Fuzzy XPath Through Fuzzy Logic Programming

Jesús M. ALMENDROS-JIMÉNEZ, Alejandro LUNA and Ginés MORENO

Department of Informatics,
University of Almería,
Almería-04120, Spain
jalmen@ual.es

Department of Computing Systems,
University of Castilla-La Mancha,
Albacete-02071, Spain
alejandro.luna@alu.uclm.es, gines.moreno@uclm.es

Received

Abstract In this paper we present a fuzzy variant of the XPath query language for the flexible information retrieval on XML documents. Our main purpose is to provide a repertoire of operators that offer the possibility of managing satisfaction degrees by adding structural constraints and fuzzy operators inside conditions, in order to produce a ranked sorted list of answers according to user's preferences when composing queries. By using the FLOPER system designed in our research group, our proposal has been implemented with a fuzzy logic language to take profit of the clear synergies between both target and source fuzzy languages.

Keywords XPath Query Language, Fuzzy Logic Programming, Software Tools.

§1 Introduction

User's preferences play a key role in information retrieval. In modern Web based information retrieval systems, user expects to introduce his(er) key words and preferences to influence the search results. However, while the technology is still improving, users get sometimes frustrated with those retrieval

systems which only offer a poor/rigid set of expressive resources. Therefore the need for *flexible query languages* arises, in which the user can formulate queries according to his(er) preferences, being adaptable to data schema but without increasing complexity. In addition, flexible query languages should be equipped with a mechanism for obtaining a certain *ranked list* of answers. The ranking of answers can provide *satisfaction degrees* depending on several factors.

The *XPath* language [11] has been proposed as a standard for XML querying and it is based on the description of the path in the XML tree to be retrieved. XPath allows to specify the name of nodes (i.e., tags) and attributes to be present in the XML tree together with boolean conditions about the content of nodes and attributes.

XPath querying mechanism is based on a boolean logic: the nodes retrieved from an XPath expression are those matching the path of the XML tree. Therefore, the user should know the *XML schema* in order to specify queries. However, even when the XML schema exists, it may not be available for users. Moreover, XML documents with the same XML schema can be very different in structure. Let us suppose the case of XML documents containing the curriculum vitae of a certain group of persons. Although they can share the same schema, each one can decide to include studies, jobs, training, etc. organized in several ways: by year, by relevance, and with different nesting degree. In an XPath-based structural query, the main criteria to provide a certain degree of satisfaction are the *hierarchical deepness* and *document order*. However, user's preferences play also a key role in determining the best solutions. Conditions on XPath expressions are usually of varying importance for a user, that is, the user gives a higher degree of importance to certain requirements when satisfying his(er) wishes. Therefore, the query language should provide mechanisms for assigning *priority to answers*, when they occur in different parts of the document, as well as *priority to queries*, with regard to user's preferences.

In this paper we present a fuzzy variant of the XPath query language for the flexible information retrieval on XML documents. Our main purpose is to provide a repertoire of operators that offer the possibility of managing satisfaction degrees by adding structural constraints and fuzzy operators inside conditions (which must be considered from now on as *fuzzy conditions* instead of *boolean conditions*), in order to produce a ranked sorted list of answers according to user's preferences when composing queries. By using the FLOPER system designed in our research group, our proposal has been implemented with a fuzzy

logic language to take profit of the clear synergies between both target and source fuzzy languages.

Our approach firstly proposes two structural constraints called *DOWN* and *DEEP* for which a certain degree of relevance can be associated. So, whereas *DOWN* provides a ranked set of answers depending on the path they are found from "top to down" in the XML document, *DEEP* provides a ranked set of answers depending on the path they are found from "left to right" in the XML document. Both structural constraints can be used together, assigning degree of importance with respect to the distance to the root XML element.

Secondly, we provide fuzzy variants of *and* and *or* for XPath conditions. Crisp *and* and *or* operators are used in standard XPath over boolean conditions, and enable to impose boolean requirements of the answers. XPath boolean conditions can be referred to attribute values and node content, in the form of equality and range of literal values, among others. Nevertheless, the boolean *and* and *or* operators applied to two sub-conditions would not be precise enough if both sub-conditions would represent more flexible values than true/false (as occurs in the fuzzy case). When several conditions are imposed in a query, each one contributes to satisfy the user's preferences in a different way and perhaps, the user's satisfaction is distinct for each solution.

We have enriched the arsenal of operators of XPath with fuzzy variants of *and* and *or*. Particularly, we have considered three versions of *and*: *and+*, *and*, *and-* (and the same for *or* : *or+*, *or*, *or-*) which make more flexible the composition of fuzzy conditions. Three versions for each operator that come for free from our adaptation of fuzzy logic to the XPath paradigm. One of the most known elements of fuzzy logic is the introduction of fuzzy versions of classical boolean operators. *Product*, *Łukasiewicz* and *Gödel* fuzzy logics are considered as the most prominent logics and give a suitable semantics to fuzzy operators. Our contribution is now to give sense to fuzzy operators into the XPath paradigm, and particularly in user's preferences. We claim that in our work the fuzzy versions provide a mechanism to force (and debilitate) conditions in the sense that stronger (and weaker) user preferences can be modeled with the use of stronger (and weaker) fuzzy conditions. The combination of fuzzy operators in queries permits to specify a ranked set of fuzzy conditions according to user's requirements.

