We also point to query 9. In that case, even if the database responds within 0.07 s/query in average, the large size of queries (2300) prevents a reasonable query answering time so far. In this case we have both a large answer set as well as a large set of reformulated queries. Also, if $|r(\mathtt{q},\mathcal{O})|$ is small ($\leqslant$100), then the query answering time is almost the same as the database retrieval time. All the test data can be accessed at http://www.straccia.info/software/SoftFacts/Experiments/URLTest.zip.

6. Related work

While there are many works addressing the top-k problem for vague queries in databases (see *e.g.*, [14] for a survey), little is known for the corresponding problem in knowledge representation and reasoning. For instance, [36] considers non-recursive fuzzy logic programs in which the score combination function is a function of the score of the atoms in the body only (no scoring predicates are allowed). The work [25] considers non-recursive fuzzy logic programs as well, though the score combination function may consider scoring predicates. However, a score combination function is allowed in the query rule only. We point out that in the case of non-recursive rules and/or axioms, we may rely on a query rewriting mechanism, which, given an initial query, rewrites it, using rules and/or axioms of the KB, into a set of new queries until no new query rule can be derived (this phase may require exponential time relative to the size of the KB). The obtained queries may then be submitted directly to a top-k retrieval database engine. The answers to each query are then merged using the disjunctive threshold algorithm given in [25]. The works [24,26] address the top-k retrieval problem for the description logic *DL-Lite* only, though recursion is allowed among the axioms. Again, the score combination function may consider scoring predicates. However, a score combination function is allowed in the query only. The work [32] shows an application of top-k retrieval to

the case of multimedia information retrieval by relying on a fuzzy variant of *DLR-Lite* and is subsumed by the work here (as no scoring function is allowed in the axioms, no scoring function is allowed in the query language), while [30,31] report its use in the context of "competence search" (see also [19–21]). Finally, [27] addresses the top-k retrieval for general (recursive) fuzzy LPs, though no scoring predicates are allowed, which however are allowed in [15]. Closest to our work are [15,27]. In fact, our work extends [15,27] by allowing scoring predicates and scoring aggregate functions, which have the effect that the threshold mechanism designed in [15,27] does not work anymore and no DTA has been provided. Furthermore, in this paper, we made an effort to plug-in current top-k database technology, while [27] does not and provides an ad hoc solution.

We also refer the reader to [9] for a crisp DL-Lite implementation and application, and to [16] for a fuzzy DL-Lite language in the spirit of [26] (and less expressive language than ours) but not addressing specifically the top-k retrieval problem and ranking aggregates. Ref. [16] also reports some experiments on the LUBM benchmark,[6] which relies on a small ontology, around 80 axioms (around 40 atomic concepts, 20 binary relations). Therefore, $r(\mathtt{q}, \mathcal{O})$ is of almost of negligible size and, thus, so will be the query answer time (database engine retrieval time).

Finally, we point out the difference to so-called fuzzy relational databases (see, *e.g.* [5,37]), in which values of a column may be fuzzy (*e.g.*, the age of a person is "young"), whereas this is not the case here and in which the top-k retrieval problem is not addressed specifically.

7. Summary and outlook

The top-k retrieval problem is an important problem in logic-based languages for the Semantic Web. We have addressed this issue by describing an ontology mediated top-k information retrieval framework over relational databases. Indeed, an ontology layer is used to define, in terms of a tractable DLR-Lite like description logic, the relevant abstract concepts and relations of the application domain, facts are stored into a relational database, accessed via a mapping component. The results of a query may be ranked according to some scoring function and aggregation functions. We have illustrate the representation and the query language, and the query answering algorithm. We have also provided an experiment.

Concerning future work, we recall that the query reformulation step is a generalisation of [8,24,26] to our case. Recent works, such as [17] propose an alternative query rewriting method than [8] for the crisp case. We plan to investigate how and whether this method can be adapted to our case as well and how it impacts the size of the rewritings. Concerning the experimental part, we are planning extensive tests with the SoftFacts system also with other datasets, which will be the subject of a separate work (which also includes several practical optimisation methods, which are beyond the scope of this paper).

