SoftFacts: A Top-k Retrieval Engine for a Tractable Description Logic Accessing Relational Databases

Umberto Straccia
ISTI-CNR, Via G. Moruzzi 1, I-56124 Pisa, Italy
straccia@isti.cnr.it
TR: ISTI-CNR TueJan200916441

January 20, 2009

Abstract

We outline SoftFacts, an ontology mediated top-k information retrieval system over relational databases. 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, while facts are stored into a relational database. The results of a query may be ranked according to some scoring function. We will illustrate its logical model, its architecture, its representation and query language, the reasoning algorithms and the experiments we conducted.

1 Introduction

Description Logics (DLs) [2] 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 DL* [16]. 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 that is exponential on the intentional part, it is mandatory that reasoning is polynomial in the data size, *i.e.* in *data complexity* [31]. 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 [1, 5, 8, 10, 15, 17].

In this paper, we describe the salient features of the SoftFacts system [1], whose aim is to allow an ontology mediated access to relational databases for data intensive applications. Informally, data is stored into a database and a DL is used to define the

[1] See, `http://www.straccia.info/software/SoftFacts/SoftFacts.html`

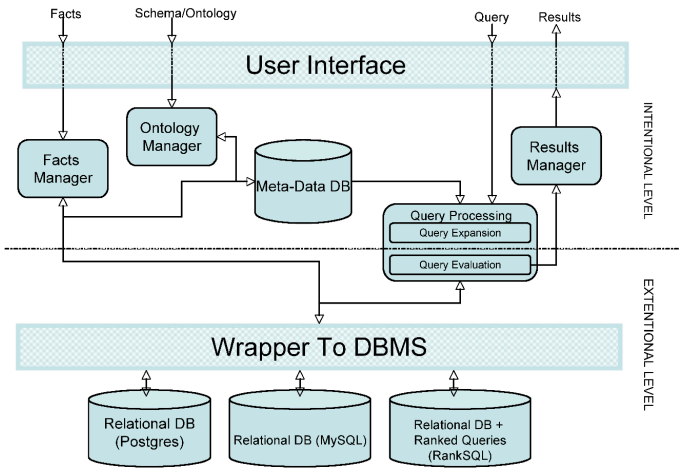

Figure 1: Architecture.

relevant abstract concepts and relations of the application domain. Main features of the SoftFacts system are:

1. The SoftFacts ontology language belongs to the family of DLR-Lite [6]. DLR-Lite is different from usual DLs as it supports n-ary relations ($n \geqslant 1$), whereas DLs support usual unary relations (called *concepts*) and binary relations (called *roles*);

2. A SoftFacts query is the union of conjunctive queries;

3. The results of a query may be ranked according to some scoring function. In fact, SoftFacts supports *Top-k Query Answering* [23, 27, 25, 26, 28], (find top-k scored tuples satisfying query), *e.g.* "find cheap hotels close to the train station", where cheap and price are an user defined function of the distance and price, respectively.

In the following, we will illustrate SoftFacts's logical model, its architecture, its representation and query language, the reasoning algorithms and the experiments we conducted.

2 The architecture

The SoftFacts architecture has two basic components: the DL-based ontology component and the database component (see Figure 1).

The DL-component supports both the definition of the ontology and query answering. In particular, it provides a logical query and representation language, which is an extension of the DL language DLR-Lite [6, 25, 27, 29] and support ranking queries. Concerning the database component, SoftFacts supports access to various different

database systems. The access to these systems is transparent as managed by an appropriate wrapper.

Operationally, a user submits a conceptual query (a Datalog like logic-based conjunctive query) by means of the DL-component. The DL-component will then use the ontology to *reformulate* the initially query into one or several queries (query expansion) [2]. These queries are then translated and submitted to the underlying database system (using the wrapper). The database system then provides back the top-k answers for each of the issued queries. The ranked lists will then be merged into one final result list and displayed to the user (using the the query evaluation module).

3 The query and representation language

For computational reasons, the particular logic SoftFacts adopts is based on an extension of the DLR-Lite [6] Description Logic (DL) [2] without negation. The 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. The SoftFacts logic extends DLR-Lite by enriching it with build-in predicates. Conjunctive queries are enriched with scoring functions that allow to rank and retrieve the top-k answers.

To start with, as *e.g.*, answers to a query are scored according to some scoring function, tuples may have associated a score. We will call the score also truth degree, or simply degree. Hence we have to fix a truth-space. We will consider as truth-space the set $[0, \top]$, where $\top > 0$ is a sufficiently large number (maximal machine representable number). 0 denotes false, while $\top$ indicates true [3].

A *knowledge base* $\mathcal{K} = \langle \mathcal{F}, \mathcal{O}, \mathcal{A} \rangle$ consists of a *facts component* $\mathcal{F}$, an *Ontology component* (also called, *DL component*) $\mathcal{O}$ and an *abstraction component* $\mathcal{A}$, which are defined below.

Facts Component. SoftFacts allows 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 of truth (or *score*) in $[0, \top]$ indicating to which extent the tuple $\langle c_1, \ldots, c_n \rangle$ is an instance of relation R. For instance, given a relation $\mathsf{SportsCar}$ with signature $\mathsf{SportsCar}(\mathsf{carID}, \mathsf{Name}, \mathsf{Speed})$,

$$\mathsf{SportsCar}(2, \mathsf{AudiTT}, 210)[0.85] \, .$$

is a fact, dictating to which extent the tuple $\langle 2, \mathsf{AudiTT}, 210 \rangle$ is an instance of $\mathsf{SportsCar}$ (hence, to which extent the Audi TT is a sports car).

[2] Cyclic definitions may be present in the DL-component as well.

[3] One might wonder why not necessary $\top = 1$. This is due to the fact that we may use aggregation functions (*e.g.*, the sum) that yield a score above 1.0.

Profile										
profID	FirstName	LastName	Genre	BirthDate	CityOfBirth	Address	City	ZipCode	Country	...
2	Wayne	Hernandez	female	1979-10-04	Berlin	Via Volta	Terni	05100	Italy	...
34	Hillary 156	Gadducci	female	1978-01-27	Bangalore	Church ST	New York	10027	USA	...
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Figure 2: The profile table.

For each 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 score). Each table is sorted in descending order with respect to the scores. For ease, we may omit the score component and in such cases the value 1 is assumed.

Example 1 *Suppose we have Curricula Vitæ. Some basic information is stored into the* Profile *relation and an excerpt is shown in Fig. 2. Here, the score is implicitly assumed to be* 1 *in each record. For instance,*

$$\mathsf{Profile}(2, \mathsf{Wayne}, \mathsf{Hernandez}, \mathsf{female}, 1979-10-04, \mathsf{Berlin}, \mathsf{ViaVolta}, \mathsf{Terni}, 05100, \mathsf{Italy}, \ldots)\ ,$$

corresponds to the fact related to the first record. □

Ontology Component. The ontology component is used to define the relevant abstract concepts and relations of the application domain by means of axioms.

But, before we address the syntax of axioms, let us introduce the notion of concrete domains. In fact, SoftFacts supports concrete domains with specific predicates on it. The *concrete predicates* that SoftFacts allows are relational predicates such as $([i] \leqslant 1500)$ (*e.g.* the value of the i-th column is less or equal than 1500) and $([i] = \text{“}Mayer''\text{)}$ (*e.g.* the value of the i-th column is equal to the string “Mayer”). Formally, a *concrete domain* in SoftFacts is a pair $\langle \Delta_D, \Phi_D\rangle$, where Δ_D is an interpretation domain and Φ_D is the set of *domain predicates* d with a predefined arity n and an interpretation $d^D \colon \Delta_D^n \to \{0, 1\}$.

Now, SoftFacts allows to specify an ontology by relying on axioms. Consider an alphabet of n-ary relation symbols (denoted R), *e.g.* Profiles as described in Table 2, and an alphabet of unary relations, called *atomic concepts* (and denoted A), *e.g.* ItalianCity. Now, the DL component $\mathcal{O}$ is a finite set of *axioms* having the form

$$(Rl_1 \sqcap \ldots \sqcap Rl_m \sqsubseteq Rr)[n]$$

where $m \geqslant 1$, 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*, and n is a truth degree assigning a “weight” to the axiom. For ease, we may omit the value n and in such cases the value $n = 1$ is assumed. As illustrative purpose, a simple ontology axiom may be of the form

$$\mathsf{ItalianCity} \sqsubseteq \mathsf{EuropeanCity}$$

with informal reading “any italian city is an european city”. Here ItalianCity and EuropeanCity are unary relations with signature ItalianCity(id) and EuropeanCity(id),

respectively. Similarly, the ontology axiom

$$\mathsf{ItalianCity} \sqcap \mathsf{BigCity} \sqsubseteq \mathsf{BigEuropeanCity}$$

has informal reading "any italian city, which is also big, is a big european city" (here BigCity and BigEuropeanCity are again unary relations with signature $\mathsf{BigCity}(\mathsf{id})$ and $\mathsf{BigEuropeanCity}(\mathsf{id})$, respectively).

We may also involve n-ary realtions in ontology axioms. For instance, suppose that from the profiles records, we would like to extract just the profile ID and the last name, and call this new relation HasLastName with signature $\mathsf{HasLastName}(\mathsf{profID}, \mathsf{LastName})$. In database terminology this amounts in a projection of the Profile relation on the first and third column. In our language, the projection of an n-ary relation R on the columns $i_1, \ldots, i_k$ ($1 \leqslant i_1, i_2, \ldots, i_k \leqslant n$, $1 \leqslant i \leqslant n$), will be indicated with $\exists[i_1, \ldots, i_k]R$. Hence, *e.g.*

$$\exists[1, 3]\mathsf{Profile}$$

is the binary relation that is the projection on the first and third column of the Profile relation. So, for instance, the axioms

$$\exists[1, 3]\mathsf{Profile} \sqsubseteq \exists[1, 2]\mathsf{HasLastName}$$

$$\exists[1, 4]\mathsf{Profile} \sqsubseteq \exists[1, 2]\mathsf{HasGenre}$$

$$\exists[1, 2]\mathsf{Profile} \sqsubseteq \exists[1, 2]\mathsf{HasFirstName}$$

state, *e.g.*, that the relation HasLastName contains the projection of the Profile relation on the first and third column (the other axioms are interpreted similarly).