Furthermore, we have equipped XPath with an additional operator that is also traditional in fuzzy logic (apart from *min*, *max*, etc.): the average oper-

ator *avg*. This operator offers the possibility to explicitly give weight to fuzzy conditions. Rating such conditions by *avg*, solutions increase its weight in a proportional way. However, from the point view of the user's preferences, it forces the user to quantify his(er) wishes which, in some occasions, can be difficult to measure. For this reason, fuzzy versions of *and* and *or* are better choices in some circumstances.

Finally, we have equipped our XPath based query language with a mechanism for thresholding user's preferences, in such a way that user can request that requirements are satisfied over a certain percentage.

1.1 Related Work

From the early works about the introduction of fuzzy concepts in databases [15, 52, 53, 16], many works have proposed the handling of imprecise and vague data. For instance, *SQLf* [12] extends SQL by introducing fuzzy conditions evaluated on crisp information. Fuzzy logic plays also a key role in information retrieval [39, 29, 28, 51, 47, 48, 34], and the need for providing fuzzy/flexible mechanisms to XML querying has recently motivated the investigation of extensions of the XPath language.

We can distinguish those in which the main goal is the introduction of fuzzy information in data (similarity, proximity, vagueness, etc) [14, 13, 27, 59, 45, 46] and the proposals in which the main goal is the handling of crisp information by fuzzy concepts [17, 19, 22, 21, 36, 20]. Our work focuses on the second line of research.

The most relevant works are [17, 19] in which authors introduce in XPath flexible matching by means of fuzzy constraints called *close* and *similar* for node content, together with *below* and *near* for path structure. In addition, they have studied the *deep-similar* notion for tree matching, and fuzzy versions for *not*, *and* and *or* operators. In order to provide ranked answers they adopt a fuzzy set based approach in which each answer has an associated numeric value (the membership degree). The numeric value represents the *Retrieval Status Value (RSV)* of the associated item.

Our work is similar to the proposed by [17, 19]. The *below* operator of [17, 19] is equivalent to our proposed *DOWN*: both extract elements that are direct descendants of the current node, and the penalization is proportional to the distance. The *near* operator of [17, 19], which is defined as a generalization of *below*, ranks answers depending on the distance to the required node, in any XPath axis. Our proposed *DEEP* ranks answers depending of the distance to the

current node, but the considered nodes can be direct and non direct descendants. Therefore our proposed *DEEP* combined with *DOWN* is a particular case of *near*. To have the same expressivity power as *near* we could incorporate to our framework a new operator to rank answers from bottom to up. It is not a serious problem, but makes more difficult a direct implementation by fuzzy logic programming. With respect to *similar* and *close* operators proposed in [17, 19], our framework lacks similarity relations and rather focuses on structural (i.e. path-based) flexibility.

On the other hand, some works have studied the *relaxation* of XPath expressions, with the aim to retrieve elements that "almost satisfies" a query. XPath relaxation [21] permits to answer by relaxing the match of the XPath expression with the XML tree. Relaxation provides *approximate query answering*, in which an *score* is provided to answers. The score represents the degree in which the relaxed query coincides with the original one. For instance, in [25] they give a *satisfaction score* in [0,1] to similar queries, which measures the number and type of relaxations applied to the original query. The satisfaction score of the answer is the highest satisfaction score of queries providing the answer. With this aim, an extend XPath language is defined to weight tags and Boolean conditions with scores. Relaxed queries replace tags on the original one by applying renaming, axis relaxation and deletion. In the same line, in [57] they provide vague search on both content and structure based Boolean conditions with dynamic query relaxation. Content conditions are similar to XPath Full-Text specification [18]. XPath relaxation is a mixture of content scores and the number of structural query conditions satisfied. A thesaurus-based similarity method relaxes tags. They count the number of navigational query conditions that are satisfied and combine with the content conditions matched. They assign a small and tunable constant c for every navigational condition that is matched. Query relaxation is also the target of other works [14, 57, 24, 9, 38, 23, 36]. Once scored answers are returned, ranking can be provided, and thus top answers can be considered. The computation of the best k answers with efficient algorithms has been also subject of study. Top-k answering aims to provide as soon as it can, the k top-ranked result elements according to the score. In [25], they study how to provide top-k answers for XPath relaxation including negation. The method used in [57] for top-k similarity queries is based on thresholding. Thresholding and top-k answering for XPath relaxation have been also studied in [22, 7, 8, 38].

Our fuzzy variant of XPath cannot be properly considered an XPath re-

laxation technique, although fuzzy conditions relaxes Boolean conditions: fuzzy *and* and *avg* are weaker that Boolean *and*. Nevertheless, answer ranking can be provided in our work, and thresholding techniques can be applied to efficiently retrieve best solutions. In Section 5 will show the benefits of using a fuzzy command (FILTER) for thresholding, which filters the set of ranked answers in a dynamic way, in order to reduce the runtime and complexity of computations when dealing with large files. Both thresholding and top-k are pruning algorithms which in the case of fuzzy logic programming prune branches of the search tree of solutions. FLOPER was already equipped with thresholding and thus facilitated the implementation of dynamic filtering. In our case, top-k has not been still considered because user preferences are usually measured in terms of degree of satisfaction rather than a number of answers. In any case, we will consider to extend our work with dynamic top-k filtering, by using (fuzzy logic programming) top-k answering in the line of [56]. Additionally, we have studied in [3] how to debug XPath, by introducing the command *DEBUG* to XPath expressions, and whose result is an XPath expression with annotations (*SWAP*,*DELETE* and *JUMP*) for node replacing and deletion, and axis relaxation. Answers of queries annotated with DEBUG are ranked according to degree of similarity of the original query. The reader can found more details in [3].

