An Algebra for probabilistic XML retrieval

Benjamin Piwowarski
LIP6, University Paris 6
8, rue du capitaine Scott
75015 Paris, France
bpiwowar@poleia.lip6.fr

Patrick Gallinari
LIP6, University Paris 6
8, rue du capitaine Scott
75015 Paris, France
gallinar@poleia.lip6.fr

ABSTRACT

In this paper, we describe a new algebra for XML retrieval. We first describe how to transform an XPath-like query in our algebra. The latter contains a vague predicate, `about`, which defines a set of document parts within an XML document that fullfill a query expressed as in "flat" Information Retrieval – a query that contains only constraints on content but not on structure. This predicate is evaluated in a probabilistic way: we thus need a probabilistic interpretation of our algebra. Answers to query needs expressed with vague content and vague structure constraints can then be evaluated.

1. INTRODUCTION

With the expansion of the Web and of large textual resources like electronic libraries, appeared the need for new textual representations allowing interoperability and providing rich document descriptions. Several structured document representations and formats were then proposed during the last few years together with description languages like XML. This allows for richer descriptions with the incorporation of metadata, annotations and multimedia information. Document structure is an important source of evidence, and should be considered together with textual content for information access tasks. Information retrieval engines should be able to cope with the complexity of new document standards so as to fully exploit the potential of these representations and to provide new functionalities for information access. For example, users may need to access some specific document part, navigate through complex documents or structured collections; queries may address both metadata and textual content.

In INEX (Initiative for the Evaluation of XML retrieval), queries are expressed in a query language (NEXI) which is very similar to XPath. A *vague* operator (about) is introduced in order to allow for queries in a similar fashion than in information retrieval, that is queries that only contain contraints on what the retrieved element should be *about*, but which do not contain any contraint on structural relationship constraints an element must fullfill. Such languages can be used to express query needs that mix possibly vague content and structure constraints. XPath is for XML documents what SQL for databases is: a language that describes which information should be retrieved from XML documents. In traditional databases, this request is usually mapped into an algebra which in turn is used as a query plan. This query plan is closely related to physical operations that will give the answers to the query. In databases, the result of a formula of the algebra is a set of tuples. In XML databases, the result is a set of elements.

Defining or choosing an algebra is a very important issue if one wants to answer complex query needs. This is proven in many papers of the semi-structured database field [4, 1]. Such approaches are also used in INEX [3]. As in classical IR, XML IR aims at retrieving the set of document elements that fullfill a given query need. This query need is by definition imprecise. The algebra we define in this paper can be used to answers *vague* queries that have constraints on both content and structure and fits into our Bayesian Networks framework [5] we are using for "Content Only" queries – queries that have no constraint on structure.

Our algebra is closely related to the algebra defined by Fuhr and Grossjohan [2] which we considered before defining our own algebra for XML retrieval. However, their algebra has some drawbacks. Firstly, it is too much closely related to XPath and to the datalog approach they use. Secondly, the algebra they define is somewhat complex and cannot be easily extended to cope with new task like vague constraints on structure. However, we kept their idea of defining the relevance of a document element[1] as an event: the probability that a doxel is an answer is the probability of the event. This probability can be computed by a probabilistic model.

The interest is more explicit when one consider an example. Let us suppose that we are searching doxels that are (1) *about* `cats`. This query need can also be expressed as the set of doxels that are (2) the parents of children *about* `cats`. Let $R(\texttt{cat})$ denote the set of doxels that are *about* `cats`. Then the answers to the original query need (1) is the set $R(\texttt{cat})$ while the answers to the modified query need (2) is the set $\text{pa}(\text{child}(R(\texttt{cat})))$[2], the parent of children of

TDM'04, the first Twente Data Management Workshop on XML Databases and Information Retrieval, Enschede, The Netherlands

[1] we will use the word "doxel" in the remainder of the paper in place of "document element"
[2] See the table 2 for a description of notations.

elements which are about cats. When the set $R(\texttt{cat})$ is defined by a distribution of probability, we must not evaluate too quickly the probability that an XML element belongs to the set (2). Let us suppose that a doxel x has two children y_1 and y_2, and that the probability of a belonging to $R(\texttt{cat})$ is p_a. Then, we can assume that the probability that y_1 (or y_2) belongs to child($R(\texttt{cat})$) is also p_a. The probability that x is the parent of a doxel in child($R(\texttt{cat})$) is $P(y_1 \in R(\texttt{cat}) \vee y_2 \in R(\texttt{cat}))$. if we use the classical "noisy-or" to compute the probability that either y_1 or y_2 is such a children, then the probability that a is an answer to our query need is $1 - P(y_1 \notin R(\texttt{cat}))P(y_2 \notin R(\texttt{cat})) = 1 - (1 - p_a)^2 \neq p_a$. We thus need to keep the information that p_a "comes from" the relevance of a to the query *about* `cats`.