In case of a projection, we may further restrict it according to some conditions, using concrete predicates. For instance,

$$\exists[1, 5]\mathsf{Profile}.(([5] \geqslant 1979))$$

corresponds to the set of tuples $\langle \mathsf{profID}, \mathsf{BirthDate} \rangle$ such that the fifth column of the relation Profile, *i.e.* the person's birth date, is equal or greater than 1979. Other examples of axioms are

$$\exists[1, 5]\mathsf{Profile} \sqsubseteq \exists[1, 2]\mathsf{hasBirthDate}$$

$$\exists[1, 4]\mathsf{Profile} \sqsubseteq \exists[1, 2]\mathsf{hasSex}$$

$$\exists[3, 2, 6]\mathsf{Profile}.(([5] \leqslant 1991) \sqcap ([4] = \mathsf{male})) \sqsubseteq \exists[1, 2, 3]\mathsf{AdultMalePerson}$$

Note that in the last axiom, we have multiple conditions: we require that the age is greater or equal than 18 (w.r.t. year 2009) and the gender is male. This axiom defines the relation $\mathsf{AdultMalePerson}(\mathsf{LastName}, \mathsf{FirstName}, \mathsf{CityOfBirth})$.

Another feature of axioms is that a degree $n \in [0, 1]$ may be attached to the inclusion $Rl_1 \sqcap \ldots \sqcap Rl_m \sqsubseteq Rr$. For instance, the degree n in a inclusion axiom may be

computed automatically by an ontology alignment tool [11], indicating to what extend a concept/relation of an ontology $\mathcal{O}_1$ has the same meaning of another concept/relation of ontology $\mathcal{O}_2$. Examples of such axioms, called ontology mappings, may be

$$(\mathsf{SportyCar} \sqsubseteq \mathsf{SportsCar})[0.97]$$
$$(\exists[1,2]\mathsf{HasInvoice} \sqsubseteq \exists[1,2]\mathsf{HasPrice})[0.64] \ ,$$

where the signatures of the atomic concepts and the relations are $\mathsf{SportyCar}(\mathsf{CarID})$, $\mathsf{SportsCar}(\mathsf{CarID})$, $\mathsf{HasInvoice}(\mathsf{CarID}, \mathsf{Price})$ and $\mathsf{HasPrice}(\mathsf{CarID}, \mathsf{Price})$, respectively. Informally, the former axiom states that the concept $\mathsf{SportyCar}$ of ontology $\mathcal{O}_1$ has the same meaning of the concept $\mathsf{SportsCar}$ of ontology $\mathcal{O}_2$ with degree 0.97 (the informal meaning of the latter axiom is similar).

In general and informally, an axiom $(Rl_1 \sqcap \ldots \sqcap Rl_m \sqsubseteq Rr)[n]$ states that if $\mathbf{c}$ is an instance of Rl_i to degree s_i, then $\mathbf{c}$ is an instance of Rr to degree at least $n \otimes s_1 \otimes \ldots \otimes s_l$, where $\otimes$ is a so-called *t-norm* that combines the score/truth of the "conjunctive" expression in the left-hand side of the axiom [4] (see [14]). Some typical t-norms are

$x \otimes y$	$=$	$\min(x, y)$	Gödel conjunction
$x \otimes y$	$=$	$\max(x + y - 1, 0)$	Łukasiewicz conjunction
$x \otimes y$	$=$	$x \cdot y$	Product conjunction .

So, for instance, if an Audi TT has been classified as a $\mathsf{SportyCar}$ to degree 0.85, *i.e.* we have the fact $\mathsf{SportyCar}(\mathsf{AudiTT})[0.85]$, and we ask about the instances of $\mathsf{SportsCar}$, then we may retrieve the Audi TT with degree $0.97 \otimes 0.85 = 0.82$ (if we use product t-norm).

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

$$\begin{array}{lcl}
Rr & \longrightarrow & A \mid \exists[i_1, \ldots, i_k]R \\
\\
Rl & \longrightarrow & A \mid \exists[i_1, \ldots, i_k]R \mid \\
 & & \exists[i_1, \ldots, i_k]R.(Cond_1 \sqcap \ldots \sqcap Cond_h) \\
\\
Cond & \longrightarrow & ([i] \leqslant v) \mid ([i] < v) \mid ([i] \geqslant v) \mid ([i] > v) \mid \\
 & & ([i] = v) \mid ([i] \neq v)
\end{array}$$

where A is an atomic concept, R is an n-ary relation with $1 \leqslant i_1, i_2, \ldots, i_k \leqslant n$, $1 \leqslant i \leqslant n$ and v is a value of the concrete interpretation domain of the appropriate type.

Here $\exists[i_1, \ldots, i_k]R$ is the projection of the relation R on the columns $i_1, \ldots, i_k$ (the order of the indexes matters). Hence, $\exists[i_1, \ldots, i_k]R$ has arity k. On the other hand, $\exists[i_1, \ldots, i_k]R.(Cond_1 \sqcap \ldots \sqcap Cond_l)$ further restricts the projection $\exists[i_1, \ldots, i_k]R$ 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.

[4] Given truth degrees x and y, the conjunction of x and y is $x \otimes y$. $\otimes$ has to be symmetric, associative, monotone in its arguments and such that $x \otimes 1 = x$.

$$\begin{aligned}
\mathsf{Legislators} &\sqsubseteq \mathsf{Legal_Support_Workers} && (1)\\
\mathsf{Auditors} &\sqsubseteq \mathsf{Accountants_and_Auditors} && (2)\\
\mathsf{Bakers} &\sqsubseteq \mathsf{Food_Processing_Workers} && (3)\\
\exists[2]\mathsf{knowsLanguage} &\sqsubseteq \mathsf{Language} && (4)\\
\exists[2]\mathsf{hasDegree} &\sqsubseteq \mathsf{Degree} && (5)
\end{aligned}$$

Figure 3: Excerpt of a CV ontology.

The syntax is inspired by the description logic *DLR-Lite* [6], a LogSpace data complexity family of DL languages, but still with good representation capabilities. We recall that despite the simplicity of its language, the DL component is able to capture the main notions (though not all, obviously) to represent structured knowledge. For instance, the axioms allow us to specify *subsumption*, concept A_1 is subsumed by concept A_2, using $A_1 \sqsubseteq A_2$; *typing*, using $\exists[i]R \sqsubseteq A$ (the i-th column of R is of type A); and *participation constraints*, using $A \sqsubseteq \exists[i]R$ (all instance of A occur in the projection of R on the i-th column).

Example 2 (Example 1 cont.) *Consider again Example 1. 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 a relation* $\mathsf{HasDegree}(\mathsf{profID}, \mathsf{degID}, \mathsf{Marks})$ *and an atomic concept* $\mathsf{Degree}(\mathsf{degID})$*. For instance, axiom* (4) *states that the languages known by profile* profID *should be languages.* □

Finally, we assume that relations occurring in $\mathcal{F}$ do not occur in axioms (so, we do not allow that database relation names occur in $\mathcal{O}$.

Abstraction Component. The abstraction component is a set of "abstraction 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. As illustrative purpose, assume that we have a relation Jobs in a database with signature $\mathsf{Jobs}(\mathsf{jobID}, \mathsf{Name}[\mathsf{string}])$, where the first column is of type int, while the second is of type String. Then, an example of abstraction statement is

$$\mathsf{jobName} \mapsto \mathsf{Jobs}(\mathsf{jobID}[\mathsf{int}], \mathsf{Name}[\mathsf{string}]) \ ,$$

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. Another example of abstraction statement is

$$\mathsf{CV} \mapsto \mathsf{Profile}(\mathsf{profID}[\mathsf{int}]) \ ,$$

declaring that CV is an atomic concept, whose instances are the projection on the profID column of the Profile relation of the database. Note that the arity of Profile is greater than one.

The exact syntax is defined as follows. We have two variants: the former one is a simple version, called simple abstraction statement, while the latter is more general, called complex abstraction statement and are similar to the one presented in [7].

Let R_1 be a relation symbol and let R_2 be an m-ary table in the database. Let $c_1, \ldots, c_n$ be $n \leqslant m$ column names of relation R_2 each of which of type t_i. We also assume that the score of an instance of R_2 is stored in the column c_{score}. Then a *simple abstraction statement* is of the form

$$R_1 \mapsto R_2(c_1[t_1], \ldots, c_n[t_n])[c_{score}] \ .$$

stating that R_1 is an n-ary relation of the ontology component, that is mapped into the projection on columns $c_1 \ldots, c_n$ of relation R_2. The score of these tuples is provided by column c_{score} of relation R_2. The score column c_{score} may be omitted and in that case the score 1 is assumed for the tuples. We assume that R_1 occurs in $\mathcal{O}$, while R_2 occurs in $\mathcal{F}$.

On the other hand, a *complex abstraction statement* is of the form

$$R_1 \mapsto (t_1, \ldots, t_n)[c_{score}].\ sql \ ,$$

where sql is an SQL statement returning n-ary tuples of type $\langle t_1, \ldots, t_n \rangle$ with score determined by the c_{score} column. The tuples are ranked in descreasing 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 column c_{score} may be omitted and in that case the score 1 is assumed for the tuples. We assume that R_1 occurs in $\mathcal{O}$, while all of the relational tables occurring in the SQL statement occur in $\mathcal{F}$. An example of complex abstraction statement is

$$\begin{aligned} \mathsf{BigCity} \mapsto (id)[score].(&\mathsf{SELECT} \\ &id, min(1, size/10^6)\ \mathsf{AS}\ score \\ &\mathsf{FROM}\ CityTable \\ &\mathsf{ORDER\ BY}\ score\ ) \ . \end{aligned}$$

In the above case, we defined the abstract relation BigCity as the set of city IDs, where the score of being big is determined by $min(1, size/10^6)$, 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.

Note that a simple abstraction statement can be expressed as a complex abstraction statement of the form

$$R_1 \mapsto (t_1, \ldots, t_n)[c_{score}].(\mathsf{SELECT}\ c_1, \ldots, c_n, c_{score}\ \mathsf{FROM}\ R_2) \ .$$

Finally, we assume that there is at most one abstract statement for each abstract relational symbol R_1 and we will assume that a SQL statement is considered as an $n+1$-ary concrete predicate with obvious semantics.