Finally, during the last decades several fuzzy logic programming systems have been developed where the classical logic programming inference mechanism of SLD–resolution has been replaced with a fuzzy variant able to handle partial truth and to reason with uncertainty. Most of these systems implement the fuzzy resolution principle introduced by Lee in [35], such as languages Prolog-Elf [30], Fril [10], RFuzzy [44], the QLP schemes of [50, 49], as well as many-valued logic programming languages [58, 55]. Our work focuses on the so-called *Multi-Adjoint Logic Programming (MALP)* approach [40, 33], which is an extension of logic programming whose syntax is close to Prolog but enjoys higher levels of flexibility. We have developed the *FLOPER "Fuzzy LOgic Programming Environment for Research"* tool [*1] which manages MALP programs (see [41, 42, 43] for more details). We have found a great opportunity to transfer our technology to the XPath setting.

1.2 Contributions of the Approach

The main contribution of our approach is that we have extended the number of

[*1] `http://dectau.uclm.es/floper`.

fuzzy operators considered so far thanks to the highly expressive power of our framework based on fuzzy logic programming. Our current proposal shows that fuzzy logic programming can be useful in this area. However, the combination of both paradigms involves to re-interpretate fuzzy logic programming concepts in terms of XML and XPath. Multi-adjoint logic programming uses as theoretical basis elements of multi-adjoint lattices as *truth* values associated to logic computations. Truth values are computed from a special type of SLD-resolution. Fuzzy operators are semantically integrated in the framework and serve as underlined mechanism in resolution. Logic rules are labeled with truth values and use fuzzy operators in the rule body as well as implication connectives. Traditional logic programming computations are modified in such a way that resolution steps involve the computation of truth values by the fuzzy operators. The adaptation of the MALP framework to XML and XPath involves the following elements.

- We have accommodated XML documents to fuzzy logic programming. Fortunately, we have been previously working with XML taking logic programming as basis for the XPath implementation [2, 1]. We follow in case of fuzzy logic programming the same kind of encoding of XPath proposed in [2]. There, a predicate called *xpath* is defined by means of logic rules, which basically traverse the representation of the XML tree as a Prolog term. Here, we have modified the predicate *xpath* in order to include the handling of the new fuzzy operators, and the retrieval of ranked lists of answers.
- We have adapted fuzzy operators to XML and XPath. The adaptation can be seen from two points of view: we give denotational semantics to fuzzy operators in XPath queries, and we give operational semantics to fuzzy operators w.r.t. fuzzy logic programming.
- We have integrated multi-adjoint fuzzy logic programming with XML and XPath. Truth values usually associated to logic computations have to be transformed into RSVs (i.e. Retrieval Status Values) of XML nodes. We have defined a new concept of truth value, in the fuzzy logic terminology, to be associated to XPath computations which represents a tree of RSVs, called TV tree. The TV tree describes degrees of satisfaction of each node of the XML tree with respect to an XPath query. Intuitively, for a given XML whose skeleton is a tree, we can define a TV tree with exactly the same topology, such that after "matching" both trees, the content of each node would be labeled with a real number in the unit interval.

- We have accommodated multi-adjoint logic programming theory to XML and XPath. Fuzzy operators and TV trees have been justified in terms of the multi-adjoint schema. TV trees are elements of a multi-adjoint lattice and fuzzy operators are well-defined for such lattice.
- The *fuzzyXPath* predicate of the fuzzy logic based implementation of XPath is defined with MALP rules. MALP rules handle TV trees and computations are mapped to TV trees as truth values in the fuzzy logic programming terminology.
- We have implemented a prototype of our proposal using the FLOPER system. The prototype is publicly available from `http://dectau.uclm.es/fuzzyXPath/`, and it is equipped with a Web interface in which examples can be tested.

Finally, let us remark that the current work agglutinates some elements we have previously studied in [6, 4]. In the preliminary work of [6] (extended in [4]) we focused on the introduction of structural constraints in which we imposed that *DEEP* and *DOWN* operators are just used once in (the root of) XPath expressions. We also proposed the introduction of classical (i.e. product logic) fuzzy logic operators in conditions. Here we will generalize the use of structural constraints which now might occur in any position of the XPath expression. Moreover, we will increase the number of fuzzy operators, incorporating *Łukasiewicz* and *Gödel* fuzzy logic operators, and thresholding constraints. Therefore the current work subsumes but extends our previous work.

1.3 Structure of the Paper

The structure of the paper is as follows. Section 2 will present our proposal of fuzzy XPath, and will show examples of use; Section 3 will provide MALP based foundations to our approach; Section 4 will describe the main elements of the implementation; Section 5 will show the benefits of using the filtering command to reduce the runtime and complexity of computations when dealing with large files; and, finally, Section 6 will conclude and present future work.