The paper is organised as follows. In section 2 we briefly define the subset of XPath which is the basis of our query language. We then show how we transform a query need expressed in an XPath-like language into our algebra in section 3. As queries are *vague*, we define how to compute a probability that a doxel is an answer - that is, that a doxel is in the set defined by a formula of our algebra - in section 4.

2. QUERY LANGUAGE

In this section, we define briefly the part of XPath we use. We also define notations that will be used in the remainder of the paper.

Components of an `XPath`

An XPath is read from left to right. The different components are separated by a slash `/` which is not within brackets. For example, we decompose `/A1::A2[B1::B2/C1::C2 and F1::F2]/D1::D2/E1::E2` in three parts, `/A1::A2[B1::B2/C1::C2 and F1::F2]`, `/D1::D2` and `/E1::E2`.

Each component is itself composed of three parts:

1. The *axis* (before `::`). The axis can take the following values and define a set for a doxel x:
   `/child` (the set of the children of x),
   `/descendant` (the set of all descendants of x),
   `/attribute` (the set of the attributes of x),
   `/self` (x itself),
   `/descendant-or-self` (x itself and its descendants),
   `/following-sibling` (the next siblings of x),
   `/following` (the next doxels with respect to the document order),
   `/parent` (its parent),
   `/ancestor` (its ancestors),
   `/preceding-sibling` (its preceding siblings),
   `/preceding` (the previous doxels with respect to the document order),
   `/ancestor-or-self` (x or its ancestors) ;
2. The *label* (after `::`).
3. The *filter* (between brackets) that express a boolean condition on doxels.

The axis and the label are very often grouped together in order to shorten the XPath expression. The list of abbreviated syntaxes is shown in table 1. In the remainder of the paper, we use this abbreviated syntax as often as possible so as to shorten the query needs expressed in XPath.

Usual syntax	Abbreviated syntax
`child::a`	`a`
`/child::a`	`/a`
`child::a`	`a`
`/child::*`	`/*`
`child::*`	`*`
`/attribute::a`	`/@a`
`self::a`	`a`
`/attribute::*`	`/@*`
`attribute::a`	`/@a`
`/parent::*`	`/..`
`parent::*`	`..`
`/self::*`	`/.`
`self::*`	`.`
`/descendant-or-self::a`	`//a`
`/descendant-or-self::*`	`//*`

Table 1: Abbreviated syntax of XPath. In the table, `a` is a label (XML tag name).

As we use a restricted XPath, we will suppose that each component of our query language is defined as follows:

The axis selects a set of doxels. For the first component of the XPath, this set is defined with respect to the document d. For the first component of an XPath within a filter, the set of selected doxels is evaluated with respect to the document d or to the filtered doxel. For any another component, the selection is made with respect to the set of doxels selected by the previous component.

The label `a` filters the set of doxels selected by the axis and keep only those which have the label `a`. When the label is `*`, the "axis" set is not filtered. Then,

the filter returns the subset of those doxels which fullfill the boolean conditions expressed in the filter.

The filter is a boolean expression which defines the conditions that a doxel must fullfill. An XPath can be used in the filter: it is defined with respect to the doxel for which the condition is evaluated (if the path does not begin with `/`) or with respect to the document root d (if the path begins with `/`). In our subset of XPath, a filter is a condition which can be true or false for one doxel. For a given doxel x, we define recursively the following filters:

- f_1 `and` f_2 which is true if f_1 are f_2 true ;
- f_1 `or` f_2 which is true f_1 or f_2 are true ;
- `not` f which is true if f is not true ;
- `x`, where `x` is an XPath, which is true if the set of elements defined by `x` is not empty.
- `about`$(\texttt{x}, q)$ which is true if the set of elements defined by `x` answer the query need q. It is possible to define other predicates (for other medias for example).

- $\mathtt{x} = A$ (where $\mathtt{x}$ is an XPATH and A is a constant (text, number, etc.) which is true if one of the doxels of the set defined by $\mathtt{x}$ has a content which is equal to A. The operators ($\neq$, $>$, $<$, ...) are also defined for some specific datatypes like number, strings, etc.

