Bridging Information Retrieval and Databases

Norbert Fuhr

University of Duisburg-Essen, Germany
norbert.fuhr@uni-due.de

Abstract. For bridging the gap between information retrieval (IR) and databases (DB), this article focuses on the logical view. We claim that IR should adopt three major concepts from DB, namely inference, vague predicates and expressive query languages. By regarding IR as uncertain inference, probabilistic versions of relational algebra and Datalog yield very powerful inference mechanisms for IR as well as allowing for more flexible systems. For dealing with various media and data types, vague predicates form a natural extension of text retrieval methods to attribute values, thus switching from propositional to predicate logic. A more expressive IR query language should support joins, be able to compute aggregated results, and allow for restructuring of the result objects.

1 Introduction

For several decades, information retrieval (IR) and databases (DB) have evolved as separate subfields of computer science (see e.g. the juxtaposition in [18, ch. 1]). However, in recent years, there have been increasing research activities to bridge the gap between these two areas and develop approaches integrating IR and DB features. There are various levels where such an integration can take place, namely at the physical, the logical or the conceptual level of information systems. In this article, we will focus on the logical level, mainly due to the fact that there is a nice theoretical framework that supports the integration of IR and DB at this level.

In the logical view on DB, the (retrieval) task of the system can be described as follows: given a query q, find objects o which imply the query, i. e. $o \rightarrow q$. On the other hand,, Rijsbergen defines IR as being based on uncertain inference where for a given query q, the IR system should compute the probability $P(d \rightarrow q)$ for each document d. By comparing the two definitions, we can see that IR can be regarded as a generalization of the DB approach here, since it replaces deterministic by uncertain inference.

Based on this interpretation, this article discusses how three major DB concepts can be adopted and extended in order to enhance current IR systems. In the next section, we will focus on inference, showing how probabilistic versions of relational algebra and Datalog increase the inferential capabilities of IR systems. Section 3 introduces vague predicates as a method for extending classical IR methods for dealing with attribute values and multimedia data. Query language expressiveness is discussed in Section 4, pointing out potential benefits

N. Ferro (Ed.): PROMISE Winter School 2013, LNCS 8173, pp. 97–115, 2014.

from more expressive IR query languages. Two further concepts are briefly addressed in Section 5, namely four-valued logic and the architecture of future IR systems. Section 6 concludes this contribution.

2 Inference

Following Rijsbergen's interpretation of IR as uncertain inference, this section will demonstrate the close connection between IR and the logical view on databases. For that, we start from relational algebra. As uniform notation, we will use Datalog (see e.g. [17], [3]).

First we show how document retrieval can be formulated in Datalog. For that, we assume that there is a predicate (a database relation) `docTerm(D,T)`, where each ground fact gives for a document `D` a term `T` the document is indexed with, e.g.:

```
docTerm(d1,ir). docTerm(d2,ir).
docTerm(d1,db). docTerm(d2,oop).
```

In our notation of Datalog formulas, constants start with lowercase letters and variables with capitals. A query now can be formulated as a logical formula involving the predicate `docTerm`, e.g.

`?- docTerm(D,ir)` searches for documents about IR, and

`?- docTerm(D,ir) & docTerm(D,db)` for documents both about IR and DB.

For demonstrating the close connection between relational algebra and IR, we also use the notation of database relations in tabular form. Our running example consists of the two relations shown in Figure 1. Now we discuss the five basic operations of relational algebra.

docTerm

DOCNO	TERM
1	ir
1	db
2	ir
3	db
3	oop
4	ir
4	ai
5	db
5	oop

author

DOCNO	NAME
1	smith
2	miller
3	johnson
4	firefly
4	bradford
5	bates

Fig. 1. Relations in our example database

Projection. As an example for projection, let us ask what the collection is about: `topic(T) :- docTerm(D,T)`, which results in the following four tuples:

```
topic(ir). topic(db). topic(oop). topic(ai).
```

Selection. If we want to know which documents are about IR, we would ask `aboutir(D) :- docTerm(D,ir)`, which returns the tuples
`aboutir(1). aboutir(2). aboutir(4).`

Join. This operation allows for the combination of two relations (which is an unusual concept in standard IR, where we mostly assume that all the necessary data is within one document or object). As an example, we want to know authors writing about IR:
`irauthor(A):- docTerm(D,ir) & author(D,A)`, resulting in the four tuples
`irauthor(smith). irauthor(miller).`
`irauthor(firefly). irauthor(bradford).`

Union. If we want to know documents about IR or DB, this can be expressed via the union operator, which we map onto disjunction in Datalog:
`irordb(D) :- docTerm(D,ir). irordb(D) :- docTerm(D,db)`, giving us
`irordb(1). irordb(2). irordb(3). irordb(4). irordb(5).`

Difference. The last of the five basic relational algebra operators is (set) difference, which we can use e.g. for finding documents about IR, but not about DB:
`irnotdb(D) :- docTerm(D,ir) & not(docTerm(D,db))`, leading to the answer
`irnotdb(2). irnotdb(4).`

2.1 The Probabilistic Relational Model

Since IR is about *uncertain* inference, we have to add probabilities to the relational model [10,16], and switch from deterministic to probabilistic Datalog (pD) [8]. For that, let us assume, that we attach a probability value to each tuple, which gives a probability that this specific tuple belongs to the relation under consideration (see the example relation in Figure 2).