Appendix A. Relation to OWL QL

The syntax of *OWL QL* expressions is essentially as follows. In the following, A is an *atom* (unary predicate), R is a *role* (binary predicate) and R^- is the inverse of role R. Symbols may have a subscript.

Assertions. Facts are expressions of the form $A(c)$ and $R(c,d)$, specifically,

$$\mathsf{ClassAssertion}(A, c)$$
$$\mathsf{ObjectPropertyAssertion}(R, c, d)$$

Class expressions. In OWL 2 QL, there are two types of class expressions. The *subClass* production defines the class expressions that can occur as sub class expressions in *SubClassOf* axioms, and the *superClass* production defines the classes that can occur as super class expressions in *SubClassOf* axioms. Their syntax is:

$$\mathit{SubClass} \rightarrow A | \exists R$$
$$\mathit{SuperClass} \rightarrow A | \mathit{SuperClass}_1 \sqcap \mathit{SuperClass}_2 | \neg \mathit{SubClass} | \exists R.A$$

Here, $\exists R.A$ is a qualified existential quantification on roles and in FOL, it corresponds to the formula $\exists y.R(x,y) \wedge A(y)$, while $\exists R$ is an unqualified existential quantification on roles that corresponds to the formula $\exists y.R(x,y)$.

Class axioms. The class axioms of OWL 2 QL are of the following form

$$\mathsf{SubClassOf}(\mathit{SubClass}, \mathit{SuperClass})$$
$$\mathsf{EquivalentClasses}(\mathit{SubClass}_1, \mathit{SubClass}_2)$$
$$\mathsf{DisjointClasses}(\mathit{SubClass}_1, \mathit{SubClass}_2)$$

Property axioms. The property axioms of OWL 2 QL are of the following form

[6] http://swat.cse.lehigh.edu/projects/lubm/.

SubObjectPropertyOf(R_1, R_2)
EquivalentObjectProperties(R_1, R_2)
DisjointObjectProperties(R_1, R_2)
InverseObjectProperties(R_1, R_2)
ObjectPropertyDomain($R, SuperClass$)
ObjectPropertyRange($R, SuperClass$)
ReflexiveObjectProperty(R)
SymmetricObjectProperty(R)
AsymmetricObjectProperty(R)

On the translation of OWL 2 QL *to DL-Lite$_R$.* We recap here the translation of *OWL 2 QL* to DL-Lite$_R$ [10]. We recall that in DL-Lite$_R$ class and property (role) expressions have the following syntax (A is an atomic concept, P is a role and P^- is its inverse):

$$B \rightarrow A | \exists R$$
$$C \rightarrow B | \neg B$$
$$R \rightarrow P | P^-$$
$$E \rightarrow R | R^-.$$

Here, B is a so-called *basic concept*. $\exists R$ is an unqualified existential quantification on roles (in FOL, it corresponds to the formula $\exists y.R(x,y)$) and in our setting is simulated as the projection of R on the first column, *i.e.* $R[1]$. C denotes a (general) *concept*, which can be a basic concept or its negation, and E denotes a (general) *role*, which can be a basic role or its negation.

Inclusion axioms are of the form

$$B \sqsubseteq C \quad \text{and} \quad R \sqsubseteq E.$$

Precisely we refer to assertions of the form $B_1 \sqsubseteq B_2$ (resp. $R_1 \sqsubseteq R_2$) as *positive inclusion assertions* (PIs), and to assertions of the form $B_1 \sqsubseteq \neg B_2$ (resp. $R_1 \sqsubseteq \neg R_2$) as *negative inclusion assertions* (NIs). Therefore general concepts (resp., roles) may only occur on the right-hand side of inclusion assertions.

Note that there are some differences among DL-Lite$_R$ and *OWL 2 QL*:

1. existential quantification to a class $\exists R.C$;
2. symmetric property axioms and asymmetric property axioms;
3. reflexive property axioms and irreflexive property axioms.

Notice that, although 1. and 2. are not natively supported by DL-Lite$_R$, they are actually expressible in DL-Lite$_R$ by suitably processing the intensional level of the ontology. Conversely, reflexivity and irreflexivity axioms are brand new features and require new inference mechanisms.