An XPATH-like language can be used to define a query need with constraints on both structure and content in XML documents. In the next section, we show how we transform an XPATH into our algebra. The result will be a function that returns the set of doxels in a given document that are the answers to the query need. In the remainder of the paper, we will refer to the three examples defined in figure 1.

3. THE ALGEBRA

In this section, we give an explicit way to transform any query need expressed in an XPATH-like language into an algebra which is defined on the parts of the set of doxels. Besides classical operators of the set theory like the intersection, the union and the complement, we use structural operators which are defined in table 2. We denote $\mathcal{X}$ the set of all doxels.

	Notation	Relation	Notation
Parent	$\text{pa}_?(x)$	Parent and self	$\overline{\text{pa}}_?(x)$
Ancestors	$\text{anc}_?(x)$	... and self	$\overline{\text{anc}}_?(x)$
Children	$\text{child}_?(x)$	... and self	$\overline{\text{child}}_?(x)$
Descendants	$\text{desc}_?(x)$	... and self	$\overline{\text{desc}}_?(x)$
Siblings	$\text{siblings}_?(x)$	... and self	$\overline{\text{siblings}}_?(x)$
Preceding[a]	$\text{preceding}_?(x)$	... and self	$\overline{\text{preceding}}_?(x)$
Following[b]	$\text{following}_?(x)$	... and self	$\text{following}_?(x)$

[a] wrt the document order
[b] wrt the document order

Table 2: Relationships in an XML document. When the operator have a subscript, the result set is restricted to the subscript set: "?" is either / (for XML elements) or @ (for XML attributes). These operators are defined on the power set $\mathcal{P}(\mathcal{X})$ of doxels $\mathcal{X}$ and take their values in the same set: for example, $\text{pa}(A)$ is the set of all the doxels which have a child in A.

In the remainder of the paper, we will restrict the evaluation of an XPATH to a given document d. We need to introduce three new functions:

1. $R(q)$ which returns the set of doxels which are answers to the query need q. We first consider that this set is "strict" (an element is in the set or not), but latter we will consider this set as *vague*. That is, a doxel is in the set with a given probability.
2. $terms\,(x)$ which returns the textual content of a doxel and the function. The $terms^{-1}\,(t)$ then returns the set of doxels that have the content t.
3. $label\,(x)$ which returns the label of the doxel (the tag name). The function $label^{-1}\,(l)$ returns the set of doxels which have a label l.

The algebra is defined on the set $\mathcal{P}(\mathcal{X})$ (the set of all the part of the set of doxels). We use the operator $\circ$ to compose the different functions defined on $\mathcal{P}(\mathcal{X})$ which take values in $\mathcal{P}(\mathcal{X})$. We also use the intersection, the union and the complement. A constant in the algebra is thus a subset of the set of doxels $\mathcal{X}$. For example, the formula $\text{pa}(R(\mathtt{cat}) \cap R(\mathtt{black}))$ is the composition of two intersection functions of two constants ($R(...)$) with the function pa that returns the parents of a set of doxels.

An XPATH $\mathtt{x}$ is composed of n components:

$$\mathtt{x} = (\mathtt{x}_1, \ldots, \mathtt{x}_n) \text{ where } \mathtt{x}_i = (a_i, l_i, f_i)$$

where a_i is the axis, l_i the label and f_i the filter of the i^{th} component. For example, **I3** is defined by the sequence:
(`/child`, `document`,`@year = 2002 and .//image and about(title,"cat")`),
(`descendant-or-self`,`p`,`about(.,black)`).
Each part is a component that can be processed separately and is transformed into a function $f : \mathcal{P}(\mathcal{X}) \rightarrow \mathcal{P}(\mathcal{X})$:

- The axis a_i is transformed into a structural operator: for example, `/child` is transformed into child.
- The label (or a set of labels) l_i is transformed into a function that selects a subset of doxels which have a label in the set l_i.
- The transformation f_i of the filter is more complex and is described latter. Briefly, we process the filter by defining the set of doxels that fullfill the condition(s) expressed in the filter. The filter is then defined as the intersection of the latter with the set of doxels we have to filter (that is, which have been selected by the axis and filtered by the label).