Query language. Concerning queries, a SoftFacts *query* consists of a "conjunctive query", with a scoring function to rank the answers.

Before we give the formal syntax, we will provide some examples, with rising complexity, with their informal meaning.

A simple query form is

$$q(\mathbf{x})[s] \quad \leftarrow \exists \mathbf{y} \quad R_1(\mathbf{z}_1)[s_1], \ldots, R_l(\mathbf{z}_l)[s_l], \\ \mathsf{OrderBy}(s = s_1 \otimes \ldots \otimes s_l)$$

where q is an n-ary predicate, every R_i is an n_i-ary predicate, $\mathbf{x}$ is a vector of variables, and every $\mathbf{z_i}$ is a vector of constants, or variables; $\mathbf{x}$ are the *distinguished variables*; $\mathbf{y}$ are existentially quantified variables called the *non-distinguished variables*; $\mathbf{z_i}$ are tuples of constants or variables in $\mathbf{x}$ or $\mathbf{y}$. We omit to write $\exists \mathbf{y}$ when $\mathbf{y}$ is clear from the context. $s, s_1, \ldots, s_l$ are distinct variables and different from those in $\mathbf{x}$ and $\mathbf{y}$. $R_i(\mathbf{z_i})$ 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 concrete domain. 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* and $\mathsf{OrderBy}(s = s_1 \otimes \ldots \otimes s_l)$ the *scoring atom*.

The informal meaning of such a query is: if $\mathbf{z}_i$ is an instance of R_i to degree s_i, then $\mathbf{x}$ is an instance of q to degree at least or equal to $s_1 \otimes \ldots \otimes s_l$. We also allow the scores $[s], [s_1], \ldots, [s_l]$ and the scoring atom to be omitted. In this case we assume the value 1 instead.

Example queries are:

$\mathsf{q(x)} \leftarrow \mathsf{SportsCar(x)}$
// 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

So far, we used $\otimes$ as scoring combination function. We may further generalize queries, by allowing the form

$$q(\mathbf{x})[s] \quad \leftarrow \exists \mathbf{y} \quad R_1(\mathbf{z}_1)[s_1], \ldots, R_l(\mathbf{z}_l)[s_l], \\ \mathsf{OrderBy}(s = f(s_1, \ldots, s_l))$$

where f is a *scoring* function $f\colon ([0, \top])^l \to [0, \top]$, which combines the scores s_i of the l relations $R_i(\mathbf{c}'_i)$ into an overall *score* s to be assigned to the rule head $q(\mathbf{c})$. We assume that f is *monotone*, that is, for each $\mathbf{v}, \mathbf{v}' \in ([0, \top])^l$ such that $\mathbf{v} \leqslant \mathbf{v}'$, it holds $f(\mathbf{v}) \leqslant f(\mathbf{v}')$, where $(v_1, \ldots, v_l) \leqslant (v'_1, \ldots, v'_l)$ iff $v_i \leqslant v'_i$ for all i. An example query of this form is:

$$\mathsf{q(x)[s]} \quad \leftarrow \quad \mathsf{SportsCar(x)[s_1]}, \mathsf{Cheap(x)[s_2]}, \\ \mathsf{OrderBy(s = 0.7 \cdot s_1 + 0.3 \cdot s_2)}$$

// find cheap sports cars

Here, we may assume to have database tables $\mathsf{WonderCars}(\mathsf{carID}, \mathsf{carName}, \mathsf{speed}, \mathsf{score})$ and $\mathsf{Cars}(\mathsf{ID}, \mathsf{price}, \mathsf{cheapdegree})$, with abstraction statements

$$\begin{aligned} \mathsf{SportsCar} &\mapsto \mathsf{WonderCars}(\mathsf{carID}[\mathsf{int}])[\mathsf{score}] \\ \mathsf{Cheap} &\mapsto \mathsf{Cars}(\mathsf{ID}[\mathsf{int}])[\mathsf{cheapdegree}] \ . \end{aligned}$$

Therefore, SportsCar and Cheap are atomic concepts, whose instances are gathered from the tables WonderCars and Cars, and whose score are in the score and cheapdegree column, respectively. Eventually, the answers to the above query are ranked according to the score computed as a linear combination of the score of being a sport car and cheap.

In the previous example, we also assume that there is some fixed procedure that for each car, computes its degree of cheapness as a function of the price and stores the result in the Cars table. As next, we would like this procedure to be also allowed to be user-defined in the sense that a function computing the score occurs in a query. For instance, a user may decide that the degree of cheapness of a car is a function of its price and is determined by

$$\mathsf{mycheap}(\mathsf{price}) = \max(0, 1 - \frac{\mathsf{price}}{12000}) \ .$$

Then he is allowed to write the query

$$\begin{aligned} \mathsf{q}(\mathsf{x})[\mathsf{s}] \quad \leftarrow \quad & \mathsf{SportsCar}(\mathsf{x})[\mathsf{s}_1], \mathsf{hasPrice}(\mathsf{x}, \mathsf{p}), \\ & \mathsf{OrderBy}(\mathsf{s} = 0.7 \cdot \mathsf{s}_1 + 0.3 \cdot \mathsf{mycheap}(\mathsf{p})) \end{aligned}$$

that is,

$$\begin{aligned} \mathsf{q}(\mathsf{x})[\mathsf{s}] \quad \leftarrow \quad & \mathsf{SportsCar}(\mathsf{x})[\mathsf{s}_1], \mathsf{hasPrice}(\mathsf{x}, \mathsf{p}), \\ & \mathsf{OrderBy}(\mathsf{s} = 0.7 \cdot \mathsf{s}_1 + 0.3 \cdot \max(0, 1 - \tfrac{\mathsf{p}}{12000})) \end{aligned}$$

where we consider the abstract mapping

$$\mathsf{hasPrice} \mapsto \mathsf{Cars}(\mathsf{ID}[\mathsf{int}], \mathsf{price}[\mathsf{int}])$$

stating that hasPrice is a binary relation of tuples $\langle \mathsf{ID}, \mathsf{price} \rangle$ each of which having score 1 (as the score component is omitted in the abstract mapping). Now, such queries are of the general form

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

where additionally, p_j is an n_j-ary *fuzzy predicate* assigning to each n_j-ary tuple $\mathbf{c}_j$ a *score* $p_j(\mathbf{c}_j) \in [0, \top]$. Such predicates are called *expensive predicates* in [9] as the score is not pre-computed off-line, but is computed on query execution. We require that an n-ary fuzzy predicate p is *safe*, that is, there is not an m-ary fuzzy predicate p' such that $m < n$ and $p = p'$. Informally, all parameters are needed in the definition of p. Note that concerning fuzzy predicates, we may use the so-called left-shoulder, right-shoulder, triangular and trapezoidal functions (see Fig. 4), which are well known

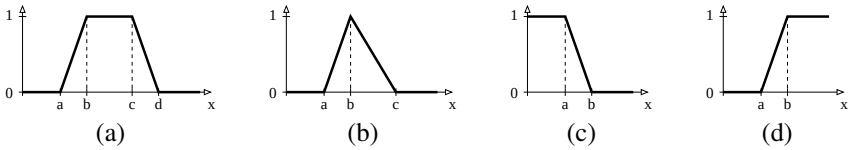

Figure 4: (a) Trapezoidal function $trz(x; a, b, c, d)$, (b) triangular function $tri(x; a, b, c)$, (c) left shoulder function $ls(x; a, b)$, and (d) right shoulder function $rs(x; a, b)$.

HasDegree		
profID	classID	Mark
2	29	107
34	25	104
.	.	.
.	.	.
.	.	.

Degree	
degID	Name
29	Civil_Structural_Engineering
25	Chemical_Engineering
.	.
.	.
.	.

Figure 5: The HasDegree and Degree tables.

fuzzy membership functions in fuzzy set theory. So, for instance, we may write the query

$$\begin{aligned} \mathsf{q(x)[s]} \quad \leftarrow \quad & \mathsf{SportsCar(x)[s_1], hasPrice(x, p),} \\ & \mathsf{OrderBy(s = 0.7 \cdot s_1 + 0.3 \cdot } ls(\mathsf{p}; 10000, 14000)) \end{aligned}$$

Here, $\mathsf{ls}(x; 10000, 14000)$ dictates that we are definitely satisfied if the price is less than 10000, but can pay up to 14000 to a lesser degree of satisfaction.

Example 3 (Example 2 cont.) *Consider Example 2. Assume that we have the two relational tables in Figure 5, which given a profID, gives us the degreeID and the marks obtained by profID and given an degreeID, the degree name. Assume that we have also the abstract mappings*

$$\begin{aligned} \mathsf{hasName} \quad & \mapsto \quad \mathsf{Profile(LastName[string])} \\ \mathsf{hasDegree} \quad & \mapsto \quad \mathsf{HasDegree(profID[int], classID[int])} \\ \mathsf{hasMark} \quad & \mapsto \quad \mathsf{HasDegree(profID[int], Mark[int])} \\ \mathsf{hasDegreeName} \quad & \mapsto \quad \mathsf{Degree(degID[int], Name[string])} \ . \end{aligned}$$

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

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

Then we may have the results

id	name	degree	mark	score
⋮	⋮	⋮	⋮	⋮
2	Hernandez	Civil_Structural_Engineering	107	0.7
⋮	⋮	⋮	⋮	⋮
34	Gadducci	Chemical_Engineering	104	0.4
⋮	⋮	⋮	⋮	⋮

□

Finally, we address our last query language extension, by allowing so-called ranking aggregates to occur in a query [21]. Essentially, ranking aggregates apply usual SQL aggregate functions such as SUM, AVG, MAX, MIN to the scores of group of tuples, which are answers to a query. For instance, by referring to Example 3, a person (*e.g.*, Gadducci) may held more than one degree and we would like to rank a CV with more degrees better than one with just one (*e.g.* we may would like to *sum-up* the scores of all degrees of Gadducci).