docTerm

β	DOCNO	TERM
0.9	1	IR
0.5	1	DB
0.6	2	IR
0.7	3	DB
0.8	3	OOP
0.9	4	IR
0.4	4	AI
0.8	5	DB
0.3	5	OOP

Fig. 2. Example probabilistic relation

Asking now for documents about DB via
`aboutdb(D) :- docTerm(D,db).` selects the following three tuples with their corresponding probabilistic weights:
`0.5 aboutdb(1). 0.7 aboutdb(3). 0.8 aboutdb(5).`

In contrast, when we ask for documents about both IR and DB
`aboutirdb(D) :- docTerm(D,ir) & docTerm(D,db),`
the weights of the tuples have to be combined. Assuming probabilistic independence (as is standard in most IR applications), we get
`0.45 aboutirdb(1).`

Extensional vs. intensional semantics. For more complex queries, however, we have to be careful in order to get the probabilities right. For illustrating this point, let us look at the following example:

```
0.9 docterm(d1,ir). 0.5 docterm(d1,db). 0.7 link(d2,d1)
about(D,T) :- docTerm(D,T).
about(D,T) :- link(D,D1) & about(D1,T)
q(D) :- about(D,ir) & about(D,db).
```

Obviously, the correct result cannot be computed in a straightforward way, like

$$
\begin{aligned}
&P(\texttt{q(d2)}) = \\
&= P(\texttt{about(d2,ir)}) \cdot P(\texttt{about(d2,db)}) \\
&= P(\texttt{link(d2,d1)}) \cdot P(\texttt{docterm(d1,ir)}) \cdot P(\texttt{link(d2,d1)}) \cdot P(\texttt{docterm(d1,db)}) \\
&= (0.7 \cdot 0.9) \cdot (0.7 \cdot 0.5).
\end{aligned}
$$

The problem is that the probability associated with the link is multiplied twice into the result. This approach of combining the weights without paying attention to the associated probabilistic events is also called *extensional semantics*, where we suffer from "improper treatment of correlated sources of evidence" [13].

Instead, we have to use *intensional semantics*, where the weight of any derived fact is computed as a function of weights of underlying ground facts.

In [10] the concept of *event keys and event expressions* is introduced for handling intensional semantics. Here each tuple in a base relation of our database is associated with a unique identifier, a so-called event key, which denotes the corresponding probabilistic event (for didactic reasons, here we use event keys denoting the original tuple in abbreviated form), as in the following example

docterm

β	κ	DOC	TERM
0.9	dT(d1,ir)	d1	ir
0.5	dT(d1,db)	d1	db

link

β	κ	S	T
0.7	l(d2,d1)	d2	d1

Fig. 3. Probabilistic relations with event keys

For any derived fact, we now compute an event expression as Boolean combination of the underlying event keys, like e.g.

```
?- docTerm(D,ir) & docTerm(D,db).
```
resulting in
```
d1 [dT(d1,ir) & dT(d1,db)]                                  0.9 · 0.5 = 0.45
```
For the more complex query from above `?- about(D,ir) & about(D,db)`, we get
```
d1 [dT(d1,ir) & dT(d1,db)]                                  0.9 · 0.5 = 0.45
d2 [l(d2,d1) & dT(d1,ir) & l(d2,d1) & dT(d1,db)] 0.7 · 0.9 · 0.5 = 0.315
```

Recursion. Probabilistic Datalog can also deal with recursive rules, without running into problems. As an example, consider the probabilistic facts illustrated in Figure 4.

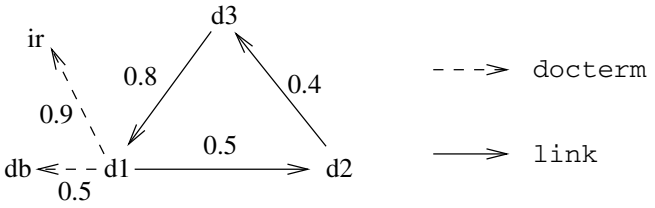

Fig. 4. An example for probabilistic rules with recursion

Using the same rules as before
```
about(D,T) :- docTerm(D,T).
about(D,T) :- link(D,D1) & about(D1,T).
```
the query `?- about(D,ir)` would result in the following derived facts:
```
d1 [dT(d1,ir) | l(d1,d2) & l(d2,d3) & l(d3,d1) &
     dT(d1,ir) | ...]                                       0.900
d3 [l(d3,d1) & dT(d1,ir)]                                   0.720
d2 [l(d2,d3) & l(d3,d1) & dT(d1,ir)]                        0.288
```
Obviously, a naive evaluation algorithm would run into an infinite loop, as indicated in the event expression for `d1`. However, the underlying evaluation algorithm of probabilistic Datalog can handle these cases correctly [14] (by stopping when a fixpoint is reached).

Likewise, `?- about(D,ir) & about(D,db)` would produce
```
d1 [dT(d1,ir) & dT(d1,db)]                                  0.450
d3 [l(d3,d1) & dT(d1,ir) & l(d3,d1) & dT(d1,db)]            0.360
d2 [l(d2,d3) & l(d3,d1) & dT(d1,ir) & dT(d1,db)]            0.144
```

Computation of probabilities for event expressions. Since event expressions can become rather complex, we need a method for computing the corresponding probability for any Boolean combination of event keys in a correct way. For that, we can apply the following method:

1. Transformation of the event expression into disjunctive normal form
2. Application of the inclusion-exclusion ('sieve') formula.