We now define more formally how to transform a component in a function which maps a set of doxels to another set of doxels. This transformation is based on three transformations that all return (except $\Psi_A^{(0)}$) functions $f : \mathcal{P}(\mathcal{X}) \rightarrow \mathcal{P}(\mathcal{X})$:

- The transformation of the axis is denoted Ψ_A;
- The transformation of the label Ψ_L;
- The transformation of the filter Ψ_F.

With these notations, the XPATH $\mathtt{x}$ can be defined as the result of the evaluation of the following algebraic expression:

$$\begin{aligned}\Psi(d, \mathtt{x}) \quad = \quad & \Psi_F(f_n) \circ \Psi_L(l_n) \circ \Psi_A(a_n) \\ & \circ \cdots \\ & \circ \Psi_F(f_2) \circ \Psi_L(l_2) \circ \Psi_A(a_2) \\ & \circ \Psi_F(f_1) \circ \Psi_L(l_1) \circ \Psi_A^{(0)}(d, a_1)\end{aligned}$$

For XPATH within filters, $\Psi_A^{(0)}(a_1)$ is replaced by $\Psi_{AF}^{(0)}$ in the last line. In the remainder of this section, we focus on the transformation of each component separately.

The axis

Let us begin with the axis transformation of the first component of an XPATH, which associate a function from $\mathcal{P}(\mathcal{X})$

I1 Doxels that are *about* cats
```
//*[about(.,"cat")]
```

I2 Images in paragraphs about *about* cat pictures
```
//image[../p[about(.,"cat pictures")]]
```
or
```
//p[about(.,"cat pictures")]/image
```

I3 Paragraphs *about* black in a document written in 2002 which title is *about* cat and that contains a picture
```
/document[@year = 2002 and .//image and about(title,"cat")]//p[about(.,"black")]
```

Figure 1: Original queries

Axis	Transformation $\Psi_A^{(0)}$ and $\Psi_{AF}^{(0)}$
/child	d
/descendant	$\overline{\text{desc}_/}(d)$
/descendant-or-self	$\overline{\text{desc}_/}(d)$
Only within a filter $\Psi_{AF}^{(0)}$	
child	$\text{child}_/$
descendant	$\text{desc}_/$
attribute	$\text{desc}_/ \cap \mathcal{X}_@$
self	Id
descendant-or-self	$\overline{\text{desc}_/}$
following-sibling	$\text{following}_/ \cap \text{siblings}$
following	$\text{following}_/$
parent	$\text{pa}_/$
ancestor	$\text{anc}_/$
preceding-sibling	$\text{preceding}_/ \cap \text{siblings}$
preceding	$\text{preceding}_/$
ancestor-or-self	$\overline{\text{anc}_/}$

Table 3: Sets and XPath (initialisation and axis). The transformation of an axis (left part) is given by the right column of the table. The functions are either constants (for example, x or $\text{desc}(x)$) or structural operators. An intersection $I = \varphi_1 \cap \varphi_2$ is defined in the function set $\mathcal{X} \mapsto \mathcal{P}(\mathcal{X})$ in the following manner: $I : x \in \mathcal{X} \mapsto \varphi_1(x) \cap \varphi_2(x)$.

to $\mathcal{P}(\mathcal{X})$ to an axis. The simplest (constant) is $\Psi_A^{(0)}$ which selects doxels in the document (table 3, top). Within a filter, this function is $\Psi_{AF}^{(0)}$, which selects a set of doxels with respect to the evaluated doxel. When the axis is not a part of the first component, then the table 4 gives the transformation.

Axis	Transformation Ψ_A
/child	$\text{child}_/$
/descendant	$\text{desc}_/$
/attribute	$\text{desc}_@$
/self	$x_/$
/descendant-or-self	$\overline{\text{desc}_/}$
/following-sibling	$\text{following}_/ \cap \text{siblings}$
/following	$\text{following}_/$
/parent	$\text{pa}_/$
/ancestor	$\text{anc}_/$
/preceding-sibling	$\text{preceding}_/ \cap \text{siblings}$
/preceding	$\text{preceding}_/$
/ancestor-or-self	$\overline{\text{anc}_/}$

Table 4: Set and XPath (axes). The transformation of an axis (left part) is given by the right column of the table.

The label

The transformation of a label l is simple, as it is reduced to the evaluation of the intersection of the set of doxels which have a label l and the set of evaluated doxels:

$$\begin{aligned} \Psi_L(l): \quad \mathcal{P}(\mathcal{X}) &\mapsto \mathcal{P}(\mathcal{X}) \\ X &\mapsto X \cap label^{-1}(l) \end{aligned}$$

where we handle the special case of $*$ by defining $label^{-1}(*) = \mathcal{X}$.