§2 A Flexible XPath Language

Our proposal of flexible XPath is defined by the following grammar:

xpath :=	['**[**'deep-down'**]**']path
path :=	literal | text() | node | **@**att | node/path | node//path
node :=	QName | QName[cond]
cond :=	xpath op xpath | xpath num-op number
deep :=	**DEEP**=number
down :=	**DOWN**=number
deep-down :=	deep | down | deep '**;**' down
num-op :=	$> \mid = \mid < \mid <>$
fuzzy-op :=	**and** | **and+** | **and-** | **or** | **or+** | **or-** | **avg** | **avg**{number,number}
op :=	num-op | fuzzy-op

Basically, our proposal extends XPath as follows:

- A given XPath expression can be adorned with «[DEEP = r_1; DOWN = r_2]» which means that the *deepness* of elements is penalized by r_1 and that the *order* of elements is penalized by r_2, and such penalization is proportional to the distance (i.e., the length of the branch and the weight of the tree, respectively). In particular, «[DEEP = 1; DOWN = r_2]» can be used for penalizing only w.r.t. document order. *DEEP* works for //, that is, the deepness in the XML tree is only computed when descendant nodes are explored, while *DOWN* works for both / and //. Let us remark that in our preliminary version [6] *DEEP* and *DOWN* could only be used at the beginning of the XPath expression, however, now they can be used anywhere, many times in an XPath expression.
- We consider three versions for each one of the conjunction and disjunction operators (also called connectives or aggregators) which are based in the so-called *Product*, *Gödel* and *Łukasiewicz* fuzzy logics. The *Gödel* and *Łukasiewicz* logic based fuzzy symbols [1] are represented in our application by *and+*, *and-*, *or-* and *or+*, in contrast with product logic operators *and* and *or* (see Figure 2). Adjectives like *pessimistic*, *realistic* and *optimistic* are sometimes applied to the *Łukasiewicz*, *Product* and *Gödel* fuzzy logics since operators satisfy that, for any pair of real numbers x and y in $[0, 1]$ (as used in MALP):

$$0 \leq \&_{\mathtt{L}}(x, y) \leq \&_{\mathtt{P}}(x, y) \leq \&_{\mathtt{G}}(x, y) \leq 1$$

[1] The fuzzy logic community frequently uses the terms *t-norm* and *t-conorm* for expressing generalized versions of conjunctions and disjunctions.

$\&_{\mathtt{P}}(x,y)$	=	$x * y$	$\|_{\mathtt{P}}(x,y)$	=	$x + y - x * y$	*Product: and/or*
$\&_{\mathtt{G}}(x,y)$	=	$\min(x,y)$	$\|_{\mathtt{G}}(x,y)$	=	$\max(x,y)$	*Gödel: and+/or-*
$\&_{\mathtt{L}}(x,y)$	=	$\max(x+y-1,0)$	$\|_{\mathtt{L}}(x,y)$	=	$\min(x+y,1)$	*Łuka: and-/or+*

Fig. 1 Fuzzy Logical Operators

and the contrary for the disjunction operations (as used in MALP):

$$0 \leq |_{\mathtt{G}}(x,y) \leq |_{\mathtt{P}}(x,y) \leq |_{\mathtt{L}}(x,y) \leq 1$$

So, note that it is more difficult to satisfy a condition based on a pessimistic conjuntor/disjunctor (i.e, *and-/or-* inspired by the *Łukasiewicz* and *Gödel* logics, respectively) than with *Product* logic based operators (i.e, *and/or*), while the optimistic versions of such connectives (i.e., *and+/or+*) are less restrictive, obtaining a greater set of answers. This is a consequence of the following chain of inequalities (as used in MALP):

$$0 \leq \&_{\mathtt{L}}(x,y) \leq \&_{\mathtt{P}}(x,y) \leq \&_{\mathtt{G}}(x,y) \leq |_{\mathtt{G}}(x,y) \leq |_{\mathtt{P}}(x,y) \leq |_{\mathtt{L}}(x,y) \leq 1$$

or equivalently, by using the notation of our application:

$$0 \leq and-(x,y) \leq and(x,y) \leq and+(x,y) \leq or-(x,y) \leq or(x,y) \leq or+(x,y) \leq 1$$

Therefore users should refine queries by choosing operators in the previous sequence from left to right (or from right to left), till finding solutions satisfying in a stronger (or weaker) way the requirements.

- Furthermore, the *avg* operator is defined too in a *weighted* way. Assuming two given *RSV*'s r_1 and r_2, *avg* is defined as $(r_1 + r_2)/2$, and $avg(p_1, p_2)$ is defined as $(p_1 * r_1 + p_2 * r_2)/p_1 + p_2$.
- Finally, we have considered a special case of constraint, a thresholding constraint of the form *xpath op r*, where $r \in [0,1]$ and $op \in \{<, >, =\}$, in which the user can specify the threshold that the RSV of a given fuzzy condition has to raise.

In general, an extended XPath expression defines, w.r.t. an XML document, a sequence of subtrees of the XML document where each subtree has an associated RSV. XPath conditions, which are defined as fuzzy operators applied to XPath expressions, compute a new RSV from the RSVs of the involved XPath expressions, which at the same time, provides an RSV to the node.