The latter is a generalization of the method for handling two conjuncts: $P(a \vee b) = P(a) + P(b) - P(a \wedge b)$. In the general case, where c_i denotes a conjunct of event keys (as a result of the first step), we have to compute the following alternating sum:

$$P(c_1 \vee \ldots \vee c_n) = \sum_{i=1}^{n} (-1)^{i-1} \sum_{1 \le j_1 < \ldots < j_i \le n} P(c_{j_1} \wedge \ldots \wedge c_{j_i}).$$

Unfortunately, this formula has exponential complexity. However, there are methods for identifying the cases where extensional semantics computes the correct result [4], thus the sieve formula is used only when necessary.

2.2 Interpretation of Probabilistic Weights

The interpretation of the probabilities is based on a *possible worlds semantics*. Here we have a set of worlds $\mathcal{W} = \{W_1, \ldots, W_n\}$, where each world W_j has a so-called probability of accessibility $P(W_j)$, such that $\sum_{i=1}^{n} P(W_i) = 1$. Each world contains a deterministic relational database. For computing the probability with which a formula (tuple) is true, we have to sum up the probabilities of those words in which the formula holds.

As a simple example, the probabilistic database containing the single tuple
`0.9 docTerm(d1,ir).`
has as possible interpretation
$P(W_1) = 0.9$: {`docTerm(d1,ir)`}
$P(W_2) = 0.1$: {}

When we have more than one tuple, then there are different possible interpretations. For the example

```
0.6 docTerm(d1,ir). 0.5 docTerm(d1,db).
```

there are, among others, the following interpretations:

I_1: $P(W_1) = 0.3$: {`docTerm(d1,ir)`}
$P(W_2) = 0.3$: {`docTerm(d1,ir), docTerm(d1,db)`}
$P(W_3) = 0.2$: {`docTerm(d1,db)`}
$P(W_4) = 0.2$: {}

I_2: $P(W_1) = 0.5$: {`docTerm(d1,ir)`}
$P(W_2) = 0.1$: {`docTerm(d1,ir), docTerm(d1,db)`}
$P(W_3) = 0.4$: {`docTerm(d1,db)`}

I_3: $P(W_1) = 0.1$: {`docTerm(d1,ir)`}
$P(W_2) = 0.5$: {`docTerm(d1,ir), docTerm(d1,db)`}
$P(W_3) = 0.4$: {}

Here *probabilistic logic* would take a cautious approach and allow all these interpretations (and many others); by considering the extreme cases I_2 and I_3, we can infer that

$$P(\texttt{docTerm(d1, ir)}\&\texttt{docTerm(d1, db)}) \in [0.1, 0.5].$$

In contrast, probabilistic Datalog assumes that the underlying probabilistic events are all independent (unless specified otherwise), which is only true for interpretation I_1 here, and so we get a point estimate of 0.3 for the cooccurrence of the two events.

2.3 Extensions

Disjoint Events. In some IR applications, we need disjoint probabilistic events. As an example, assume that we are performing information extraction on a text talking about Paris, and we have no further clue which city is referred to here. We only have general knowledge that in 70% of all cases, Paris refers to the French capital, in 20% to the city in Texas and in 10% to Paris, Idaho. This knowledge could be mapped by the following relation of tuples with disjoint events; thus, in the corresponding interpretation, in each world, only one tuple belongs to the relation CiSt:

CiSt

β	City	State
0.7	Paris	France
0.2	Paris	Texas
0.1	Paris	Idaho

$P(W_1) = 0.7$: {`CiSt(paris, france)`}
$P(W_2) = 0.2$: {`CiSt(paris, texas)`}
$P(W_3) = 0.1$: {`CiSt(paris, idaho)`}

As a consequence, we have to consider the disjointness of events when computing the final probability from the event expression, like e.g. $P(\texttt{CiSt(paris, france)}$ & $\texttt{CiSt(paris, texas)}) = 0$

Relational Bayes. In some IR applications, the probabilistic weights are not given beforehand, they have to be derived from deterministic facts. Actually, all probabilistic indexing methods start from some deterministic facts (like e.g. tf·idf weighting). Thus it would be nice if we could formulate these weighting methods also as Datalog rules. The *Relational Bayes* described in [15] does exactly this job.

As a starting point, we regard an example similar to the previous one, where we now have a deterministic database of cities and their nationality observed in a text corpus.

nationality_and_city	
Nationality	City
"British"	"London"
"British"	"London"
"British"	"London"
"Scottish"	"London"
"French"	"London"
"German"	"Hamburg"
"German"	"Hamburg"
"Danish"	"Hamburg"
"British"	"Hamburg"
"German"	"Dortmund"
"German"	"Dortmund"
"Turkish"	"Dortmund"
"Scottish"	"Glasgow"

$\Longrightarrow$

nationality_city		
P(Nationality|City)	Nationality	City
0.600	"British"	"London"
0.200	"Scottish"	"London"
0.200	"French"	"London"
0.500	"German"	"Hamburg"
0.250	"Danish"	"Hamburg"
0.250	"British"	"Hamburg"
0.667	"German"	"Dortmund"
0.333	"Turkish"	"Dortmund"
1.000	"Scottish"	"Glasgow"

The mapping onto probabilities is performed by computing the conditional probabilities of nationalities conditioned on cities, which we express in probabilistic Datalog as follows:

```
1  # P(Nationality | City):
2  nationality_city  SUM(Nat, City) :−
3          nationality_and_city (Nat, City)  |  (City);
```

Here the conditioning operator | generates a uniform probabilistic distribution over all the tuples having the same value for the attribute we condition on (here: City), then the SUM operator groups by the attributes specified as argument (here: (Nat, City)), summing up the probabilities of tuples having the same values for these attributes.