The filter

Finally, we have to define how to transform a filter. As stated before, we first have to define the set of all doxels that fullfill the filter, and then we only have to take the intersection between that set and the set of doxels for which we have to evaluate the condition within the filter. The function which transforms a filter in the set of of doxels that fullfill the filter is denoted Ψ'_F. Then:

$$\begin{aligned} \Psi_F(f): \quad \mathcal{P}(\mathcal{X}) &\mapsto \mathcal{P}(\mathcal{X}) \\ X &\mapsto X \cap \Psi'_F(f) \end{aligned}$$

To define Ψ'_F, we have to introduce first what we call a

"pseudo inverse" of a function $\varphi : \mathcal{P}(X) \mapsto \mathcal{P}(X)$. It is defined as follows:

$$\begin{aligned} \varphi^* : \quad \mathcal{P}(\mathcal{X}) &\mapsto \mathcal{P}(\mathcal{X}) \\ X &\mapsto \{x/x \in \mathcal{X}, \varphi(x) \cap X \neq \emptyset\} \end{aligned}$$

It can be shown easily that the pseudo inverse of a constant function is the same constant. For the other structural operators, the pseudo inverse is given in table 5: the pseudo inverse of a structural operator is itself a structural operator.

Function	Pseudo inverse
pa	child
anc	desc
child	pa
desc	anc

Table 5: Pseudo inverse of structural operators

We now have to demonstrate that it behaves like the inverse with respect to the composition operator

$$(\varphi_1 \circ \varphi_2)^* = \varphi_2^* \circ \varphi_1^*$$

For all $X \subseteq \mathcal{X}$,

$$\begin{aligned} & x \in \varphi_2^* \circ \varphi_1^*(X) = \{x/x \in \mathcal{X}, \varphi_2(x) \cap \varphi_1^*(X) \neq \emptyset\} \\ & \Leftrightarrow \exists x' \in \mathcal{X}, x' \in \varphi_2(x) \cap \varphi_1^*(X) \\ & \Leftrightarrow \exists x' \in \mathcal{X}, x' \in \varphi_2(x) \cap \{x/x \in \mathcal{X}, \varphi_1(X) \cap X \neq \emptyset\} \\ & \Leftrightarrow \exists x' \in X, \exists x'' \in \varphi_2(x), x' \in \varphi_1(x'') \cap X \\ & \Leftrightarrow \exists x' \in \mathcal{X}, x' \in \varphi_1 \circ \varphi_2(x') \cap X \\ & \Leftrightarrow x \in (\varphi_1 \circ \varphi_2)^*(X) \end{aligned}$$

We can also remark that if $\varphi_1 = X' \cap \varphi_2$, then

$$\varphi_1^*(X) = \varphi_2(X \cap X')$$

Filter f	Function Ψ'_F
`x`	$\Psi(\mathtt{x})^*(\mathcal{X})$
`about(x,`q`)`	$\Psi(\mathtt{x})^*(R(q))$
$\mathtt{x} = A$	$\Psi(\mathtt{x})^*(terms^{-1}(A))$
f_1 `and` f_2	$\Psi'_F(f_1) \cap \Psi'_F(f_2)$
f_1 `or` f_2	$\Psi'_F(f_1) \cup \Psi'_F(f_2)$

Table 6: Filter transformation

The table 6 shows how to use the pseudo inverse in order to transform the filter:

- For the filter `x` which is reduced to an XPath expression, an element a is in the filtered set if and only if there exists an element defined by `x` relatively to a, that is if $\Psi(\mathtt{x})(a) \neq \emptyset \Leftrightarrow a \in \Psi(\mathtt{x})^*(\mathcal{X})$.
- For the filter `about(x,`q`)`, an element a is in the filtered set iff there exists an element defined by `x` relatively to a which is also in $R(q)$, that is if $a \in \Psi(\mathtt{x})^*(\mathcal{X}) \cap R(q) \Leftrightarrow a \in \Psi(\mathtt{x})^*(R(q))$.
- The filter $\mathtt{x} = A$ is similar to the previous filter.
- The operator *and* (resp. *or*) is simply defined as the intersection (resp. the union) of elements that fullfill the first and the second filters.