Native handling of qualified existential quantification. It is well known that an axiom τ of the form

$$B \sqsubseteq \exists R.A$$

can be replaced with the transformation

$$\tau \mapsto \{B \sqsubseteq \exists R_{aux}, R_{aux} \sqsubseteq R, \exists R_{aux}^- \sqsubseteq A\},$$

where R_{aux} is a new role. We point out that [10] provides also a query reformulation method that does not require the translation.

Native handling of symmetric and asymmetric object property axioms.

1. each SymmetricObjectProperty(R) is managed by adding $R \sqsubseteq R^-$ to the DL component;
2. each AsymmetricObjectProperty(R) is managed by adding $R \sqsubseteq \neg R^-$ to the DL component;

Handling of Reflexive and irreflexive object property axioms. It has been observed in [10] that since (i) the asymmetry axiom on a property implies the irreflexivity axiom on the same property; and (ii) the asymmetry axiom influences only the consistency check on the ontology; the irreflexivity axiom influences only the consistency check, too. [10] shows how to deal with irreflexivity axioms in the consistency checking phase.

On the other hand, a reflexive role declarations impact the query reformulation phase, cannot be represented in DL-Lite$_R$ and they have to dealt with specifically in the reformulation algorithm, though the solution is easy. Essentially, a reflexive role R means that $R(x,x)$ is always true, we have to appropriately drop any occurrence of $R(x,x)$ occurring in a query during the query reformulation procedure. We refer the reader to [10] for the details.

Mapping *OWL 2 QL* statements. We now show how to map *OWL 2 QL* statements into DL_{Lite}. Assertions (class assertions and object property assertions) are mapped directly as DL-Lite$_R$ facts. Concerning roles, R^{--} is mapped into R and, thus, *OWL 2 QL* object properties are DL-Lite$_R$ roles.

Concerning class axioms, we have:

SubClassOf: an axiom $\mathsf{SubClassOf}(\mathit{SubClass}, \mathit{SuperClass})$ is transformed according the following rules:

$$\mathsf{SubClassOf}(\mathit{SubClass}, \mathit{SuperClass}_1 \sqcap \mathit{SuperClass}_2) \mapsto \mathsf{SubClassOf}(\mathit{SubClass}, \mathit{SuperClass}_1), \mathsf{SubClassOf}(\mathit{SubClass}, \mathit{SuperClass}_2)$$

$$\mathsf{SubClassOf}(\mathit{SubClass}_1, \mathit{SubClass}_2) \mapsto \mathit{SubClass}_1 \sqsubseteq \mathit{SubClass}_2$$

$$\mathsf{SubClassOf}(\mathit{SubClass}_1, \neg \mathit{SubClass}_2) \mapsto \mathit{SubClass}_1 \sqsubseteq \neg \mathit{SubClass}_2$$

$$\mathsf{SubClassOf}(\mathit{SubClass}, \exists R.A) \mapsto \mathit{SubClass} \sqsubseteq \exists R.A$$

EquivalentClasses: an axiom $\mathsf{EquivalentClasses}(\mathit{SubClass}_1, \mathit{SubClass}_2)$ is transformed as

$$\mathsf{EquivalentClasses}(\mathit{SubClass}_1, \mathit{SubClass}_2) \mapsto \mathsf{SubClassOf}(\mathit{SubClass}_1, \mathit{SubClass}_2), \mathsf{SubClassOf}(\mathit{SubClass}_2, \mathit{SubClass}_1)$$

DisjointClasses: an axiom $\mathsf{DisjointClasses}(\mathit{SubClass}_1, \mathit{SubClass}_2)$ is transformed as

$$\mathsf{DisjointClasses}(\mathit{SubClass}_1, \mathit{SubClass}_2) \mapsto \mathsf{SubClassOf}(\mathit{SubClass}_1, \neg \mathit{SubClass}_2)$$

Concerning property axioms, we have:

SubObjectPropertyOf: an axiom $\mathsf{SubObjectPropertyOf}(R_1, R_2)$ is transformed as

$$\mathsf{SubObjectPropertyOf}(R_1, R_2) \mapsto R_1 \sqsubseteq R_2$$

EquivalentObjectProperties: an axiom $\mathsf{EquivalentObjectProperties}(R_1, R_2)$ is transformed as

$$\mathsf{EquivalentObjectProperties}(R_1, R_2) \mapsto R_1 \sqsubseteq R_2, R_2 \sqsubseteq R_1$$