As an application to IR, we can use this method for computing a simple form of tf weights, as shown in the following example:

term	
Term	DocId
sailing	doc1
boats	doc1
sailing	doc2
boats	doc2
sailing	doc2
east	doc3
coast	doc3
sailing	doc3
sailing	doc4
boats	doc5

p_t_d_space(Term, DocId) :- term(Term, DocId) | (DocId);		
$P(t\|d)$	Term	DocId
0.50	sailing	doc1
0.50	boats	doc1
0.33	sailing	doc2
0.33	boats	doc2
0.33	sailing	doc2
0.33	east	doc3
0.33	coast	doc3
0.33	sailing	doc3
1.00	sailing	doc4
1.00	boats	doc5

p_t_d SUM(Term, DocId) :- term(Term, DocId) | (DocId);		
$P(t\|d)$	Term	DocId
0.50	sailing	doc1
0.50	boats	doc1
0.67	sailing	doc2
0.33	boats	doc2
0.33	east	doc3
0.33	coast	doc3
0.33	sailing	doc3
1.00	sailing	doc4
1.00	boats	doc5

Probabilistic Rules. Another useful extension of probabilistic Datalog are probabilistic rules. We start with rules for deterministic facts, stating that 70% of all men like sports, but only 40% of all women:

```
0.7 likes-sports(X) :- man(X). 0.4 likes-sports(X) :- woman(X).
man(peter).
```

Knowing for certain that peter is a man, the corresponding interpretation is as follows:
$P(W_1) = 0.7$: {`man(peter), likes-sports(peter)`}
$P(W_2) = 0.3$: {`man(peter)`}

Things get more complex when we apply probabilistic rules on uncertain facts, e.g. when we don't know `jo`'s gender:

```
# gender is disjoint on the first attribute
0.7 l-sports(X) :- gender(X,male).
0.4 l-sports(X) :- gender(X,female).
0.5 gender(X,male) :- human(X).
0.5 gender(X,female) :- human(X).
human(jo).
```

In probabilistic Datalog, the correct interpretation in this case is (see [8]):
$P(W_1) = 0.35$: {`gender(jo,male), l-sports(jo)`}
$P(W_2) = 0.15$: {`gender(jo,male)`}
$P(W_3) = 0.20$: {`gender(jo,female), l-sports(jo)`}
$P(W_4) = 0.30$: {`gender(jo,female)`}
Thus, for `l-sports(jo)`, we have to sum the probabilities of the two worlds where this fact holds: $P(W_1) + P(W_3) = 0.55$

Another problem with probabilistic rules occurs when multiple rules derive the same fact, like in the following example:

```
sameauthor(D1,D2) :-
author(D1,X) & author(D2,X). 0.5 link(D1,D2) :- refer(D1,D2).
0.2 link(D1,D2) :- sameauthor(D1,D2).
```

In case we have two documents written by the same author and referring to each other, we might wonder about the probabilistic weight of `link`:

```
?? link(D1,D2) :- refer(D1,D2) & sameauthor(D1,D2).
```

The problem is that given $P(l|r)$ and $P(l|s)$, this does not yield enough information on how to compute $P(l|r \wedge s)$. Thus, this probability has to be specified explicitly, like in the following form:

```
0.7 link(D1,D2) :- refer(D1,D2) & sameauthor(D1,D2).
0.5 link(D1,D2) :- refer(D1,D2) & not(sameauthor(D1,D2)).
0.2 link(D1,D2) :- sameauthor(D1,D2) & not(refer(D1,D2)).
```

In fact, this form corresponds to *probabilistic inference networks* [13], where our rules define a so-called link matrix.

3 Vague Predicates

Vague Predicates play an important role when users are searching for objects with certain attributes (e.g. in online shops), but have only soft constraints on

these facts, like e.g. searching for an LCD TV with high contrast and wide viewing angle, but for a reasonable price. Also, if the user does not know about standard sizes (e.g. asking for a 45 inch screen), this calls for a vague interpretation of this specification, instead of returning an empty answer set. In the following, we first discuss the underlying logical issue, and then present a solution that extends pD.

The Logical View on Vague Predicates. Current IR systems are based on proposition logic—a query term is present or absent in document, and thus either true or false (with a certain probability) in a document. Usually, similarity of values (e.g. similar terms) is not considered in standard text retrieval. On the other hand, multimedia IR deals with similarity already, like e.g. similarity of images, music or video (i. e. features thereof). In order to deal with these issues from a logical point of view, the transition from propositional to predicate logic becomes necessary.

In the previous sections of this article, we talked about (probabilistic) databases and Datalog, which are already based on predicate logic. So it seems quite natural to extend these formalisms to deal with similarity of values and vague predicates. The underlying ideas have been described in [9,8].