Example 4 (Example 3 cont.) *Consider Example 3. Assume that we additionally would like to sum-up the scores of the degrees of each person. Then, such a query may be expressed as*

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

Intuitively, for the above query, we ask to group all tuples according to id *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*, where each tuple has score s_i computes as*

$$\min(\mathsf{CV}(\mathsf{id}), \mathsf{hasName}(\mathsf{id},\mathsf{name}), \mathsf{hasMark}(\mathsf{id},\mathsf{mark}), \mathsf{rs}(\mathsf{mark}; 100, 110)) \ ,$$

then the score s_g of the group g is $\sum_{ti} s_i$. A group g is ranked then according to its score s_g and the top-k ranked groups are returned. □

More formally, let @ be ranking aggregate function with @ $\in \{\mathsf{SUM}, \mathsf{AVG}, \mathsf{MAX}, \mathsf{MIN}\}$ then a query with ranking aggregates is of the form

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

where additionally $\mathbf{w}$ is a list of variables according to which we want to group the tuples and @ is the aggregate function according to which to compute the score of the group. $\mathsf{GroupBy}(\mathbf{w})$ is acalled the *grouping atom*.

In summary, a query in its general form is an expression

$$\begin{array}{lll} q(\mathbf{x})[s] & \leftarrow \exists \mathbf{y} & R_1(\mathbf{z}_1)[s_1], \ldots, R_l(\mathbf{z}_l)[s_l], \\ & & \mathsf{GroupBy}(\mathbf{w}), \\ & & \mathsf{OrderBy}(s = @[f(s_1, \ldots, s_l, p_1(\mathbf{z}'_1), \ldots, p_h(\mathbf{z}'_h)]) \end{array} \tag{6}$$

where

1. q is an n-ary relation, every R_i is an n_i-ary relation,
2. $\mathbf{x}$ are the n *distinguished variables*;
3. s, s_i are distinct *score variables* and neither occurring in $\mathbf{x}$ nor in $\mathbf{y}$. All s_i will take values in $[0, \top]$. s will take value in $[0, \top]$, except for the case of ranking aggregates, where s may be a non-negative real value;
4. $\mathbf{y}$ are so-called *non-distinguished variables* and are distinct from the variables in $\mathbf{x}$;
5. $\mathbf{w}$ are variables in $\mathbf{x}$ or $\mathbf{y}$ such that each variable in $\mathbf{x}$ occurs in $\mathbf{w}$;
6. $\mathbf{z_i}, \mathbf{z_j}'$ are tuples of constants or variables in $\mathbf{x}$ or $\mathbf{y}$. Any variable in $\mathbf{x}$ occurs in some $\mathbf{z_i}$. Any variable in $\mathbf{z_j}'$ occurs in some $\mathbf{z_i}$;
7. p_j is an n_j-ary *fuzzy predicate* assigning to each n_j-ary tuple $\mathbf{c}_j$ a *score* $p_j(\mathbf{c}_j) \in [0, \top]$. Such predicates are called *expensive predicates* in [9] as the score is not pre-computed off-line, but is computed on query execution. We require that an n-ary fuzzy predicate p is *safe*, that is, there is not an m-ary fuzzy 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 $f\colon ([0, \top])^{l+h} \to [0, \top]$, which combines the scores of the l relations $R_i(\mathbf{c}'_i)$ and the h fuzzy predicates $p_j(\mathbf{c}''_j)$ into an overall *score* to be assigned to the rule head $R(\mathbf{c})$. 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;
9. $@$ is a ranking aggregate function with $@ \in \{\mathsf{SUM}, \mathsf{AVG}, \mathsf{MAX}, \mathsf{MIN}\}$;
10. We also assume that the computational cost of f and all fuzzy predicates p_i is bounded by a constant.

Finally, a *disjunctive query* $\mathbf{q}$ is, as usual, a finite set of conjunctive queries in which all the rules have the same head.

Semantics. From a semantics point of view, SoftFacts is based on mathematical fuzzy logic [14]. Given a concrete domain $\langle \Delta_D, \Phi_D \rangle$, an *interpretation* $\mathcal{I} = \langle \Delta, \cdot^{\mathcal{I}} \rangle$ consists of a *fixed infinite domain* Δ, containing Δ_D, and an *interpretation function* $\cdot^{\mathcal{I}}$ that maps

- every atom A to a partial function $A^{\mathcal{I}}\colon \Delta \to [0, 1]$

- maps an n-ary predicate R to a partial function $R^{\mathcal{I}}\colon \Delta^n \to [0,1]$
- constants to elements of Δ such that $a^{\mathcal{I}} \neq b^{\mathcal{I}}$ if $a \neq b$ (unique name assumption).

Intuitively, rather than being an expression (*e.g.* $R(\mathbf{c})$) either true or false in an interpretation, it has a degree of truth in $[0,1]$. So, given a constant c, $A^{\mathcal{I}}(c)$ determines to which degree the individual c is an instance of atom A. Similarly, given an n-tuple of constants $\mathbf{c}$, $R^{\mathcal{I}}(\mathbf{c})$ determines to which degree the tuple $\mathbf{c}$ is an instance of the relation R. 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 formalization of this aspect and leave it at the intuitive level.

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 $\mathbf{c}[i_1,...,i_k]$ 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).

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 concrete comparison predicates, the interpretation function $\cdot^{\mathcal{I}}$ has to satisfy

$$([i] \leqslant v)^{\mathcal{I}}(\mathbf{c}) \quad = \quad \begin{cases} \top & \text{if} \quad \mathbf{c}[i] \leqslant v \\ 0 & \text{otherwise} \end{cases}$$

and similarly for the other comparison constructs, $([i] < v), ([i] \geqslant v), ([i] > v)$ and $([i] = v) \mid ([i] \neq v)$.

Concerning axioms, as in an interpretation each $Rl_i(\mathbf{c})$ has a degree of truth, we have to specify how to combine them to determine the degree of truth of the conjunction $Rl_1 \sqcap \ldots \sqcap Rl_m$. Usually, in mathematical fuzzy logic one uses a so-called T-norm $\otimes$ to combine the truth of "conjunctive" expressions (see [14]).

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

$$(\exists[i_1,\ldots,i_k]R)^{\mathcal{I}}(\mathbf{c}) = \sup_{\mathbf{c}' \in \Delta^n,\, \mathbf{c}'[i_1,\ldots,i_k]=\mathbf{c}} R^{\mathcal{I}}(\mathbf{c}')$$

$$(\exists[i_1,\ldots,i_k]R.(Cond_1 \sqcap \ldots \sqcap Cond_l))^{\mathcal{I}}(\mathbf{c}) = \\ \sup_{\mathbf{c}' \in \Delta^n,\, \mathbf{c}'[i_1,\ldots,i_k]=\mathbf{c}} R^{\mathcal{I}}(\mathbf{c}') \otimes Cond_1{}^{\mathcal{I}}(\mathbf{c}') \otimes \ldots \otimes Cond_l{}^{\mathcal{I}}(\mathbf{c}'))$$

Some explanation is in place. Consider $(\exists[i_1,\ldots,i_k]R)$. Informally, from a classical semantics point of view, $(\exists[i_1,\ldots,i_k]R)$ is the projection of the relation R over the

columns $i_1, \ldots, i_k$ and, thus, corresponds to the set of tuples

$$\{\mathbf{c} \mid \exists \mathbf{c}' \in R \text{ s.t. } \mathbf{c}'[i_1, \ldots, i_k] = \mathbf{c}\} \, .$$

Note that for a fixed tuple $\mathbf{c}$ there may be several tuples $\mathbf{c}' \in R$ such that $\mathbf{c}'[i_1, \ldots, i_k] = \mathbf{c}$. Now, if we switch to fuzzy logic, for a fixed tuple $\mathbf{c}$ and interpretation $\mathcal{I}$, each of the previous mentioned $\mathbf{c}'$ is instance of R to a degree $R^{\mathcal{I}}(\mathbf{c}')$. It is usual practice in mathematical fuzzy logic to consider the supremum among these degrees (the existential is interpreted as supremum), which motivates the expression $\sup_{\mathbf{c}' \in \Delta^n, \, \mathbf{c}'[i_1,\ldots,i_k]=\mathbf{c}} R^{\mathcal{I}}(\mathbf{c}')$. The argument is similar for the $\exists [i_1, \ldots, i_k] R.(Cond_1 \sqcap \ldots \sqcap Cond_l)$ construct except that we consider also the additional conditions as conjuncts.

Note also that since in our specific case ${Cond_i}^{\mathcal{I}}(\mathbf{c}') \in \{0, \top\}$ we have that

$$\begin{array}{l} (\exists [i_1, \ldots, i_k] R.(Cond_1 \sqcap \ldots \sqcap Cond_l))^{\mathcal{I}}(\mathbf{c}) = \\ \sup_{\mathbf{c}' \in \Delta^n, \, \mathbf{c}'[i_1,\ldots,i_k]=\mathbf{c}} \min(R^{\mathcal{I}}(\mathbf{c}'), {Cond_1}^{\mathcal{I}}(\mathbf{c}'), \ldots, {Cond_l}^{\mathcal{I}}(\mathbf{c}')) \end{array}$$

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:

$$\mathcal{I} \models (Rl_1 \sqcap \ldots \sqcap Rl_m \sqsubseteq Rr)[n]$$

$$\text{iff for all } \mathbf{c} \in \Delta^k$$

$$n \otimes {Rl_1}^{\mathcal{I}}(\mathbf{c}) \otimes \ldots \otimes {Rl_l}^{\mathcal{I}}(\mathbf{c})) \leqslant Rr^{\mathcal{I}}(\mathbf{c}) \ ,$$

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 abstraction statements, the notion of $\mathcal{I}$ *is a model of* (*satisfies*) a simple abstraction statement σ, denoted $\mathcal{I} \models \sigma$, is defined as follows:

$$\mathcal{I} \models R_1 \mapsto R_2(c_1[t_1], \ldots, c_n[t_n])[c_{score}]$$

$$\text{iff for all } \mathbf{c} \in \Delta^n$$

$${R_1}^{\mathcal{I}}(\mathbf{c}) \geqslant \max\{s \mid \mathbf{c}' \in T_{R_2}, \mathbf{c}'[i_1, \ldots, i_n] = \mathbf{c}, s = \mathbf{c}'[i_s]\} \ ,$$

where $i_1, \ldots, i_n$ are the column numeration corresponding to the column names $c_1, \ldots, c_n$, and i_s is the column number associated to the scoring column c_{score}. Essentially, the value v of ${R_1}^{\mathcal{I}}(\mathbf{c})$ is obtained as follows. We select all tuples $\mathbf{c}'$ in the database table of R_2, *i.e.* T_{R_2}, whose projection on the columns $c_1, \ldots, c_n$ is $\mathbf{c}$. The score of these tuples $\mathbf{c}'$ is $s = \mathbf{c}'[i_s]$. Then the value v is the maximum of all these scores s. [5]

More generally, the notion of $\mathcal{I}$ *is a model of* (*satisfies*) a complex abstraction statement σ, denoted $\mathcal{I} \models \sigma$, is defined as follows: let sql be a SQL statement and let ${sql}^{\mathcal{I}} = \{\langle \mathbf{c}, s \rangle \mid SQL^D(\mathbf{c}, s) = 1\}$ (the set of answers to sql), then

[5] Note that the semantics is quite close to the one of $(\exists [i_1, \ldots, i_n] R_2)(\mathbf{c})$.

$$\mathcal{I} \models R \mapsto (t_1, \ldots, t_n)[c_{score}].\ sql$$

$$\text{iff for all } \mathbf{c} \in \Delta^n$$

$$R^{\mathcal{I}}(\mathbf{c}) \geqslant s \text{ if } \langle \mathbf{c}, s \rangle \in sql^{\mathcal{I}} \ .$$

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

Concerning queries, we may assume that they are of the form

$$q(\mathbf{x})[s] \quad \leftarrow \exists \mathbf{y} \quad \phi(\mathbf{x}, \mathbf{y})[\mathbf{s}] \ , \tag{7}$$

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

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

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.