DisjointObjectProperties: an axiom $\mathsf{DisjointObjectProperties}(R_1, R_2)$ is transformed as

$$\mathsf{DisjointObjectProperties}(R_1, R_2) \mapsto R_1 \sqsubseteq \neg R_2$$

InverseObjectProperties: an axiom $\mathsf{InverseObjectProperties}(R_1, R_2)$ is transformed as

$$\mathsf{InverseObjectProperties}(R_1, R_2) \mapsto R_2^- \sqsubseteq R_1$$

ObjectPropertyDomain: an axiom $\mathsf{ObjectPropertyDomain}(R, \mathit{SuperClass})$ is transformed as

$$\mathsf{ObjectPropertyDomain}(R, \mathit{SuperClass}) \mapsto \mathsf{SubClassOf}(\exists R, \mathit{SuperClass})$$

ObjectPropertyRange: an axiom $\mathsf{ObjectPropertyRange}(R, \mathit{SuperClass})$ is transformed as

$$\mathsf{ObjectPropertyRange}(R, \mathit{SuperClass}) \mapsto \mathsf{SubClassOf}(\exists R^-, \mathit{SuperClass})$$

ReflexiveObjectProperty and IrreflexiveObjectProperty: cannot be mapped into DL-Lite$_R$ and needs special treatment, as pointed out before;

SymmetricObjectProperty: an axiom $\mathsf{SymmetricObjectProperty}(R)$ is transformed as

$$\mathsf{SymmetricObjectProperty}(R) \mapsto R \sqsubseteq R^-$$

AsymmetricObjectProperty: an axiom $\mathsf{AsymmetricObjectProperty}(R)$ is transformed as

$$\mathsf{AsymmetricObjectProperty}(R) \mapsto R \sqsubseteq \neg R^-$$

On the translation of OWL 2 QL *to our language* We first translate *OWL 2 QL* into DL-Lite$_R$ axioms. Then, we drop all axioms involving atom negation or role negation and then use the transformation

$$\exists R \mapsto R[1] \ \exists R^- \mapsto R[2] \ R^- \mapsto R[2,1].$$

We point out that:

- we do not support atom negation or role negation and, thus, constructs such as disjointness of classes and roles and asymmetry of roles. However, please note that these play a role in KB consistency checking only, but do not play any role in query answering. Therefore, to what concerns query answering, the drop of such axioms is harmless;
- as for DL-Lite$_R$, we do not support role reflexivity. However, we can easily include this by using exactly the same method as described in [10].

Appendix B. Queries of the experiment

The queries used in the experiment are the followings (for illustrative purposes, for some queries we provide also the encoding in our language):

1. Retrieve CV's with knowledge in Engineering Technology:

$$\begin{aligned} q(id, lastName, hasKnowledge, Years) \leftarrow\ & profileLastName(id, lastName), \\ & hasKnowledge(id, classID, Years, Type, Level), \\ & knowledgeName(classID, hasKnowledge), \\ & Engineering\_and\_Technology(classID) \end{aligned}$$

2. Retrieve CV's referred to candidates with degree in Engineering.
3. Retrieve CV's referred to candidates with knowledge in Artificial Intelligence and degree final mark no less than 100/110.
4. Retrieve CV's referred to candidates with knowledge in Artificial Intelligence, degree in Engineering with final mark no less than 100/110.
5. Retrieve CV's referred to candidates experienced in Information Systems (no less than 15 years), with degree final mark no less than 100.
6. Retrieve top-k CV's referred to candidates with knowledge in Artificial Intelligence and degree final mark scored according to $rs(mark; 100, 110)$:

$$\begin{aligned} q(id, lastName, degreeName, mark, hasKnowledge, years) \leftarrow\ & \\ & profileLastName(id, lastName), \\ & hasDegree(id, degreeId, mark), \\ & degreeName(degreeId, degreeName), \\ & hasKnowledge(id, classID, years, type, level), \\ & knowledgeName(classID, hasKnowledge), \\ & Artificial\_Intelligence(classID), \\ & OrderBy(s = rs(mark; 100, 110)) \end{aligned}$$