To illustrate these ideas, let us go back to the search for a 45 inch LCD TV. We assume that vague predicates are implemented as builtin predicates, e.g. in the form $\approx (X, Y)$. Then our query could be formulated as

```
query(D):- category(D,tv) & type(D,lcd) & size(D,X) & ≈(X,45)
```

For illustration purposes, here we represent the builtin predicate as a table shown below. With this interpretation, the system would be able to return the existing devices with sizes of 46 and 42 inches, although with a reduced certainty.

X ≈ Y

β	...	0.7	0.8	0.9	1.0	0.9	0.8	...
X	...	42	43	44	45	46	47	...
Y	...	45	45	45	45	45	45	...

Vague Predicates in IR and Databases. There are many applications where the concept of vague predicates may be helpful. From a more formal point of view, we have various data types in a database, with a set of vague predicates for each data type. Here are some examples: When we have texts in various languages, then each language may be regarded as a specific data type, where language-specific stemming methods can be regarded as vague predicates. When we are searching for proper names (e.g. persons, companies or products) then phonetic similarity as well as spelling-tolerant search may be useful. For dates (e.g. "the email I received about a month ago"), a vague date condition might often be useful, as well as for amounts ("a TV set for up to 500 Euros"). Similar statements can be made for technical measurements ("at room temperature"), and the search for chemical formulas also involves certain concepts of similarity.

Overall, we see that vague criteria are very frequent in end-user querying of fact databases. However, as there is no appropriate support for them in SQL, this calls for the integration of IR methods.

Probabilistic Modeling of Vague Predicates. Once we want to use vague predicates, there is the problem of estimating the corresponding probabilistic weights. In the very beginning, one can define them in an ad-hoc way; but once we have a running system, we can use feedback data (e.g. clickthrough data) in order to derive better estimates. In [7], an approach based on machine learning has been proposed for this purpose. The basic idea is to construct a feature vector $\boldsymbol{x}(q_i, d_i)$ from the query value q_i and document value d_i for each query-document pair; a simple feature could e.g. be the relative difference between q_i and d_i. Once we have collected enough training samples, we can apply some probabilistic classification method, like e.g. logistic regression. Figure 5 shows two examples of logistic functions, where the symmetric one could be used for vague equality, and the other one as a vague interpretation of 'greater than'.

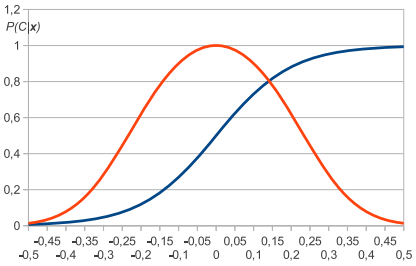

Fig. 5. Logistic functions

4 Expressiveness

The third major area where IR can benefit from DB concepts is expressiveness of the query language. Traditionally, IR has been focusing on the retrieval of relevant documents, where each document is regarded as a kind of independent, atomic unit. For this reason, there was hardly any need for an expressive query language. With the applications we are facing nowadays, however, there is a need for formulating more expressive queries. [1]

4.1 Retrieval Rules, Joins, Aggregations and Restructuring

In comparison to classical text retrieval as sketched above, pD already gives us a significant improvement in terms of expressiveness. Starting form logic rules like e.g. `about(D,T) :- docTerm(D,T)`, we are now able to consider document linking or anchor text (like in Web retrieval):

```
about(D,T) :- link(D1,D),about(D1,T).
```

[1] In many of today's IR applications, the required expressiveness is hard-coded in the application program; this approach corresponds to the very early days of DB development.

In case we have a thesaurus or an ontology, we can consider term hierarchy
`about(D,T) :- subconcept(T,T1) & about(D,T1).`

Also, it is possible to perform field-specific term weighting:
```
0.9 docTerm(D,T) :- occurs(D,T,title).
0.5 docTerm(D,T) :- occurs(D,T,body).
```

While these examples are just rules for methods provided by most of today's IR systems, more database oriented queries can also be formulated in pD, especially when we want to consider relationships between documents and other kinds of objects in the database. As shown above, joins allow for searching for authors writing about certain topics, like e.g.
`irauthor(N):- about(D,ir) & author(D,N).`
We can also ask more complex queries, e.g. Smith's IR papers cited by Miller:
`?- author(D,smith) & about(D,ir) & author(D1,miller) & cites(D,D1).`

docTerm

β	DNO	TERM
0.9	1	ir
0.8	1	db
0.6	2	ir
0.8	3	ir
0.7	3	ai

author

DNO	NAME
1	smith
2	miller
3	smith

irauths

1.7	smith
0.6	miller

Fig. 6. Example of probabilistic aggregation with summing

Another important element of query expressiveness is *aggregation*. If we want to know the names of the major IR authors, this could be formulated as
`irauthor(A):- docTerm(D,ir) & author(D,A).`
However, this form of aggregation through projection is not very meaningful, since a person with a single paper certainly about IR would get the same weight as another person with dozens of IR papers. Thus, we need some form of (probabilistic) counting, for which the relational Bayes mentioned above provides the necessary functions. Figure 6 shows the evaluation of the query
```
irauth(D,A):- docTerm(D,ir) & author(D,A).
irauths SUM(Name) :- irdbauth(Doc,Name) | (Name)
```

4.2 Expressiveness in XML Retrieval

So far, we have hardly talked about document structure (only about links between documents). In many IR applications, document structure plays an important role, and the documents of a (sub)collection have a quite regular structure (in contrast to Web documents). As XML is the most popular standard for representing document structure, here we discuss how we can exploit this structure for increasing precision in retrieval. Figure 7 gives a survey over the possible views on XML documents, which can also be regarded as a design space for XML IR systems. Here we distinguish two dimensions, namely the *structure* and *content*

type. The former deals with the structural aspects of XML documents, starting from a simple view as a tree (nested) structure up to the database-oriented view as implemented in the XQuery language. The second dimension deals with the types of content we may find in XML documents. In most cases, we assume all content to be text only. However, markup of an element may indicate a specific data type (e. g. a date) or even complex object types. In the following, we describe each of the two design dimensions in more detail.