Now, $\mathcal{I}$ *is a model of* (*satisfies*) a query of the form (7) without ranking aggregates iff for all $\mathbf{c} \in \Delta^n$, $q^{\mathcal{I}}(\mathbf{c}) \geqslant s$ whenever

$$\begin{array}{lll} s & = \sup\{s' \mid & \mathbf{c}' \in \Delta \times \cdots \times \Delta \text{ substitution of variables in } \mathbf{y}, \\ & & \mathcal{I} \models R_i(\mathbf{c}_i)[s_i], \text{ where } \mathbf{c}_i \text{ is the projection of } \langle \mathbf{c}, \mathbf{c}' \rangle \text{ on the variables } \mathbf{z}_i, \\ & & v_j = p_j(\bar{\mathbf{c}}_j), \text{ where } \bar{\mathbf{c}}_j \text{ is the projection of } \langle \mathbf{c}, \mathbf{c}' \rangle \text{ on the variables } \mathbf{z}'_j, \\ & & s' = f(s_1, \ldots, s_l, v_1, \ldots, v_h)\} \ . \end{array}$$

We also assume that $\sup \emptyset$ is undefined. Essentially, as the variables $\mathbf{y}$ are existentially quantified, we take the supremum of the scores s' computed over all possible tuples $\langle \mathbf{c}, \mathbf{c}' \rangle$, where $\mathbf{c}'$ is a substitution for the variables $\mathbf{y}$.

In case we have also ranking aggregates, the definition is slightly more involved. We say that $\mathcal{I}$ *is a model of* (*satisfies*) a query of the form (7) with ranking aggregates iff for all $\mathbf{c} \in \Delta^n$, $q^{\mathcal{I}}(\mathbf{c}) \geqslant s$ whenever

$$\begin{array}{lll} s & = \sup\{s' \mid & g = \{\langle \mathbf{c}, \mathbf{c}'_1 \rangle, \ldots, \langle \mathbf{c}, \mathbf{c}'_k \rangle\} \text{ group of tuples with identical projection on the variables in } \mathbf{w}, \\ & & \text{for } 1 \leqslant r \leqslant k, \mathbf{c}'_r \in \Delta \times \cdots \times \Delta \text{ substitution of variables in } \mathbf{y}, \\ & & \text{for } 1 \leqslant r \leqslant k, \mathcal{I} \models R_i(\mathbf{c}_i)[s_i^r], \text{ where } \mathbf{c}_i \text{ is the projection of } \langle \mathbf{c}, \mathbf{c}'_r \rangle \text{ on the variables } \mathbf{z}_i, \\ & & \text{for } 1 \leqslant r \leqslant k, v_j^r = p_j(\bar{\mathbf{c}}_j), \text{ where } \bar{\mathbf{c}}_j \text{ is the projection of } \langle \mathbf{c}, \mathbf{c}'_r \rangle \text{ on the variables } \mathbf{z}'_j, \\ & & \text{for } 1 \leqslant r \leqslant k, s^r = f(s_1^r, \ldots, s_l^r, v_1^r, \ldots, v_h^r)\} \\ & & s' = @[s^1, \ldots, s^k]\} \ . \end{array}$$

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.

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.

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 $q_i \in \mathbf{q}$, while the *best entailment degree* of

$\mathbf{q}(\mathbf{c})$ relative to $\mathcal{K}$ is $bed(\mathcal{K}, \mathbf{q}(\mathbf{c})) = \sup\{s \mid \mathcal{K} \models \mathbf{q}(\mathbf{c})[s]\}$. The *answer set* of $\mathbf{q}$ w.r.t. $\mathcal{K}$ is

$$ans(\mathcal{K}, \mathbf{q}) = \{\langle \mathbf{c}, s\rangle \mid s = bed(\mathcal{K}, \mathbf{q}(\mathbf{c}))\} .$$

As now each answer to a query has a degree of truth, the basic inference problem that is of interest in SoftFacts is the top-k retrieval problem, formulated as follows.

Top-k Retrieval. 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

$$ans_k(\mathcal{K}, \mathbf{q}) = Top_k\ ans(\mathcal{K}, \mathbf{q}) .$$

Example 5 ([27]) *Suppose the set of axioms is*

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

Let us assume that we have the abstraction statements

$$\begin{array}{rcl} P_2 & \mapsto & TabP2(c[int], s[string]) \\ B & \mapsto & TabB(c[int]) \\ C & \mapsto & TabC(c[int]) , \end{array}$$

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

$$\begin{array}{rcl} TabP2 & = & \{\langle 0, s\rangle, \langle 3, t\rangle, \langle 4, q\rangle, \langle 6, q\rangle\} \\ TabB & = & \{\langle 1\rangle, \langle 2\rangle, \langle 5\rangle, \langle 7\rangle\} \\ TabC & = & \{\langle 5\rangle, \langle 3\rangle, \langle 2\rangle, \langle 4\rangle\} \end{array}$$

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

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

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

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

Now, it can be verified that $\mathcal{K} \models q(3)[0.7]$, $\mathcal{K} \models q(2)[0.84]$ and for any $v \in [0, 1]$, $\mathcal{K} \not\models q(9)[v]$.

In the former case, any model $\mathcal{I}$ of $\mathcal{K}$ satisfies $P_2(3, t)$. But, $\mathcal{I}$ satisfies $\mathcal{O}$, so $\mathcal{I}$ satisfies $\exists[2]P_2 \sqsubseteq \exists P_1$. As $\mathcal{I}$ satisfies $P_2(3, t)$, $\mathcal{I}$ satisfies $(\exists[2]P_2)(t)$ and, thus, $(\exists[1]P_1)(t)$. 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 $ans_4(\mathcal{K}, \mathbf{q}) = [\langle 0, 1.0\rangle, \langle 1, 0.9\rangle, \langle 2, 0.84\rangle, \langle 3, 0.7\rangle]$.* □

4 Query answering

From a query answering point of view, the SoftFacts system extends the DL-Lite/DLR-Lite reasoning method [6] to the fuzzy case. The algorithm is an extension of the one described in [6, 25, 27]). Roughly, 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 $r(\mathbf{q}, \mathcal{O})$. 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 $(B_1 \sqsubseteq A)[n]$ and $(B_2 \sqsubseteq A)[m]$, then we can reformulate the query into two queries

$$\begin{array}{lcl} q(x)[s] & \leftarrow & B_1(x)[s_1], \mathsf{OrderBy}(s = f(n \otimes s_1) \\ q(x)[s] & \leftarrow & B_2(x)[s_2], \mathsf{OrderBy}(s = f(m \otimes s_2) \ ; \end{array}$$

2. from the set of reformulated queries $r(\mathbf{q}, \mathcal{O})$ we remove redundant queries;

3. the reformulated queries $q' \in r(\mathbf{q}, \mathcal{O})$ are translated to ranked SQL queries and evaluated. The query evaluation of each ranked SQL query returns the top-k answer set for that query;

4. all the $n = |r(\mathbf{q}, \mathcal{O})|$ top-k answer sets have to be merged into the unique top-k answer set $ans_k(\mathcal{K}, \mathbf{q})$. As $k \cdot n$ may be large, we apply a *Disjunctive Threshold Algorithm* (DTA, see *e.g.* [27]) to merge all the answer sets.

We next address in more detail the above steps.

At first, the input query $\mathbf{q}$ is *reformulated* into a set of conjunctive queries $r(\mathbf{q}, \mathcal{O})$, by using $\mathcal{O}$ only. After having submitted the queries in $r(\mathbf{q}, \mathcal{O})$ to a top-k database retrieval engine (specifically, RankSQL [22]), we merge the returned ranked lists using the DTA. The DTA is an extension of the one described in [27] and presented later on.

4.1 Query reformulation

The query reformulation step is an extension of [6, 27, 25] to our case and is as follows.