2.1 Examples with DEEP and DOWN

```
<hotels>
<hotel name="Melia">
    <close_to>Gran Via
        <close_to>Callao</close_to>
        <close_to>Plaza de Espana</close_to>
    </close_to>
    <services>
        <pool></pool>
        <metro>150</metro>
    </services>
    <price>100</price>
</hotel>
<hotel name="NH">
    <close_to>Sol
        <close_to>Gran Via</close_to>
        <close_to>Callao</close_to>
    </close_to>
    <services>
        <metro>300</metro>
    </services>
    <price>150</price>
</hotel>
<hotel name="Hilton">
    <close_to>Moncloa
        <close_to>Gran Via</close_to>
        <close_to>Sol</close_to>
    </close_to>
    <services>
        <metro>150</metro>
    </services>
    <price>50</price>
</hotel>
<hotel name="Tryp">
    <close_to>Cibeles
        <close_to>Alcala
            <close_to>Gran Via</close_to>
        </close_to>
        <close_to>Retiro</close_to>
    </close_to>
    <services>
        <pool></pool>
        <metro>10</metro>
    </services>
    <price>575</price>
</hotel>
<hotel name="Sheraton">
    <close_to>Recoletos
        <close_to>Cibeles</close_to>
        <close_to>Gran Via
            <close_to>Sol</close_to>
        </close_to>
    </close_to>
    <close_to>Sol</close_to>
    <services>
        <pool></pool>
        <metro>300</metro>
    </services>
    <price>475</price>
</hotel>
</hotels>
```

Fig. 2 Input XML document collecting Hotel's information

In order to illustrate the language, let us see some examples of flexible queries in XPath. We will take as input document the one shown in Figure 2. The example shows a sequence of hotels where each one is described by *name* and *price*, proximity to streets (*close_to*) and provided services (*pool* and *metro* -together with distance-). In the example, we assume that document order has the following semantics. The tag *close_to* specifies the proximity to a given street. However, the order of *close_to* tags is relevant, and the top streets are closer than the streets at the bottom. In other words, the case:

```
hotel_H
        close_to street_A
        close_to street_B
```

implies that hotel H is near to both streets A and B, but closer to A than to B. The nesting of *close_to* has also a relevant meaning. While a given street A can be close to the hotel H, the streets close to A are not necessarily close to the hotel H. In other words, in the case:

```
hotel_H
        close_to street_A
                        close_to street_B
```

the street B is near to street A, and street A is close to hotel H, which implies that street B is also close to hotel H, but no so close as street A. H can be situated at the end of street A, and B can cross A at the beginning. We can say, in this case, that B is an *adjacent* street to H, while A is *close* to H. This means that when looking for a hotel close to a given street, the highest priority should be assigned to streets close to the hotel, while adjacent streets should be relegated to lower priority. The example has been modeled in order to illustrate the use of structural constraints and fuzzy operators. Particularly, when the user tries to find hotels very close to a given street it should be provided a high *DOWN* value and a low *DEEP* value, whereas in the case the user tries to find hotels in the neighborhood of an street should provide high *DEEP* and low *DOWN*.

Example 2.1
In our first example, we focus on the use of *DOWN*. Let us suppose that the user is interested to find a hotel close to Sol street. This might be his(er) first tentative looking for a hotel. Using crisp XPath (s)he would formulate:

$$<< /hotels/hotel[close\_to/text() = \text{“}Sol\text{”}]/@name >>$$

However, it gives the user the set of hotels close to *Sol* without distinguishing the degree of proximity. The fuzzy version of XPath permits to specify a value of degradation of answers, in such a way that the user reformulates the query as:

$$<< /hotels/hotel[[DOWN = 0.9]close\_to/text() = \text{“}Sol\text{”}]/@name >>$$

The query specifies that *close_to* tag is degraded by 0.9 from top to down. In other words, when *Sol* is found close to a hotel, the position in which it occurs gives a different satisfaction value. In this case, we will obtain:

```
<result>
  <result rsv="1.0">NH</result>
  <result rsv="0.9">Sheraton</result>
</result>
```

Fortunately, we have found a hotel (*NH*) which is very close to *Sol*, and one (*Sheraton*) which is a little bit farther from *Sol*.

Let us remark the previous example and the other examples of the Section show the results in order of satisfaction degree.

Example 2.2
Let us suppose now that we are looking for a hotel close to *Callao*. In this case, we can try to make the same question:

$$<< /hotels/hotel[[DOWN = 0.9]close\_to/text() = \text{“}Callao\text{”}]/@name >>$$

However, the result is empty. Therefore we can try to relax the query by changing '/' by '//':

$$<< /hotels/hotel[[DOWN = 0.9]//close\_to/text() = \text{“}Callao\text{”}]/@name >>$$

Now, we will find answers, however, we will not be able to distinguish the proximity of the hotels. Our fuzzy version of XPath permits to specify how the solutions are degraded but not only taking into account the order but also the deepness. In other words, there would be useful to give different weights to be a close street, and to be an adjacent street. Therefore we can use the query:

$$<< /hotels/hotel[[DEEP = 0.5; DOWN = 0.9]//$$
$$close\_to/text() = \text{“}Callao\text{”}]/@name >>$$

obtaining the following results:

```
<result>
  <result rsv="0.5">Melia</result>
  <result rsv="0.45">NH</result>
</result>
```

It seems that *Melia* is near to *Callao*, and *NH* is a little bit farther than *Melia*.

Example 2.3
The use of *DEEP* combined with *DOWN* could be considered as the best choice. However, *DEEP* can be used alone when the user only wants to penalize adjacency. If we like to search hotels near to *Gran Via* street, degrading adjacent streets with a factor of 0.5, we can consider the following query (and we obtain the following result):

$$<< //hotel[[DEEP = 0.5]//close\_to/text() = \text{“}Gran\ Via\text{”}]/@name >>$$

```
<result>
  <result rsv="1.0">Melia</result>
  <result rsv="0.5">NH</result>
  <result rsv="0.5">Hilton</result>
  <result rsv="0.5">Sheraton</result>
  <result rsv="0.25">Tryp</result>
</result>
```

We can see that *Melia* is close to *Gran Via*, while *NH*, *Hilton* and *Sheraton* are situated in adjacent streets of *Gran Via*. *Tryp* is the farthest hotel.