7. Retrieve CV's referred to candidates with degree in Engineering and final mark scored according to $rs(mark;100,110)$.
8. Retrieve top-k CV's referred to candidates with knowledge in Artificial Intelligence, degree in Engineering with final mark scored according to $rs(mark;100,110)$.
9. Retrieve CV's referred to candidates with knowledge in Information Systems, degree in Engineering and with degree final mark and years of experience both scored according to $rs(mark;100,110) \cdot 0.4 + rs(years;15,25) \cdot 0.6$.
10. Retrieve CV's referred to candidates with good knowledge in Artificial Intelligence, and with degree final mark, years and level of experience scored according to

$$rs(mark; 100, 110) \cdot 0.4 + rs(years; 15, 25) \cdot pref(level; Good/0.6, Excellent/1.0) \cdot 0.6;$$

$$\begin{aligned} q(id, lastName, degreeName, mark, hasKnowledge, years, kType) \leftarrow\ & \\ & profileLastName(id, lastName), \\ & hasDegree(id, degreeId, mark), \\ & degreeName(degreeId, degreeName), \\ & hasKnowledge(id, classID, years, type, level), \\ & knowledgeLevelName(level, kType), \\ & Good(level), \\ & knowledgeName(classID, hasKnowledge), \\ & Artificial\_Intelligence(classID), \\ & OrderBy(s = rs(mark; 100, 110) \cdot 0.4 + rs(years; 15, 25) \cdot pref(level; Good/0.6, Excellent/1.0) \cdot 0.6) \end{aligned}$$

11. Retrieve CV's referred to candidates with knowledge in Artificial Intelligence, grouped by

$$MAX[rs(mark; 100, 110) \cdot 0.4 + rs(years; 15, 25) \cdot 0.6]$$

$$q(id, lastName) \leftarrow profileLastName(id, lastName),$$
$$hasDegree(id, degreeId, mark),$$
$$hasKnowledge(id, classID, years, type, level),$$
$$Artificial\_Intelligence(classID),$$
$$GroupBy(id, lastname),$$
$$OrderBy(s = MAX[rs(mark; 100, 110) \cdot 0.4 + rs(years; 15, 25) \cdot 0.6])$$

12. Retrieve CV's referred to candidates with knowledge in Artificial Intelligence, a degree in Engineering and grouped by

$$AVG(rs(mark; 100, 110) \cdot 0.4 + rs(years; 15, 25) \cdot 0.6)$$

Queries 1–5 are crisp. As each answer has score $\top$, we would like to verify whether there is a retrieval time difference between retrieving all records, or just the k answers. The other queries are top-k queries. In query 9, we show an example of score combination, with a preference on the number of years of experience over the degree's mark, but scores are summed up. In query 10, we use the preference scoring function $pref(level; Good/0.6, Excellent/1.0)$ that returns 0.6 if the level is good, while returns 1.0 if the level is excellent. In this way we want to privilege those with an excellent knowledge level over those with a good level of knowledge. Queries 11 and 12 use ranking aggregates, which were not supported at the time of the experiments in [30,31].