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 $\exists[i_1, \ldots, i_k]R$ 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:

$$\begin{aligned}
([i] \leqslant v)(\mathbf{x}) &= (x_i \leqslant v) \\
([i] < v)(\mathbf{x}) &= (x_i < v) \\
([i] \geqslant v)(\mathbf{x}) &= (x_i \geqslant v) \\
([i] > v)(\mathbf{x}) &= (x_i > v) \\
([i] = v)(\mathbf{x}) &= (x_i = v) \\
([i] \neq v)(\mathbf{x}) &= (x_i \neq v)\ .
\end{aligned}$$

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 $\exists[i_1, \ldots, i_k]R$. 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,

- if $g = A(x)[s]$ and τ is $(Rl_1 \sqcap \ldots \sqcap Rl_m \sqsubseteq A)[n]$ then $gr(g;\tau)$ is $\{C_1(x)[s_1], \ldots, C_m(x)[s_m]\}$, where for each $t \in \{1, \ldots, m\}$, s_t is a new scoring variable, $\theta(g;\tau) = \{s/n \otimes s_1 \otimes \ldots \otimes s_m\}$, and
  - if $Rl_t = A_t$ then $C_t(x) = A_t(x)$;
  - if $Rl_t = \exists[j]R$ then $C_t(x) = \exists z_1, \ldots, z_l.R_t(\mathbf{z})$, where l is the cardinality of R and $\mathbf{z} = \langle z_1, \ldots, z_{j-1}, x, z_{j+1}, \ldots, z_l\rangle$;
  - if $Rl_t = \exists[i]R.(Cond_1 \sqcap \ldots \sqcap Cond_h)$ then $C_t(x) = \exists z_1, \ldots, z_l.R_t(\mathbf{z}) \wedge Cond_1(\mathbf{z}) \wedge \ldots \wedge Cond_h(\mathbf{z})$, where $\mathbf{z} = \langle z_1, \ldots, z_{j-1}, x, z_{j+1}, \ldots, z_l\rangle$ and l is the cardinality of R.
- if $g = R(\_, \ldots, \_, x_{i_1}, \ldots, x_{i_k}, \_, \ldots, \_)[s]$ and τ is $(Rl_1 \sqcap \ldots \sqcap Rl_m \sqsubseteq \exists[i_1, \ldots, i_k]R)[n]$ 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 $t \in \{1, \ldots, m\}$, s_t is a new scoring variable, $\theta(g;\tau) = \{s/n \otimes s_1 \otimes \ldots \otimes s_m\}$, and
  - if $Rl_t = A_t$ and $k = 1$ then $C_t(x_{i_1}, \ldots, x_{i_k}) = A_t(x_{i_1}, \ldots, x_{i_k})$;
  - if $Rl_t = \exists[j_1, \ldots, j_k]R$ then $C_t(x_{i_1}, \ldots, x_{i_k}) = \exists z_1, \ldots, z_l.R_t(\mathbf{z})$, where l is the cardinality of R and $\mathbf{z}$ is such that the variables in position $j_1, \ldots, j_k$ are $x_{i_1}, \ldots, x_{i_k}$;
  - if $Rl_t = \exists[j_1, \ldots, j_k]R.(Cond_1 \sqcap \ldots \sqcap Cond_h)$ then $C_t(x_{i_1}, \ldots, x_{i_k}) = \exists z_1, \ldots, z_l.R_t(\mathbf{z}) \wedge Cond_1(\mathbf{z}) \wedge \ldots \wedge Cond_h(\mathbf{z})$, where l is the cardinality of R and $\mathbf{z}$ is such that the variables in position $j_1, \ldots, j_k$ are $x_{i_1}, \ldots, x_{i_k}$.

Example 6 *Consider the query*

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

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

$$\begin{aligned}
\tau_1 &:= (B_1 \sqsubseteq A)[0.8] \\
\tau_2 &:= (B_2 \sqsubseteq A)[0.7]\ .
\end{aligned}$$

Suppose that $a \otimes b = a \cdot b$ and that $g = A(x)[s_1]$. Then

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

Now, using $gr(g;\tau_i)$ and $\theta(g;\tau_i)$ we may reformulate the original query by replacing in the query g with the elements in $gr(g;\tau_1)$ and then applying the score variable substitution $\theta(g;\tau_i)$ to the scoring atom. Therefore, we get two new queries

$$\begin{array}{lcl} q_1 & := & q(x)[s] \leftarrow B_1[s_3], B(x)[s_2], \mathsf{OrderBy}(s = \min(0.8 \cdot s_3, s_2)) \\ q_2 & := & q(x)[s] \leftarrow B_2[s_4], B(x)[s_2], \mathsf{OrderBy}(s = \min(0.9 \cdot s_4, s_2))\ . \end{array}$$

□

We are now ready to present the query reformulation algorithm.

Given a disjunctive query $\mathbf{q}$ and a set of axioms $\mathcal{O}$, the algorithm reformulates q in terms of a set of conjunctive queries $r(\mathbf{q}, \mathcal{O})$, which then can be evaluated over the facts $\mathcal{F}$ using the mappings in the abstraction component $\mathcal{A}$.

Algorithm 1 $QueryRef(\mathbf{q}, \mathcal{O})$

Input: A disjunctive query $\mathbf{q}$, SoftFacts axioms $\mathcal{O}$.
Output: Set of reformulated conjunctive queries $r(\mathbf{q}, \mathcal{O})$.

1: $r(\mathbf{q}, \mathcal{O}) := \mathbf{q}$
2: **repeat**
3: $S = r(\mathbf{q}, \mathcal{O})$
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: $r(\mathbf{q}, \mathcal{O}) := removeSubs(r(\mathbf{q}, \mathcal{O}))$
12: **until** $S = r(\mathbf{q}, \mathcal{O})$
13: **return** $r(\mathbf{q}, \mathcal{O})$

In the algorithm, $q[g/g']\theta(g,\tau)$ 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 [6].

Due to the unification, variables that were bound in q may become unbound in q'. Hence, inclusion axioms that were 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 $r(\mathbf{q}, \mathcal{O})$, those which are already subsumed in $r(\mathbf{q}, \mathcal{O})$. The notion of query subsumption is similar as for the classical database theory [30]. 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*

[6] 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 $z_i = y_i$ [4].

q_2, denoted $q_1 \sqsubseteq q_2$, iff for any interpretation $\mathcal{I}$, for all tuples $\mathbf{c}$, ${q_1}^{\mathcal{I}}(\mathbf{c}) \leqslant {q_2}^{\mathcal{I}}(\mathbf{c})$. Essentially, if $q_1 \sqsubseteq q_2$ and both q_1 and q_2 belong to $r(\mathbf{q}, \mathcal{O})$ then we can remove q_1 from $r(\mathbf{q}, \mathcal{O})$ as q_1 produces a lower ranked result than q_2 with respect to the same tuple $\mathbf{c}$. In order to decide query subsumption, we can take advantage of the results in [20], related to the query containment part.

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 that

Proposition 1 ([27]) *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 7 (Example 6 cont.) *Consider Example 6. Suppose that the ontology component $\mathcal{O}$ has additionally the recursive axiom*

$$\tau_3 \quad := \quad (A \sqcap B_3 \sqsubseteq A)[0.9]$$

Therefore, through the query reformulation procedure we get a new query

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

Now, let's compare q_0 with q_3. It turns out that $q_3 \sqsubseteq q_0$ as for any score n of a constant c being instance of atom A, $\min(0.9 \cdot n \cdot s_6, s_2)) \leqslant \min(n, s_2))$. for any value for s_2 and s_6.

□

We can extend the query subsumption condition in Proposition 1 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 it can be shown that

Proposition 2 ([27]) *$q_1 \sqsubseteq q_2$ 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 8 (Example 7 cont.) *Consider Example 7. Let $\theta = \{s_5/s_1\}$ then $\sigma_0 = \min(s_5, s_2))$, while $\sigma_3\theta = \min(0.9 \cdot s_5 \cdot s_6, s_2))$. It is easily verified that $\min(0.9 \cdot s_5 \cdot s_6, s_2)) = \sigma_3\theta \leqslant \sigma_1 = \min(s_5, s_2))$ as the score combination function is monotone in its arguments and $0.9 \cdot s_5 \cdot s_6 \leqslant s_5$.*

□

We use Proposition 2 to avoid the recursive application of the query reformulation steps (see, *e.g.*, Example 7). 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 of $\otimes$ and that $\otimes$ is bounded, *i.e.* $a \otimes b \leqslant a$. This concludes the query reformulation step.

Example 9 ([27]) *Consider Example 5. At step 1 $r(\mathbf{q}, \mathcal{O})$ 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,*

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

into $r(\mathbf{q}, \mathcal{O})$ using the axiom $A \sqsubseteq \exists[1]P_1$. Note that the weight of the axiom is 1 and that $f(1 \otimes x) = f(x)$.

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

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

using the axiom $\exists[2]P_2 \sqsubseteq A$.

Since the two atoms of the second query unify, $reduce(q, g_1, g_2)$ 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,

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

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

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

using the axiom $B \sqsubseteq \exists[1]P_2$ and stops.

Note that we need not to evaluate all queries q_i. Indeed, it can easily be verified that for each query 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 $r(\mathbf{q}, \mathcal{O}) = \{q'', q_3, q_4\}$ 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 and extends the result in [27] to SoftFacts:

$$ans_k(\mathcal{K}, \mathbf{q}) = Top_k\{\langle \mathbf{c}, v\rangle \mid q_i \in r(\mathbf{q}, \mathcal{O}), \mathcal{F} \models q_i(\mathbf{c}, v)\} .$$

The above property dictates that the set of reformulated queries $q_i \in r(\mathbf{q}, \mathcal{O})$ 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 $q_i \in r(\mathbf{q}, \mathcal{O})$.

A naive solution to the top-k retrieval problem is as follows: we compute for all $q_i \in r(\mathbf{q}, \mathcal{O})$ the whole answer set $ans(q_i, \mathcal{F}) = \{\langle \mathbf{c}, v\rangle \mid \mathcal{F} \models q_i(\mathbf{c}, v)\}$, then we compute the union, $\bigcup_{q_i \in r(\mathbf{q},\mathcal{O})} ans(q_i, \mathcal{O})$, 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 $q_i \in r(\mathbf{q}, \mathcal{O})$ can easily be transformed into an SQL query expressed over $DB(\mathcal{F})$, *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 $q_i \in r(\mathbf{q}, \mathcal{O})$, all these scores should be computed and the tuples should be retrieved. This is *not feasible* in practice, as a there may be millions of tuples in the knowledge base.