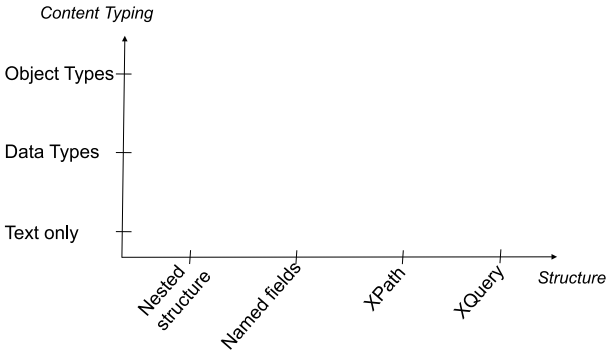

Fig. 7. Views on XML

XML Structure

Nested Structure. Whereas classical retrieval regards documents as atomic units, the XML markup of a document immediately implies a nested, tree-like structure. Following this view, a retrieval method should be able to retrieve subtrees (i. e. complete elements) instead of complete documents only. Typical query languages for this kind of retrieval provide no specific means for specifying structural constraints, in most cases they only allow for the specification of a set of terms. The corresponding retrieval method aims at performing a relevance-oriented selection of answer elements, i. e. the system should return the most specific relevant elements.

Named Fields. This view is somewhat orthogonal to the nested structure view: Here, we only regard the element names, without considering their structural relationships. Thus, a document can be seen as a set of named fields (sometimes also called a linear data model). Here we can refer to elements through field names only, whereas the context of an element is ignored (e. g., in a document, we may not be able to distinguish between the author of the document and that of a referenced paper). Another problem is that of false coordination: e. g., for a document with two authors from different institutions, our retrieval method may not be able to coordinate author names and affiliations correctly.

XPath. XPath provides full expressiveness for navigating through the document tree, by parent/child and ancestor/descendant relationships, whereas horizontal navigation is supported via operators like following/preceding, following-sibling and preceding-sibling; in addition, there are the attribute and the namespace axis. With these operators — in combination with the specification of element names — XPath allows for the selection of arbitrary elements.

XQuery. XQuery offers an even higher expressiveness than XPath, due to the fact that it was developed especially for database-like applications. Thus, in addition to XPath, it supports typical database operators like joins, aggregations and constructors for restructuring results.

As a simple example, assume a list of book titles with prices and publisher names stored in a file named 'bib.xml' ; then the following query would produce a list of publishers, each along with the average price of its books:

```
FOR $p IN distinct(document("bib.xml")//publisher)
LET $a := avg(document("bib.xml")//book[publisher = $p]/price)
RETURN
   <publisher>
      <name> {$p/text()} </name>
      <avgprice> {$$a} </avgprice>
   </publisher>
```

Here the `FOR` construct loops over all publishers, whereas the following `LET` retrieves all corresponding book prices and then computes their average. In the `RETURN` clause, the XML structure of the result is specified.

XML Content Typing. Now we regard the content dimension of XML retrieval.

Text. Most of today's XML retrieval systems assume that an XML document contains only text. In some sense, they still follow the traditional view of a document as a text block, which is now structured via XML tags.

Data Types. Different XML elements may contain different types of text, and this information could be exploited in retrieval. As discussed above, advanced IR system should support the notion of data types, where each such type is accompanied by a set of (vague) predicates.

Object Types. One can even go one step further and regard objects occurring in XML documents, like for example persons, locations or companies. Objects may have several attributes (of different data types), and queries may refer to any of these attributes. As an example, regard the following text excerpt from Wikipedia:

> Pablo Picasso (October 25, 1881 – April 8, 1973) was a Spanish painter and sculptor..... In Paris, Picasso entertained a distinguished coterie of

friends in the Montmartre and Montparnasse quarters, including André Breton, Guillaume Apollinaire, and writer Gertrude Stein.

If this text were marked up appropriately, a retrieval system should be able to answer queries like e. g. "To which other artists did Picasso have close relationships?"or "Where did he meet Gertrude Stein?". There is substantial work on named entity recognition methods, which allow for automatic markup of object types.

Overall, with data and object types, precision of XML retrieval can be increased.