References

[1] SoftFacts: System home page. <http://www.straccia.info/software/SoftFacts/SoftFacts.html>.
[2] W3C: OWL 2 Web Ontology Language document overview. <http://www.w3.org/TR/2009/REC-owl2-overview-20091027/>.
[3] Franz Baader, Sebastian Brandt, Carsten Lutz, Pushing the $\mathcal{EL}$ envelope, in: Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence IJCAI-05, Morgan-Kaufmann Publishers, Edinburgh, UK, 2005, pp. 364–369.
[4] Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, Peter F. Patel-Schneider (Eds.), The Description Logic Handbook: Theory, Implementation, and Applications, Cambridge University Press, 2003.
[5] Patrick. Bosc, Olivier Pivert, SQLf: a relational database language for fuzzy querying, EEE Transactions on Fuzzy Systems 3 (1) (1995) 1–17.
[6] Andrea Calì, Domenico Lembo, Riccardo Rosati, Query rewriting and answering under constraints in data integration systems, in: Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI-03), 2003, pp. 16–21.
[7] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, Riccardo Rosati, DL-Lite: tractable description logics for ontologies, in: Proceedings of the 20th National Conference on Artificial Intelligence (AAAI 2005), 2005.
[8] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, Riccardo Rosati, Data complexity of query answering in description logics, in: Proceedings of the Tenth International Conference on Principles of Knowledge Representation and Reasoning (KR-06), 2006, pp. 260–270.
[9] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, Antonella Poggi, Riccardo Rosati, Marco Ruzzi, Data integration through DL-Lite$_A$ ontologies, in: Semantics in Data and Knowledge Bases, Third International Workshop, SDKB 2008, Nantes, France, March 29, 2008, Revised Selected Papers, Lecture Notes in Computer Science, vol. 4925, Springer Verlag, 2008, pp. 26–47.
[10] Claudio Corona, Marco Ruzzi, Domenico Fabio Savo, Filling the gap between OWL 2 QL and QuOnto: Rowlkit, in: Proceedings of the 22nd International Workshop on Description Logics (DL-09), Oxford, UK, July 27–30, 2009, EUR Workshop Proceedings, vol. 477, 2009.
[11] Ronald Fagin, Amnon Lotem, Moni Naor, Optimal aggregation algorithms for middleware, in: Symposium on Principles of Database Systems, 2001.
[12] Ian Horrocks, Lei Li, Daniele Turi, Sean Bechhofer, The instance store: DL reasoning with large numbers of individuals, in: Proceedings of the 2004 Description Logic Workshop (DL 2004), 2004, pp. 31–40.
[13] Ullrich Hustadt, Boris Motik, Ulrike Sattler, Data complexity of reasoning in very expressive description logics, in: Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI-05), Professional Book Center, 2005, pp. 466–471.
[14] Ihab F. Ilyas, George Beskales, Mohamed A. Soliman, A survey of top-k query processing techniques in relational database systems, ACM Computing Surveys 40 (4) (2008) 1–58.
[15] Thomas Lukasiewicz, Umberto Straccia, Top-k retrieval in description logic programs under vagueness for the semantic web, in: Proceedings of the 1st International Conference on Scalable Uncertainty Management (SUM-07), Lecture Notes in Computer Science, vol. 4772, Springer-Verlag, 2007, pp. 16–30.
[16] Jeff Z. Pan, Giorgos Stamou, Giorgos Stoilos, Edward Thomas, Stuart Taylor, Scalable querying service over fuzzy ontologies, in: Proceedings of the International World Wide Web Conference (WWW 08), Beijing, 2008.
[17] Héctor Pérez-Urbina, Boris Motik, Ian Horrocks, Tractable query answering and rewriting under description logic constraints, Journal of Applied Logic 8 (2) (2009) 186–209.
[18] Antonella Poggi, Domenico Lembo, Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, Riccardo Rosati, Linking data to ontologies, Journal of Data Semantics 10 (2008) 133–173.
[19] Azzurra Ragone, Umberto Straccia, Fernando Bobillo, Tommaso Di Noia, Eugenio Di Sciascio, Fuzzy bilateral matchmaking in e-marketplaces, in: 12th International Conference on Knowledge-Based & Intelligent Information & Engineering Systems – KES2008, Lecture Notes in Artificial Intelligence, vol. 5179, Springer, 2008, pp. 293–301.
[20] Azzurra Ragone, Umberto Straccia, Tommaso Di Noia, Eugenio Di Sciascio, Francesco M. Donini, Vague knowledge bases for matchmaking in p2p e-marketplaces, in: 4th European Semantic Web Conference (ESWC-07), Lecture Notes in Computer Science, vol. 4519, Springer-Verlag, 2007, pp. 414–428.
[21] Azzurra Ragone, Umberto Straccia, Tommaso Di Noia, Eugenio Di Sciascio, Francesco M. Donini, Fuzzy matchmaking in e-marketplaces of peer entities using Datalog, Fuzzy Sets and Systems 160 (2) (2009) 251–268.
[22] Umberto Straccia, Query answering in normal logic programs under uncertainty, in: 8th European Conferences on Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU-05), Lecture Notes in Computer Science, vol. 3571, Springer Verlag, Barcelona, Spain, 2005, pp. 687–700.
[23] Umberto Straccia, Uncertainty management in logic programming: simple and effective top-down query answering, in: Rajiv Khosla, Robert J. Howlett, Lakhmi C. Jain (Eds.), 9th International Conference on Knowledge-Based & Intelligent Information & Engineering Systems (KES-05), Part II, Lecture Notes in Computer Science, vol. 3682, Springer Verlag, Melbourne, Australia, 2005, pp. 753–760.
[24] Umberto Straccia, Answering vague queries in fuzzy DL-Lite, in: Proceedings of the 11th International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems, (IPMU-06), E.D.K., Paris, 2006, pp. 2238–2245.