4.3 The DTA without ranking aggregates

A more effective solution consists in relying on existing top-k query answering algorithms for relational databases (see, *e.g.* [9, 12, 19, 22, 21]), which support efficient evaluations of ranking top-k queries in relational database systems. As shown in [27], we can profitably use top-k query answering methods for relational databases. Indeed, an immediate and much more efficient method to compute $ans_k(\mathcal{K}, \mathbf{q})$ is: we compute for all $q_i \in r(\mathbf{q}, \mathcal{O})$, the top-$k$ answers $ans_k(\mathcal{F}, q_i)$, using *e.g.* the system RankSQL [21]. If both k and the number, $n_q = |r(\mathbf{q}, \mathcal{O})|$, of reformulated queries is reasonable, then we may compute the union,

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

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 [27] of the so-called *Threshold Algorithm* (TA) [13], which we call *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 recall that we can use the same principles of the TA to compute the top-k answers of the union of conjunctive queries, *i.e.* a disjunctive query *without* ranking aggregates:

1. First, we compute for all $q_i \in r(\mathbf{q}, \mathcal{O})$, 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 $ans_k(\mathcal{F}, q_i)$ ($q_i \in r(\mathbf{q}, \mathcal{O})$) 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 tuple $\mathbf{c}$ seen, if its score is one of the k highest we have seen, 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 seen in this set. Define the threshold value θ to be $\max(\theta_1, ..., \theta_{n_q})$.

(c) As soon as at least k tuples have been seen 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 seen with the highest scores. The output is then the set $\{\langle \mathbf{c}, s_{\mathbf{c}} \rangle \mid \mathbf{c} \in Y\}$. This set is $ans_k(\mathcal{K}, \mathbf{q})$.

It is not difficult to see that the DTA determines the top-k answers. Indeed, if at least k tuples have been seen whose score is at least equal to θ then any new unseen 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 seen.

The following example illustrates the DTA.

Example 10 ([27]) *Consider Example 9. 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*

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

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*	-	-	*1.0*	$\langle 0, 1.0\rangle$
2	$\langle 1, 0.9\rangle$	*1.0*	*0.9*	-	*1.0*	$\langle 0, 1.0\rangle, \langle 1, 0.9\rangle$
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 $q_i \in r(\mathbf{q}, \mathcal{O})$ requires (sub) linear time w.r.t. the database size (using, *e.g.* [9]), it is easily verified that the disjunctive TA algorithm is linear in data complexity.

Proposition 3 ([27]) *Given $\mathcal{K} = \langle \mathcal{F}, \mathcal{O}, \mathcal{A}\rangle$ and a disjunctive query $\mathbf{q}$ without ranking aggregates, then the DTA computes $ans_k(\mathcal{K}, \mathbf{q})$ 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.

4.4 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 (ranking aggregates have been not considered in [27]), as illustrated in the following example.

Example 11 *Suppose that* $r(\mathbf{q}, \mathcal{O}) = \{q_1, q_2\}$*, where*

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

and that

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

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*	-	*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 the top-3.* □

The next example shows an even more serious problem.

Example 12 *Suppose that* $r(\mathbf{q}, \mathcal{O}) = \{q_1, q_2\}$*, where*

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

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

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

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*

$$\begin{array}{rcl} ans_1(\mathcal{F}, q_1) & = & [\langle a, 1.0\rangle] \\ ans_1(\mathcal{F}, q_2) & = & [\langle b, 0.9\rangle]\ . \end{array}$$

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 wrong scores, but more importantly we get a wrong ranking result (tuple* a *would be top-1).* □

As shown in Examples 11 and 12, in presence of ranking aggregates, unseen tuples in top-k answers, may affect the final score. This is the case for $@ \in \{\mathsf{MIN}, \mathsf{AVG}, \mathsf{SUM}\}$, but not for $@ = \mathsf{MAX}$. In fact, we have that:

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 *Given* $\mathcal{K} = \langle \mathcal{F}, \mathcal{O}, \mathcal{A}\rangle$ *and a disjunctive query* $\mathbf{q}$ *with ranking aggregates* MAX *only, then the DTA computes* $ans_k(\mathcal{K}, \mathbf{q})$ *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 $q_j \in r(\mathbf{q}, \mathcal{O})$, consider the answer set $L_j = ans(\mathcal{F}, q_j)$ $(1 \leqslant j \leqslant n = |r(\mathbf{q}, \mathcal{O})|)$. Let $\mathbf{c}_i$ be a tuple and assume that it occurrs in the $r \leqslant n$ lists $L_{i_1}, ..., L_{i_r}$ 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, $\langle \mathbf{c}, s\rangle \in ans(\mathcal{K}, \mathbf{q})$ 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 12 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 $\mathbf{c}$.

For positive integer $i \geqslant 1$, with $ans_{i,k}(\mathcal{F}, q)$ we will denote the ranked list of k answers of q, ranked between position $(i-1)\cdot k + 1$ and $i \cdot k$, *i.e.*

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

where $ans_0(\mathcal{F}, q) = \emptyset$. For instance, $ans_{1,10}(\mathcal{F}, q)$ 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 $ans_{i,k}(\mathcal{F}, q)$ to incrementally retrieve a successive ordered group of k answers to q. We point out that $ans_{i,k}(\mathcal{F}, q)$ can be computed, in *e.g.* RankSQL, by means of a statement of the form

```
SELECT    ...
FROM      ...
GROUPBY   ...
ORDERBY   ... DESC
LIMIT     k
OFFSET    (i − 1) · k
```

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

1. Initialization: let $n = |r(\mathbf{q}, \mathcal{O})|$, for all $q \in r(\mathbf{q}, \mathcal{O})$, let $i_q = 1, L_q = \emptyset$ and let $\theta = 1.0$. Compute for all $q \in r(\mathbf{q}, \mathcal{O})$, the top-$k$ answers and let $L_q = ans_k(\mathcal{F}, q)$, using top-$k$ rank-based relational database engine. We will 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 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 seen in this set.

   (b) If for a list L_q we have processed the last tuple in it and we didn't stop, then update $i_q := i_q + 1$ and append $ans_{i_q,k}(\mathcal{F}, q)$ to L_q (*i.e.*, retrieve the next k answers of q). If $ans_{i_q,k}(\mathcal{F}, q) = \emptyset$ do not process further L_q and set $\theta_q = 0$. Such a list L_q is called *completed*.

   (c) For each tuple $\mathbf{c}$ seen so far, let $\bar{\mathcal{L}}_{\mathbf{c}}$ be the set of (non-completed) ranked lists in which $\mathbf{c}$ has not yet been seen so far;

   (d) For each tuple $\mathbf{c}$ seen, let $l_{\mathbf{c}} = [s_{i_1}, \ldots, s_{i_r}]$ be the actual list of r scores seen 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 seen in all lists $L_{q_1}, \ldots, L_{q_n}$, or all lists in which $\mathbf{c}$ haven't be seen so far are completed, then $s_{\mathbf{c}}$ is the definitive score of $\mathbf{c}$;

   (f) For each tuple $\mathbf{c}$ seen so far, which has not yet a definitive score, we define the *upper score* of $\mathbf{c}$, $\bar{s}_{\mathbf{c}}$, as follows ($\bar{s}_{\mathbf{c}}$ is the highest possible score $\mathbf{c}$ may eventually achieve):

   - if $@ = \mathsf{Min}$ then $\bar{s}_{\mathbf{c}}$ is $s_{\mathbf{c}}$
   - if $@ = \mathsf{AVG}$ then $\bar{s}_{\mathbf{c}}$ is defined as follows. If there is no $L \in \bar{\mathcal{L}}_{\mathbf{c}}$ with $\theta_L > s_{\mathbf{c}}$ then $\bar{s}_{\mathbf{c}} := s_{\mathbf{c}}$. Otherwise, $\bar{s}_{\mathbf{c}} := (r \cdot s_{\mathbf{c}} + \max_{L \in \bar{\mathcal{L}}_{\mathbf{c}}} \theta_L)/(r + 1)$. In summary,

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

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

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

   (g) As soon as at least k tuples have been seen whose actual score is *definitive* and is at least equal to the definitive score or upper score of all other seen tuples, then halt.

   (h) Let Y be the set containing the k tuples that have been seen with the highest definitive score. The output is then the set $\{\langle \mathbf{c}, s_{\mathbf{c}} \rangle \mid \mathbf{c} \in Y\}$. This set is $ans_k(\mathcal{K}, \mathbf{q})$.

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, we have k tuples have been seen whose actual score is *definitive* and is at least equal to the definitive score or upper score of all other seen tuples. Consider a seen tuple $\langle \mathbf{c}, s_{\mathbf{c}} \rangle$, which didn't 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 $L \in \bar{\mathcal{L}}_{\mathbf{c}}$. Suppose that $\mathbf{c}$ will not eventually occur in some of the ranked lists $L \in \bar{\mathcal{L}}_{\mathbf{c}}$. 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 $L \in \bar{\mathcal{L}}_{\mathbf{c}}$, then we have to consider all three cases $@ \in \{\mathsf{MIN}, \mathsf{AVG}, \mathsf{SUM}\}$. By the sopping criteria $s_k \geqslant \bar{s}_{\mathbf{c}}$ 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 $s_L \leqslant \max_{L \in \bar{\mathcal{L}}_{\mathbf{c}}} \theta_L$. Therefore, the definitive score of $\mathbf{c}$ cannot be greater than $\max_{L \in \bar{\mathcal{L}}_{\mathbf{c}}} (r \cdot s_{\mathbf{c}} + s_L)/(r+1) \leqslant (r \cdot s_{\mathbf{c}} + \max_{L \in \bar{\mathcal{L}}_{\mathbf{c}}} \theta_L)/(r+1) = \bar{s}_{\mathbf{c}} \leqslant s_k$.

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

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

Proposition 5 *Given $\mathcal{K} = \langle \mathcal{F}, \mathcal{O}, \mathcal{A} \rangle$ and a disjunctive query $\mathbf{q}$ with ranking aggregates, then DTA@ computes $ans_k(\mathcal{K}, \mathbf{q})$ in (sub) linear time w.r.t. the size of $\mathcal{F}$.*

Example 13 *Consider Example 12. The execution of the DTA@ is shown below. 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 seen tuples. Note that none of the tuples a, e may make it into the top-1. □

5 Experiments

We conducted an experiment with the SoftFacts system. We considered an ontology for describing Curriculum Vitæ. It consists of 5115 axioms, 22 abstraction statements and 2550 relations. We considered size $d = 100000$ for the fact component consisting in automatically generated CVs and stored them into a database having 17 relational tables. We build several queries, with/without scoring atom and with/without ranking aggregates to be submitted to the system varying the size of the CVs and different values for k in case of top-k retrieval ($k \in \{1, 10\}$), and measured for each query

1. the number of queries generated after the reformulation process ($|r(\mathbf{q}, \mathcal{O})|$);
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}, $t_{DB_{10}}$).