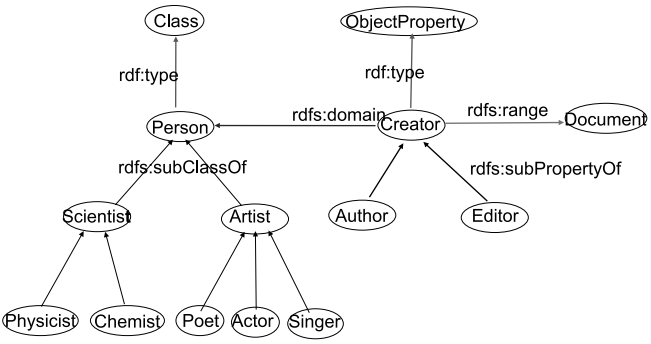

Fig. 8. OWL modeling

Towards Semantic Retrieval of XML Documents. In the discussion from above, we have not regarded the semantics of XML element names. In fact, some XML applications use rather cryptic element names. However, for new XML applications, the effort for using meaningful element names would be only marginal. Based on this information, the semantics of tag names could be exploited. In Figure 8, we have used OWL [12] for an example modelling of descriptions of artists and scientists (e. g. in Wikipedia). A first benefit would be the possibility to search for generalizations or specializations of concepts. (e. g. searching for artists would retrieve poets, actors and singers). In a similar way, also hierarchies on properties would be supported[2], and the domain and range of such properties can be considered

Readers familiar with XQuery and XPath Full Text [2] may have noticed that some of the features discussed here are already available in XQuery. However, weighting in XQuery is restricted to the text search part. Thus, probabilistic inference involving joins, rules or aggregations is missing.

[2] However, hierarchies of properties cannot be expressed in OWL.

5 Further Concepts

Here we want to point out two directions of ongoing research in the integration of IR and DB.

5.1 4-Valued (Probabilistic) Logics

In Section 2, we have demonstrated how we can enhance the inference capabilities of today's IR systems by using pD. Like in most logic-based approaches, this is only feasible if we have a consistent knowledge base. In IR, however, when dealing with large collections (from possibly heterogenous sources), it is inevitable that we introduce inconsistencies due to conflicting statements originating from different documents (e.g., there are many documents on the Web claiming that Barack Obama is a Muslim). Since we can derive anything from an inconsistent knowledge base, our standard logic-based approach is doomed to fail in such settings. The only way out is to use a different logical formalism, like e. g. four-valued logic [1]. In addition to the truth values *true* and *false*, there are also the values *unknown* and *inconsistent*. Then, for each statement, the probabilities for the four truth values add up to 1. In the Obama-Muslim example from above, a summarization over all relevant Web documents would yield a certain probability for *inconsistent*, besides a high probability for *false*.

Another benefit from 4-valued logic is that it allows for both open and closed world assumptions: Standard Datalog is based on the closed world assumption, i. e., if the system cannot infer a certain statement, then this statement is assumed to be *false*. For example, if we cannot derive `author(smith,doc123)`, then `smith` is not an author of the paper in question. For classical DB applications, this approach is very reasonable, assuming that a database is always complete and correct. On the other hand, for IR-oriented applications, a closed world assumption is often inappropriate. For example, if we are unable to infer `about(d123,ir)`, this does not mean that we are sure that this paper is not about IR. In fact, language models solve this problem by using a collection-based prior: if a term does not occur in a document, then a small default probability for the document being about the term is assumed (which is derived from the relative collection frequency of the term). From a logical point of view,[3] this situation should be modeled via an open world assumption, meaning that we cannot make a statement whether or not the document is about the term in question, represented as *unkown* in our four-valued logic. Further research will show if we can achieve reasonable benefits from an enhanced logical modeling.

4-valued probabilistic Datalog (with open and closed world assumptions) was introduced in [10], showing that the computational effort is roughly doubled in comparison to the two-valued case. In [11], this was extended towards an object-oriented logic, which allowed for the explicit modeling of contexts (e.g. documents) with accessibility probabilities. This idea was taken further in [6] for dealing with annotations, and in [5] for modeling summarization.

[3] Of course, the language model approach can be easily represented in pD, as shown in [15].

5.2 IR vs. DB Systems

Figure 9 shows a comparison between standard IR and DB systems. In the DB world, there is a clear separation between the DBMS and the application itself, while in the IR world, no such separation exists—the user interacts directly with the IR system. Besides the different architecture, this figure also highlights a major difference between IR- and DB-oriented research: In IR research, the pragmatic aspects of the application play an important role (which is, e.g., reflected in the central concept of relevance). On the other hand, in DB settings, all pragmatic aspects are delegated to the application component, which is mostly beyond the scope of DB research.