In figure 2, we show how the queries of figure 1 are transformed into our algebra. Please note that our algebra may be optimised: formulas can be transformed like in all algebras in order to be more efficiently evaluated by the search engine. This case is illustrated by the case I2bis in figure 2: the formula reduces to the formula of I2, despite of a more complex XPath used to express the query need. However, our main concern is to define an algebra for XML *vague* retrieval and we did not design with efficiency in mind.

4. PROBABILISTIC INTERPRETATION

In the previous section, we used a predicate *about* which is transformed into the function $R(q)$ that returns the set of doxels that are answers to the query q. In Information Retrieval (IR), the answers to a query are not well defined: the query is expressed in vague terms, and the real query need cannot be easily defined. We thus have to define $R(q)$ as a "*vague*" set in order to compute the answer to a query that contains predicates like *about*.

In our approach, as in the probabilistic interpretation of fuzzy sets [6], a set $A \subset X$ is not anymore defined strictly. We denote such a set by A_v (v for *vague*). A_v is defined by a probability distribution on subsets of X. The case where probability $P(A_v = A) = 1$ means that the set A_v is strict and not vague (the concept of fuzzy set is thus more general than the concept classical set). An element a belongs to A_v with a probability $P(a \in A_v)$ which is formally defined by:

$$P(a \in A_v) = \sum_{A \subset X, a \in A} P(A_v = A)$$

Most approaches like ours does only consider the event $a \in A_v$ and make a series of assumptions on the independence of those events. Our own assumptions are linked to the structure of the Bayesian Networks we use to compute the probability that an element belongs to the set of answers to a given query need. We do not discuss this point in this paper.

Fuzzy sets are not sufficient for our problem which is to evaluate if an element belongs to transformations of fuzzy sets. Furthermore, the independence of elementary events (as defined in the previous paragraph) does not hold in our case. We now describe how to define vague sets that are answers to a given XPath-like expression.

We want to define the probability that a doxel a belongs to the set $\varphi(A_v)$ where φ is a set function ($\mathcal{P}(\mathcal{X}) \rightarrow \mathcal{P}(\mathcal{X})$). For a classical set,

$$a \in \varphi(A) \Leftrightarrow \exists b \in A, a \in \varphi(\{b\})$$

which is associated to the logical formula:

$$\bigvee_{b \in \mathcal{X}, a \in \varphi(\{b\})} b \in A$$

The pseudo inverse φ^* can be used in this formula, as $\varphi^*(b) = \{a \in \mathcal{X}/a \in \varphi(b)\}$. The previous formula can be rewritten as:

$$\bigvee_{b \in \varphi^*(\{a\})} b \in A$$

I1 Doxels that are *about* `cat`s

$$\begin{aligned} I1 &= \texttt{//*[about(.,"cat")]} \\ \Psi(I1)(d) &= \Psi_F(\texttt{about(.,"cat")}) \circ \Psi_L(\texttt{*}) \circ \Psi_A^{(0)}(\texttt{/descendant-or-self})(d) \\ &= \Psi'(.)^*(R(\texttt{"cat"})) \cap \mathcal{X} \cap \overline{\text{desc}}(d) \\ &= R(\texttt{cat}) \cap \overline{\text{desc}}(d) \end{aligned}$$

I2 Images in paragraphs about *about* `cat pictures`

$$\begin{aligned} q_1 &= \texttt{"Cat pictures"} \\ I2 &= \texttt{//p[about(.,}q_1\texttt{)]/image} \\ \Psi(I2)(d) &= \Psi_L(\texttt{image}) \circ \Psi_A(\texttt{/child}) \circ \Psi_F(\texttt{about(.,}q_1\texttt{)}) \circ \Psi_L(\texttt{p}) \circ \Psi_A^{(0)}(\texttt{/desc-or-self}) \\ &= (label^{-1}(\texttt{image}) \cap \text{child}) \circ (R(q_1) \cap label^{-1}(\texttt{p}) \cap \overline{\text{desc}}(d)) \\ &= label^{-1}(\texttt{image}) \cap \text{child}(R(q_1) \cap label^{-1}(\texttt{p}) \cap \overline{\text{desc}}(d)) \end{aligned}$$