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, where no indexes have been used for the facts in the relational database. The queries 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

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

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 $rs(mark; 100, 110)$

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

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

q(id, lastName, degreeName, mark, hasKnowledge, years, kType)
← 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) · 0.4 + rs(years; 15, 25) · pref(level; Good/0.6, Excellent/1.0) · 0.6)

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)
← 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) · 0.4 + rs(years; 15, 25) · 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 1.0, 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.

The tests have been performed on a MacPro machine with Mac OS X 10.5.5, 2 x 3 GHz Dual-Core processor and 9 GB or RAM and the results are shown in Fig. 6 (time is measured in seconds). Let us consider few comments about the results:

- overall, the response time is reasonable (almost fraction of second) taking into account the non negligible size of the ontology, the number of CVs and that we did not consider any index for the relational tables;
- 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;
- we may note that the size of the set of reformulated queries $|r(\mathbf{q}, \mathcal{O})|$ may be non negligible, that the redundancy elimination may remove almost one third of them and the time of the reduction phase is negligible;

Size 100000											
Query	All	top-1	top-10	$\|ans(\mathcal{K}, q)\|$	$\|r(\mathbf{q}, \mathcal{O})\|$	q_{DB}	t_{ref}	t_{red}	$t_{DB_{all}}$	t_{DB_1}	$t_{DB_{10}}$
1	7.507	2.489	2.581	3985	1599	1066	1.723	0.010	5.738	0.203	0.258
2	0.193	0.036	0.037	445	69	46	0.016	0.001	0.137	0.017	0.039
3	0.164	0.030	0.066	19	18	12	0.006	0.001	0.126	0.015	0.057
4	3.194	2.143	3.155	8	1242	552	2.072	0.015	1.075	0.106	0.875
5	0.348	0.067	0.186	19	75	50	0.027	0.001	0.300	0.021	0.141
6	0.114	0.052	0.102	93	18	12	0.005	0.001	0.088	0.036	0.082
7	0.207	0.053	0.146	445	69	46	0.013	0.001	0.166	0.014	0.118
8	3.090	2.353	3.080	21	1242	552	1.242	0.013	1.306	0.512	1.125
9	22.764	22.702	22.754	91	5175	2300	17.850	0.058	4.819	4.604	4.766
10	0.759	0.378	0.369	40	108	48	0.229	0.003	0.498	0.159	0.145
11	0.105	0.100	0.101	37	18	12	0.004	0.001	0.075	0.072	0.074
12	2.2	2.038	2.128	15	828	552	0.794	0.005	1.370	1.296	1.128
Average	3.834	3.0303	3.248	516.6	961.5	468.4	2.318	0.01	1.4053	0.601	0.761
Median	.0554	0.223	0.278	65.5	91.5	49	0.128	0.002	0.399	0.133	0.143

Figure 6: Retrieval statistics.

- the answering time of the database is increasing with the number of top-k results to be retrieved.

We also point out that for query 9, the major time is spent for the query reformulation phase and, only after this time, we submit the queries to the database.

6 Related Work

While there are many works addressing the top-k problem for vague queries in databases (cf. [3, 9, 13, 12, 18, 19, 22, 21, 24]), little is known for the corresponding problem in knowledge representation and reasoning. For instance, [32] 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 expensive fuzzy predicates are allowed). The work [26] considers non-recursive fuzzy logic programs as well, though the score combination function may consider expensive fuzzy 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 [26]. The works [27, 25] 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 expensive fuzzy predicates. However, a score combination function is allowed in the query only. The work [29] shows an application of top-k retrieval to the case of multimedia information retrieval by relying on a fuzzy variant of *DLR-Lite*. Finally, [28] addresses the top-k retrieval for general (recursive) fuzzy LPs, though no expensive fuzzy predicates are allowed. Closest to our work is clearly [28]. In fact, our work extends [28] by allowing expensive fuzzy predicates, which have the effect that the threshold mechanism designed in [28] does not work anymore. Furthermore, in this paper, we made an effort to plug-in current top-k database technology, while [28] does not and provides an ad-hoc solution.

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 in the SoftFacts system, an ontology mediated top-k information retrieval system over relational databases. In SoftFacts, 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 an abstraction component. The results of a query may be ranked according to some scoring function. We have illustrate the logical model, the architecture, the representation and the query language, the reasoning algorithms of the SoftFacts system. We have also provided experiments, which show promising results from a scalability point of view.

References

[1] F. Baader, S. Brandt, and C. Lutz. Pushing the $\mathcal{EL}$ envelope. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence IJCAI-05*, pages 364–369, Edinburgh, UK, 2005. Morgan-Kaufmann Publishers.

[2] Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.

[3] Nicolas Bruno, Surajit Chaudhuri, and Luis Gravano. Top-k selection queries over relational databases: Mapping strategies and performance evaluation. *ACM Trans. Database Syst.*, 27(2):153–187, 2002.

[4] Andrea Calì, Domenico Lembo, and 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)*, pages 16–21, 2003.

[5] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. DL-Lite: Tractable description logics for ontologies. In *Proc. of the 20th Nat. Conf. on Artificial Intelligence (AAAI 2005)*, 2005.

[6] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and 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)*, pages 260–270, 2006.

[7] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, Antonella Poggi, Riccardo Rosati, and Marco Ruzzi. Data integration throughdl-lite$_\text{a}$ ontologies. In *Semantics in Data and Knowledge Bases, Third International Workshop, SDKB 2008, Nantes, France, March 29, 2008, Revised Selected Papers*, number 4925 in Lecture Notes in Computer Science, pages 26–47. Springer Verlag, 2008.

[8] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. Data complexity of query answering in description logics. In *Proceedings of the 2005 International Workshop on Description Logics (DL-05)*, volume 147, 2005.

[9] Kevin Chen-Chuan Chang and Seung won Hwang. Minimal probing: Supporting expensive predicates for top-k queries. In *SIGMOD Conference*, 2002.

[10] CuiMing Chen, Volker Haarslev, and JiaoYue Wang. Las: Extending racer by a large abox store. In Ian Horrocks, Ulrike Sattler, and Frank Wolter, editors, *Proceedings of the 2005 International Workshop on Description Logics (DL-05)*, volume 147. CEUR-WS.org, 2005.

[11] Jérôme Euzenat and Petko Valtchev. *Ontology Matching*. Springer Verlag, 2007.

[12] Ronald Fagin. Combining fuzzy information: an overview. *SIGMOD Rec.*, 31(2):109–118, 2002.

[13] Ronald Fagin, Amnon Lotem, and Moni Naor. Optimal aggregation algorithms for middleware. In *Symposium on Principles of Database Systems*, 2001.

[14] Petr Hájek. *Metamathematics of Fuzzy Logic*. Kluwer, 1998.

[15] Ian Horrocks, Lei Li, Daniele Turi, and Sean Bechhofer. The instance store: DL reasoning with large numbers of individuals. In *Proc. of the 2004 Description Logic Workshop (DL 2004)*, pages 31–40, 2004.

[16] Ian Horrocks, Peter F. Patel-Schneider, and Frank van Harmelen. From $\mathcal{SHIQ}$ and RDF to OWL: The making of a web ontology language. *Journal of Web Semantics*, 1(1):7–26, 2003.

[17] Ullrich Hustadt, Boris Motik, and Ulrike Sattler. Data complexity of reasoning in very expressive description logics. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI-05)*, pages 466–471. Professional Book Center, 2005.

[18] Ihab F. Ilyas, Walid G. Aref, and Ahmed K. Elmagarmid. Supporting top-k join queries in relational databases. In *Proceedings of 29th International Conference on Very Large Data Bases (VLDB-03)*, pages 754–765, 2003.

[19] Ihab F. Ilyas, Walid G. Aref, Ahmed K. Elmagarmid, Hicham G. Elmongui, Rahul Shah, and Jeffrey Scott Vitter. Adaptive rank-aware query optimization in relational databases. *ACM Transactions on Database Systems*, 31(4):1257–1304, 2006.

[20] Laks V.S. Lakshmanan and Nematollaah Shiri. A parametric approach to deductive databases with uncertainty. *IEEE Transactions on Knowledge and Data Engineering*, 13(4):554–570, 2001.

[21] Chengkai Li, Kevin Chen-Chuan Chang, and Ihab F. Ilyas. Supporting ad-hoc ranking aggregates. In *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data (SIGMOD-06)*, USA, 2006. ACM Press.

[22] Chengkai Li, Kevin Chen-Chuan Chang, Ihab F. Ilyas, and Sumin Song. RankSQL: query algebra and optimization for relational top-k queries. In *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data (SIGMOD-05)*, pages 131–142, New York, NY, USA, 2005. ACM Press.

[23] Thomas Lukasiewicz and 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)*, number 4772 in Lecture Notes in Computer Science, pages 16–30. Springer Verlag, 2007.

[24] Amélie Marian, Nicolas Bruno, and Luis Gravano. Evaluating top-k queries over web-accessible databases. *ACM Trans. Database Syst.*, 29(2):319–362, 2004.

[25] 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)*, pages 2238–2245. E.D.K., Paris, 2006.

[26] 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)*, pages 4873–4879. IEEE, 2006.

[27] 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)*, number 4160 in Lecture Notes in Computer Science, pages 439–451, Liverpool, UK, 2006. Springer Verlag.

[28] 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)*, number 4529 in Lecture Notes in Computer Science, pages 125–134, Cancun, Mexico, 2007. Springer Verlag.

[29] Umberto Straccia and Giulio Visco. DL-Media: an ontology mediated multimedia information retrieval system. In *Proceeedings of the International Workshop on Description Logics (DL-07)*, volume 250, Insbruck, Austria, 2007. CEUR.

[30] J. D. Ullman. *Principles of Database and Knowledge Base Systems*, volume 1,2. Computer Science Press, Potomac, Maryland, 1989.

[31] M. Vardi. The complexity of relational query languages. In *Proc. of the 14th ACM SIGACT Sym. on Theory of Computing (STOC-82)*, pages 137–146, 1982.

[32] Peter Vojtáš. Fuzzy logic aggregation for semantic web search for the best (top-k) answer. In Elie Sanchez, editor, *Fuzzy Logic and the Semantic Web*, Capturing Intelligence, chapter 17, pages 341–359. Elsevier, 2006.