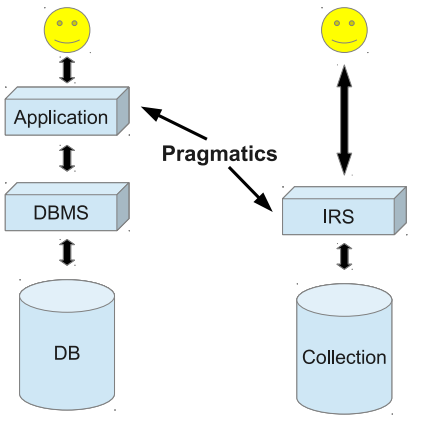

Fig. 9. Pragmatics in IR vs. DB

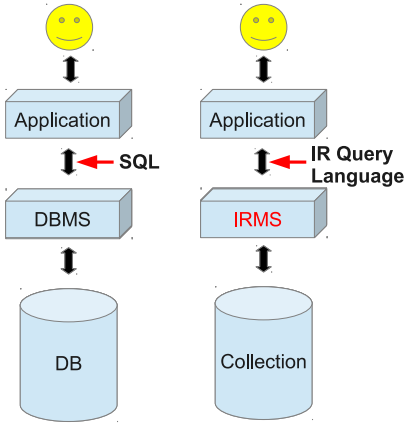

Fig. 10. Towards an IRMS

This figure might also stimulate another idea: Why should IR not switch to the DB-like architecture and introduce a separation between applications and IR management system? Given the broad variety of IR applications we are dealing with nowadays, building separate systems for each type of task is not very reasonable. Instead, it might be more effective to have a standardized IR management system (IRMS). On top this system, we can implement various applications (see Figure 10). Then of course, there is the question of the design of the interface between application and IRMS. In the DB world, SQL plays this role. In parallel, we would need an IR query language with comparable capabilities—the basic concepts of such a language have been described in this article.

6 Conclusion

In this article, we have focused on the logical view for bridging the gap between IR and DB. From this perspective, there are three major concepts that should

become part of IR, namely inference, vague predicates and expressive query languages.

Concerning inference, the logical view interprets IR as being based on uncertain inference, which is a generalization of the traditional DB approach. We have seen that the probabilistic relational model supports the integration of IR and DB, while pD yields more powerful inference mechanism; especially, pD allows for formulating retrieval strategies as logical rules, thus making IR systems much more flexible than the current approaches.

For dealing with various media and data types, vague predicates form a natural extension of IR methods to attribute values, thus switching from propositional to predicate logic. The probabilistic weights of vague predicates can be learned from feedback data.

Finally, a more expressive query language would allow for joins, for computing aggregated results, and for restructuring the result objects.

References

1. Belnap, N.: A useful four-valued logic. In: Modern Uses of Multiple-Valued Logic. Reidel, Dordrecht (1977)
2. Case, P., Dyck, M., Holstege, M., Amer-Yahia, S., Botev, C., Buxton, S., Doerre, J., Melton, J., Rys, M., Shanmugasundaram, J.: Xquery and xpath full text 1.0 (2011), `http://www.w3.org/TR/xpath-full-text-10/`
3. Ceri, S., Gottlob, G., Tanca, L.: Logic Programming and Databases. Springer, Heidelberg (1990)
4. Dalvi, N.N., Suciu, D.: Efficient query evaluation on probabilistic databases. VLDB J. 16(4), 523–544 (2007)
5. Forst, J.F., Tombros, A., Roelleke, T.: Polis: A probabilistic logic for document summarisation. In: Proceedings of the 1st International Conference on Theory of Information Retrieval (ICTIR 2007) - Studies in Theory of Information Retrieval, pp. 201–212 (2007)
6. Frommholz, I., Fuhr, N.: Probabilistic, object-oriented logics for annotation-based retrieval in digital libraries. In: Nelson, M., Marshall, C., Marchionini, G. (eds.) Opening Information Horizons – Proc. of the 6th ACM/IEEE Joint Conference on Digital Libraries (JCDL 2006), pp. 55–64. ACM, New York (2006)
7. Fuhr, N.: A probabilistic framework for vague queries and imprecise information in databases. In: Proceedings of the 16th International Conference on Very Large Databases, Los Altos, California, pp. 696–707. Morgan Kaufman (1990)
8. Fuhr, N.: Probabilistic Datalog: Implementing logical information retrieval for advanced applications. Journal of the American Society for Information Science 51(2), 95–110 (2000)
9. Fuhr, N., Rölleke, T.: A probabilistic relational algebra for the integration of information retrieval and database systems. ACM Transactions on Information Systems 14(1), 32–66 (1997)
10. Fuhr, N., Rölleke, T.: HySpirit – a probabilistic inference engine for hypermedia retrieval in large databases. In: Schek, H.-J., Saltor, F., Ramos, I., Alonso, G. (eds.) EDBT 1998. LNCS, vol. 1377, pp. 24–38. Springer, Heidelberg (1998)
11. Lalmas, M., Roelleke, T., Fuhr, N.: Intelligent hypermedia retrieval. In: Szczepaniak, P.S., Segovia, F., Zadeh, L.A. (eds.) Intelligent Exploration of the Web, pp. 324–344. Springer, Heidelberg (2002)

12. McGuinness, D.L., van Harmelen, F.: OWL. Technical report, World Wide Web Consortium (2004), http://www.w3.org/TR/owl-features/
13. Pearl, J.: Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. Morgan Kaufman, San Mateo (1988)
14. Rölleke, T., Fuhr, N.: Probabilistic reasoning for large scale databases. In: Datenbanksysteme in Büro, Technik und Wissenschaft (BTW 1997), pp. 118–132. Springer, Heidelberg (1997)
15. Rölleke, T., Wu, H., Wang, J., Azzam, H.: Modelling retrieval models in a probabilistic relational algebra with a new operator: the relational Bayes. The International Journal on Very Large Data Bases (VLDB) 17(1), 5–37 (2007)
16. Suciu, D., Olteanu, D., Ré, C., Koch, C.: Probabilistic Databases. Synthesis Lectures on Data Management. Morgan & Claypool Publishers (2011)
17. Ullman, J.D.: Principles of Database and Knowledge-Base Systems, vol. I. Computer Science Press, Rockville (1988)
18. van Rijsbergen, C.J.: Information Retrieval, 2nd edn. Butterworths, London (1979)