I2bis Images in paragraphs about *about* `cat pictures`

$$\begin{aligned} q_1 &= \texttt{"Cat pictures"} \\ I2' &= \texttt{//image[../self::p[about(.,}q_1\texttt{)]]} \\ \Psi(I2')(d) &= \Psi_F(\texttt{../self::p[about(.,}q_1\texttt{)]}) \circ \Psi_L(\texttt{image}) \circ \Psi_A^{(0)}(\texttt{desc-or-self}) \\ &= \Psi'(\texttt{../self::p[about(.,}q_1\texttt{)]})^*(\mathcal{X}) \cap (label^{-1}(\texttt{image}) \cap \overline{\text{desc}}(d)) \end{aligned}$$

and

$$\begin{aligned} \Psi_F(\texttt{../p[about(.,}q_1\texttt{)]}) &= (\Psi_F(\texttt{about(.,}q_1\texttt{)}) \circ \Psi_L(\texttt{p}) \circ \Psi_A(\texttt{/child}) \circ \Psi_{AF}^{(0)}(\texttt{parent}))^*(\mathcal{X}) \\ &= (R(q_1) \cap label^{-1}(\texttt{p}) \cap \text{Id} \circ \text{pa})^*(\mathcal{X}) \\ &= \text{child}(R(q_1) \cap label^{-1}(\texttt{p})) \end{aligned}$$

Then,

$$\Psi(I2')(d) = \text{child}(R(q_1) \cap label^{-1}(\texttt{p})) \cap label^{-1}(\texttt{image}) \cap \overline{\text{desc}}(d)$$

We find the same algebraic expression than for I2, despite of a different XPath expression.

I3 Paragraphs *about* `black` in a document written in 2002 which title is *about* `cat` and that contains a picture

$$\begin{aligned} F &= \texttt{@year = 2002 and .//image and about(title,"cat")} \\ I3 &= \texttt{/document[F]//p[about(.,"black")]} \end{aligned}$$

$$\begin{aligned} \Psi(I3)(d) &= \Psi_F(\texttt{about(.,"black")}) \circ \Psi_L(\texttt{p}) \circ \Psi_A(\texttt{/desc-or-self}) \\ &\quad \circ \Psi_F(F) \circ \Psi_L(\texttt{document}) \circ \Psi_A^{(0)}(\texttt{/child}) \\ &= R(\text{"}\texttt{black}\text{"}) \cap label^{-1}(\texttt{p}) \cap \overline{\text{desc}} \circ \Psi_F(F) \circ (label^{-1}(\texttt{document}) \cap d) \\ \Psi'_F(F) &= \text{pa}_{@}(terms^{-1}(2002) \cap label^{-1}(\texttt{year})) \\ &\quad \cap \overline{\text{anc}}(label^{-1}(\texttt{image})) \\ &\quad \cap \text{pa}(R(\text{"}\texttt{cat}\text{"}) \cap label^{-1}(\texttt{title})) \end{aligned}$$

Then,

$$\Psi(I3)(d) = R(\text{"}\texttt{black}\text{"}) \cap label^{-1}(\texttt{p}) \cap \overline{\text{desc}}\left(\Psi'_F(F) \cap label^{-1}(\texttt{document}) \cap d\right)$$

Figure 2: The transformation of our example queries into our algebra

I1 Doxels that are *about* `cats`

$$a \in I1_v \equiv a \in R(\texttt{cat}) \wedge a \in \overline{\text{desc}}(d)$$

I2 Images in paragraphs about *about* `cat pictures`

$$a \in I2_v \equiv a \in label^{-1}(\texttt{image}) \wedge \bigvee_{b \in \text{pa}(a)} \left(b \in R(q_1) \wedge b \in label^{-1}(\texttt{p}) \wedge b \in \overline{\text{desc}}(d)\right)$$

I3 Paragraphs *about* `black` in a document written in 2002 which title is *about* `cat` and that contains a picture

$$\begin{aligned} a \in I3_v \quad \equiv\ & a \in R(\text{"}\texttt{black}\text{"}) \wedge a \in label^{-1}(\texttt{p}) \wedge \bigvee_{b \in \overline{\text{anc}}(a)} \\ & \left(\left(\bigvee_{c \in \text{child}_@(b)} \left(c \in terms^{-1}(\texttt{2002}) \wedge c \in label^{-1}(\texttt{year})\right) \wedge \bigvee_{c \in \overline{\text{desc}}(b)} \left(c \in label^{-1}(\texttt{image})\right) \right. \right. \\ & \left. \left. \wedge \bigvee_{c \in \text{child}(b)} \left(c \in R(\text{"}\texttt{cat}\text{"}) \wedge c \in label^{-1}(\texttt{title})\right) \right) \wedge b \in label^{-1}(\texttt{document}) \wedge b = d \right) \end{aligned}$$