[25] Umberto Straccia, Towards top-k query answering in deductive databases, in: Proceedings of the 2006 IEEE International Conference on Systems, Man and Cybernetics (SMC-06), IEEE, 2006, pp. 4873–4879.
[26] Umberto Straccia, Towards top-k query answering in description logics: the case of DL-Lite, in: Proceedings of the 10th European Conference on Logics in Artificial Intelligence (JELIA-06), Lecture Notes in Computer Science, vol. 4160, Springer Verlag, Liverpool, UK, 2006, pp. 439–451.
[27] Umberto Straccia, Towards vague query answering in logic programming for logic-based information retrieval, in: World Congress of the International Fuzzy Systems Association (IFSA-07), Lecture Notes in Computer Science, vol. 4529, Springer Verlag, Cancun, Mexico, 2007, pp. 125–134.
[28] Umberto Straccia, Fuzzy description logic programs, in: C. Marsala B. Bouchon-Meunier, R.R. Yager, M. Rifqi (Eds.), Uncertainty and Intelligent Information Systems, World Scientific, 2008, pp. 405–418 (Chapter 29).
[29] Umberto Straccia, Managing uncertainty and vagueness in description logics, logic programs and description logic programs, in: Reasoning Web, 4th International Summer School, Tutorial Lectures, Lecture Notes in Computer Science, vol. 5224, Springer Verlag, 2008, pp. 54–103.
[30] Umberto Straccia, Eufemia Tinelli, Tommaso Di Noia, Eugenio Di Sciascio, Simona Colucci, Semantic-based top-k retrieval for competence management, in: J. Rauch et al. (Eds.), Proceedings of the 18th International Symposium on Methodologies for Intelligent Systems (ISMIS-09), Lecture Notes in Artificial Intelligence, vol. 5722, Springer Verlag, 2009, pp. 473–482.
[31] Umberto Straccia, Eufemia Tinelli, Tommaso Di Noia, Eugenio Di Sciascio, Simona Colucci, Top-k retrieval for automated human resource management, in: Proceedings of the 17th Italian Symposium on Advanced Database Systems (SEBD-09), 2009, pp. 161–168.
[32] Umberto Straccia, An ontology mediated multimedia information retrieval system, in: Proceedings of the 40th International Symposium on Multiple-Valued Logic (ISMVL-10), IEEE Computer Society, 2007, pp. 319–324.
[33] Umberto Straccia, SoftFacts: a top-k retrieval engine for ontology mediated access to relational databases, in: Proceedings of the 2010 IEEE International Conference on Systems, Man and Cybernetics (SMC-10), IEEE Press, 2010, pp. 4115–4122.
[34] Jeffrey D. Ullman, Principles of Database and Knowledge Base Systems, vols. 1 and 2, Computer Science Press, Potomac, Maryland, 1989.
[35] Moshe Vardi, The complexity of relational query languages, in: Proceedings of the 14th ACM SIGACT Symposium on Theory of Computing (STOC-82), 1982, pp. 137–146.
[36] Peter Vojtás, Fuzzy logic aggregation for semantic web search for the best (top-k) answer, in: Elie Sanchez (Ed.), Fuzzy Logic and the Semantic Web, Capturing Intelligence, Elsevier, 2006, pp. 341–359 (Chapter 17).
[37] Maria Zemankova, Abraham Kandel, Implementing imprecision in information systems, Information Science 37 (1–3) (1985) 107–141.