Example 2.4
The following table summarizes the results by combining *DEEP* and *DOWN* in a single query:

$$<< //hotel[[DEEP = r_1; DOWN = r_2]//close\_to/text() = \text{“}Gran\ Via\text{”}]/@name >>$$

HOTEL	(A: $r_1 = 0.1, r_2 = 1$)	(B: $r_1 = 0.5, r_2 = 0.5$)	(C: $r_1 = 1, r_2 = 0.1$)
Melia	1	1	1
NH	0.1	0.5	1
Hilton	0.1	0.5	1
Tryp	0.01	0.25	1
Sheraton	0.1	0.25	0.1

While case C only penalizes closeness, case A penalizes adjacency. Case B penalizes both closeness and adjacency.

2.2 AVG Examples

Example 2.5
Let us suppose that the user is interested in a hotel combining two services like pool and metro. Instead of using classical *and/or* connectives for mixing both features, we can obtain more flexible estimations on *RSV* values by using the *avg* operator as follows:

$$<< //hotel[services/pool\ avg\ services/metro]/@name >>$$

thus obtaining the following results:

```
<result>
  <result rsv="1.0">Melia</result>
  <result rsv="1.0">Tryp</result>
  <result rsv="1.0">Sheraton</result>
  <result rsv="0.5">NH</result>
  <result rsv="0.5">Hilton</result>
</result>
```

By using the *avg* fuzzy operator, the user finds that *Melia*, *Tryp* and *Sheraton* have pool and metro, while *NH* and *Hilton* lack on one of them.

Example 2.6
Now, let us suppose that the importance of the metro is the double of the importance of the pool. In this case, the user can formulate the query as follows:

$$<< //hotel[services/pool\ avg\{1,2\}\ services/metro]/@name >>$$

obtaining the following results:

```
<result>
  <result rsv="1.0">Melia</result>
  <result rsv="1.0">Tryp</result>
  <result rsv="1.0">Sheraton</result>
  <result rsv="0.666667">NH</result>
  <result rsv="0.666667">Hilton</result>
</result>
```

We can see in the results that *NH* and *Hilton* increase the degree of satisfaction w.r.t. the previous query given that they have metro station.

Example 2.7
Let us suppose the user is looking now for hotels giving more importance to the fact that the price of the hotel is lower than 150 euros than to the proximity to *Sol* street. The user can formulate the query as follows, obtaining the results below:

$$<< //hotel[[DEEP = 0.8]//close\_to/text() = \text{“}Sol\text{”}$$
$$avg\{1,2\}\ //price/text() < 150]/@name >>$$

```
<result>
  <result rsv="0.933333">Hilton</result>
  <result rsv="0.666667">Melia</result>
  <result rsv="0.333333">NH</result>
  <result rsv="0.333333">Sheraton</result>
</result>
```

2.3 Thresholding Example

Fuzzy conditions return a satisfaction degree in the infinite space of real numbers between 0 and 1. We can take profit of this feature by imposing *thresholds* to such conditions, thus filtering the set of solutions according to the satisfaction degree. The idea is to formulate queries by directly acting on the satisfaction degrees obtained after evaluating “fuzzy” conditions.

Example 2.8
For instance, let us suppose that in the query of example 2.3, the user looks for hotels in which the degree of proximity to *Gran Via* street is greater than seventy five per cent (i.e., value 0.75 measured between 0 and 1) then (s)he can

formulate the following query, obtaining the following results:

$$<< //hotel[([DEEP = 0.5]//close\_to/text() = \text{“}Gran\ Via\text{”}) > 0.75]/@name >>$$

```
<result>
  <result rsv="1.0">Melia</result>
</result>
```

2.4 Conjunctive/Disjunctive Connective Examples

Example 2.9
In the following queries we express the following requirement: hotels near to *Gran Via*, near to a metro station, having pool, with greater preference (3 to 2) to pool than metro. We will use *and+*, *and* and *and-* which provide distinct levels of exigency, which are demonstrated in the results.

$$<< //hotel[([DEEP = 0.5]//close\_to/text() = \text{“}Gran\ Via\text{”})\ and+$$
$$(//pool\ avg\{3,2\}\ //metro/text() < 200)]/@name >>$$

```
<result>
  <result rsv="1.0">Melia</result>
  <result rsv="0.5">Sheraton</result>
  <result rsv="0.4">Hilton</result>
  <result rsv="0.25">Tryp</result>
</result>
```

$$<< //hotel[([DEEP = 0.5]//close\_to/text() = \text{“}GranVia\text{”})\ and$$
$$(//pool\ avg\{3,2\}\ //metro/text() < 200)]/@name >>$$

```
<result>
  <result rsv="1.0">Melia</result>
  <result rsv="0.3">Sheraton</result>
  <result rsv="0.25">Tryp</result>
  <result rsv="0.2">Hilton</result>
</result>
```

$$<< //hotel[([DEEP = 0.5]//close\_to/text() = \text{“}GranVia\text{”})\ and-$$
$$(//pool\ avg\{3,2\}\ //metro/text() < 200)]/@name >>$$

```
<result>
  <result rsv="1.0">Melia</result>
  <result rsv="0.25">Tryp</result>
  <result rsv="0.1">Sheraton</result>
</result>
```

So, in the first case (the least demanding and optimistic) we obtain four hotels (*Melia*, *Sheraton*, *Hilton* and *Tryp*), as well as in the second case (a little bit more exigent) while third table (the strongest one) lists three candidates (*Melia*, *Tryp* and *Sheraton*). *Sheraton* and *Hilton* are degraded using *and* and *and-*. This effect would even be more evident when previous conditions are compared with a threshold. For instance, to be greater than 0.25. In such a case *and-* gives just a single solution: *Melia*.