Figure 3: Logical formulas with which we can compute the probability that a doxel is an answer to an information need

and we can then define the recursively the probability that $a \in \varphi(A_v)$:

$$P(a \in \varphi(A_v)) = P(\bigvee_{b \in \varphi^*(\{a\})} b \in A_v)$$

As Fuhr and Grossjohan [2], we define the certain event (denoted $\top$) and the events that are not certain (denoted $\bot$). For a classical set A:

$$x \in A = \begin{cases} \top & \text{if } x \in A \\ \bot & \text{else} \end{cases}$$

We can then define recursively the fact that a doxel belongs to a vague set transformed by φ:

$$x \in \varphi(A_v) \equiv \bigvee_{x' \in \varphi^*(x)} x' \in A_v$$

Finally, for intersection and union operators, we have:

$$\begin{aligned} x \in A_v \cap B_v &\equiv (x \in A_v) \wedge (x \in B_v) \\ x \in A_v \cup B_v &\equiv (x \in A_v) \vee (x \in B_v) \end{aligned}$$

In figure 3, logical formulas associated to the three queries of our initial example are shown. These formulas can be automatically derived from the figure 2. The formula can thus be automatically derived from a query expressed in a XPath-like language (figure 2) that contains an `about` predicate.

5. CONCLUSIONS AND OUTLOOK

In this paper, we presented a new algebra for XML retrieval and described how a query expressed in an XPath-like language can be transformed into an algebra and then into a logical formula. This logical formula can be evaluated by probabilistic models like the Bayesian Networks we use to answer "content only" queries [5]. We intend to use such an algebra for the next INEX campaign.

Our algebra is closely related to existing algebras which address the problem of vague content and structure retrieval like [2]. However, our algebra relies only on simple operators:

- Usual set operators (intersection, union, complement)
- Structural operators (parent, child, etc.)
- Selection operators (doxels with a given label, doxels that answers a query)

In our framework, an XPath-like query is processed in three steps: (1) syntaxic parse of the query, (2) transformation of the query into our algebra and (3) transforming the algebraic expression into an event. The score of an element is then given by its degree of belonging to the set of answers. The only vague set is the "*relevant to a query*" set. But it is easy in our algebra to define new vague sets and even vague operators. This can be useful for vague content and *vague structure* query needs. For example,

- the $label^{-1}(\texttt{paragraph})$ function can be defined as a fuzzy set. In this case, a doxel which has the label `section` can belong to this set with a (possibly low) probability;
- The structural operators (parent, children, etc.) can be defined less strictly, in order to allow some vagueness in the expression of axis.

However, we still need to implement this algebra and to evaluate the results returned to a given query. Another problem is the evaluation of the logical formulas. In simple cases (like in the query language NEXI), we can use our BN without any problem as they encode a number of independence assumptions which make the evaluation of the probability possible. But for complex XPath queries, this is not the case. If complex queries have really an interest in the context of information retrieval in XML documents, this issue should be addressed.

6. REFERENCES

[1] S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. L. Wiener. The lorel query language for semistructured data. *International Journal on Digital Libraries*, 1(1):68–88, 1997.

[2] N. Fuhr and K. Grossjohann. XIRQL: A query language for information retrieval in XML documents. In W. B. Croft, D. J. Harper, D. H. Kraft, and J. Zobel, editors, *The 24th International Conference on Research and Developmenent in Information Retrieval*, New Orleans, Louisiana, USA, Sept. 2001. ACM.

[3] J. List, V. Mihajlovic, A. P. de Vries, and G. Ramírez. The TIJAX XML-IR system at INEX 2003. In N. Fuhr, M. Lalmas, and S. Malik, editors, *INitiative for the Evaluation of XML Retrieval (INEX). Proceedings of the Second INEX Workshop*, Dagstuhl, Germany, Dec. 2003.

[4] G. Navarro and R. Baeza-Yates. Proximal nodes: A model to query document databases by content and structure. *ACM TOIS*, 15(4):401–435, Oct. 1997.

[5] B. Piwowarski and P. Gallinari. A machine learning model for information retrieval with structured documents. In P. Perner, editor, *Machine Learning and Data Mining in Pattern Recognition*, pages 425–438, Leipzig, Germany, July 2003. Springer Verlag.

[6] L. A. Zadeh. Fuzzy Sets. *Information and Control*, 8:338–358, 1965.