§3 Multi-Adjoint Logic Programming and Fuzzy XPath

In this section, we will introduce the elements of multi-adjoint logic programming (MALP). MALP will serve as semantic foundation of our proposal. Moreover, MALP will be used for the implementation of our language. The section will describe the theoretical basis of MALP: multi-adjoint lattices and fuzzy operators defined on them. In addition, it will be described an instance of MALP which considers a multi-adjoint lattice of trees with truth values, and operators defined for such lattice. Finally, we will describe the rule-based implementation of our fuzzy XPath by using MALP rules.

3.1 Multi-Adjoint Logic Programming

In multi-adjoint logic programming [40, 33], we work with a first order language, $\mathcal{L}$, containing variables, function symbols, predicate symbols, constants, quantifiers ($\forall$ and $\exists$), and several arbitrary connectives such as implications ($\leftarrow_1, \leftarrow_2, \ldots, \leftarrow_m$), conjunctions ($\&_1, \&_2, \ldots, \&_k$), disjunctions ($\vee_1, \vee_2, \ldots, \vee_l$), and general hybrid operators ("aggregators" $@_1, @_2, \ldots, @_n$), used for combining/propagating truth values through the rules, and thus increasing the language expressiveness. Additionally, our language $\mathcal{L}$ contains symbols called *truth degrees* belonging to a multi-adjoint lattice L, whose formal description will be presented afterward in Definition 3.4 (see also Figure 3.1).

A *rule* is a formula "$A \leftarrow_i \mathcal{B}$ with α", where A is an atomic formula (usu-

ally called the *head*), $\mathcal{B}$ (which is called the *body*) is a formula built from atomic formulas $B_1, \ldots, B_n$ ($n \geq 0$), truth values of L and conjunctions, disjunctions and general aggregations, and finally $\alpha \in L$ is the "weight" or *truth degree* of the rule. The set of truth values L may be the carrier of any complete lattice, as for instance occurs with the interval $([0,1], \leq)$, where $\leq$ is the usual order. Consider, for instance, the program $\mathcal{P}$ of Figure 3 composed of three rules with associated multi-adjoint lattice $\langle [0,1], \leq, \leftarrow_{\mathtt{P}}, \&_{\mathtt{P}} \rangle$, where label P stands for *Product logic* with the following connective definitions (for implication, conjunction and disjunction symbols, respectively): "$\leftarrow_{\mathtt{P}}(x,y) = \min(1, x/y)$", "$\&_{\mathtt{P}}(x,y) = x * y$" and "$|_{\mathtt{P}}(x,y) = x + y - x * y$".

$\mathcal{R}_1$:	$p(X)$	$\leftarrow_{\mathtt{P}}$	$q(X,Y) \mid_{\mathtt{P}} r(Y)$	*with*	0.8
$\mathcal{R}_2$:	$q(a,Y)$	$\leftarrow$		*with*	0.9
$\mathcal{R}_3$:	$r(b)$	$\leftarrow$		*with*	0.7

Fig. 3 Example of MALP program

In order to describe the procedural semantics of the multi–adjoint logic language, in the following we denote by $\mathcal{C}[A]$ a formula where A is a subexpression (usually an atom) which occurs in the –possibly empty– one hole context $\mathcal{C}[]$ whereas $\mathcal{C}[A/A']$ means the replacement of A by A' in context $\mathcal{C}[]$, and $mgu(E)$ is the *most general unifier* of an equation set E. The pair $\langle \mathcal{Q}; \sigma \rangle$ composed of a goal and a substitution is called a *state*. So, given a program $\mathcal{P}$, an *admissible computation* is formalized as a state transition system, whose transition relation $\overset{\text{AS}}{\rightsquigarrow}$ is the smallest relation satisfying the following *admissible rules*:

1) $\langle \mathcal{Q}[A]; \sigma \rangle \overset{\text{AS}}{\rightsquigarrow} \langle (\mathcal{Q}[A/v \&_i \mathcal{B}])\theta; \sigma\theta \rangle$ if A is the selected atom in goal $\mathcal{Q}$, $A' \leftarrow_i \mathcal{B}$ *with* v in $\mathcal{P}$, where $\mathcal{B}$ is not empty, and $\theta = mgu(\{A' = A\})$

2) $\langle \mathcal{Q}[A]; \sigma \rangle \overset{\text{AS}}{\rightsquigarrow} \langle (\mathcal{Q}[A/v])\theta; \sigma\theta \rangle$ if $A' \leftarrow$ *with* v in $\mathcal{P}$, $\theta = mgu(\{A' = A\})$

The following derivation illustrates our definition (note that the exact program rule used -after being renamed, that is, *standardized apart*- in the corresponding step is annotated as a super–index symbol, whereas exploited atoms appear underlined):

$$
\begin{array}{llllll}
\&_{\mathtt{P}}(x,y) & = & x * y & \leftarrow_{\mathtt{P}}(x,y) & = & \min(1, x/y) \quad \textit{Product} \\
\&_{\mathtt{G}}(x,y) & = & \min(x,y) & \leftarrow_{\mathtt{G}}(x,y) & = & \begin{cases} 1 & \text{if } y \leq x \\ x & \text{otherwise} \end{cases} \quad \textit{Gödel} \\
\&_{\mathtt{L}}(x,y) & = & \max(x+y-1,0) & \leftarrow_{\mathtt{L}}(x,y) & = & \min\{x-y+1,1\} \quad \textit{Łuka.}
\end{array}
$$

Fig. 4 Adjoint Pairs in $([0,1], \leq)$

$$
\begin{array}{ll}
\langle \underline{p(X)}; \{\} \rangle & \overset{\text{AS}}{\rightsquigarrow} \ \mathcal{R}_1 \\
\langle 0.8 \ \&_{\mathtt{P}} \ (\underline{q(X_1, Y_1)} \ |_{\mathtt{P}} \ r(Y_1)); \{X/X_1\} \rangle & \overset{\text{AS}}{\rightsquigarrow} \ \mathcal{R}_2 \\
\langle 0.8 \ \&_{\mathtt{P}} \ (0.9 \ |_{\mathtt{P}} \ \underline{r(Y_2)}); \{X/a, X_1/a, Y_1/Y_2\} \rangle & \overset{\text{AS}}{\rightsquigarrow} \ \mathcal{R}_3 \\
\langle 0.8 \ \&_{\mathtt{P}} \ (0.9 \ |_{\mathtt{P}} \ 0.7); \{X/a, X_1/a, Y_1/b, Y_2/b\} \rangle &
\end{array}
$$

The final formula can be directly interpreted in the lattice L to obtain the final *fuzzy computed answer*. So, since $0.8 \ \&_{\mathtt{P}} \ (0.9 \ |_{\mathtt{P}} \ 0.7) = 0.8 * (0.9 + 0.7 - (0.9 * 0.7)) = 0.776$, we say that the truth degree of $p(X)$ is 0.776 when X is a.

3.2 MALP and Fuzzy XPath

MALP can be used as basis of our proposed flexible extension of XPath as follows. The idea is to consider MALP as semantic background for fuzzy queries expressed in XPath. With this aim we have to accommodate MALP truth values and connectives to XML documents, RSVs, fuzzy operators and structural/thresholding constraints.

Firstly, we have to consider a suitable multi-adjoint lattice. The elements of the multi-adjoint lattice represent the truth values. In the context of XPath, truth values are trees of truth values, which we call *TV trees*. TV trees represent the RSV associated to each node of the ordered XML tree. From a theoretical point of view, the evaluation of a given goal w.r.t. a MALP program, will return a TV tree (i.e, an ordered tree of real numbers in the interval $[0,1]$) as the result of the computation.

Definition 3.1

(TV trees) Formally, a *TV tree* is an empty tree or a pair $(r, \overline{u}^n)$ where $r \in [0,1]$ and $\overline{u}^n$ is a sequence of n TV trees. In a TV tree t we denote by $root(t)$ to r and by $ch(t)$ to $\overline{u}^n$. Let $\mathcal{T}$ be the set of TV trees, and $\leq$ the usual order of real numbers; we can define the following order between TV trees; $t \preceq_{\mathcal{T}} s$ iff $root(t) \leq root(s)$ and $ch(t) \preceq_{\mathcal{T}} ch(s)$ whenever t is not empty; and $t \preceq_{\mathcal{T}} s$

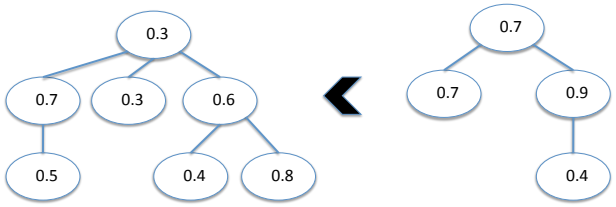

Fig. 5 TV trees partial order

whenever s is empty.

Abusing from the notation, in the previous $\preceq_T$ is used for sequences of TV trees, whose definition is as follows. Given two sequences of TV trees $\overline{u}^n = (u_1, \ldots, u_n)$ and $\overline{v}^m = (v_1, \ldots, v_m)$ then $\overline{u}^n \preceq_T \overline{v}^m$ whenever there exists a subsequence $\overline{s}^m$ of $\overline{u}^n$ such that $s_i \preceq_T v_i$ for all $1 \leq i \leq m$.

Intuitively, TV trees are ordered in decreasing order w.r.t. size, and increasing w.r.t. the relation on real numbers, and therefore the biggest element is the empty tree. We can see in Figure 5 an example of trees ordered by the $\preceq_T$ relation, where the sequence of children is $((0.7, (0.5)), (0.6, (0.4, 0.8)))$; an alternative sequence is $(0.3, (0.6, (0.4, 0.8)))$.

We denote by $first(\overline{u}^n, k)$ (resp. $last(\overline{u}^n, k)$) the first $n - k$ (resp. last k) elements of the sequence $\overline{u}^n$. When $k > n$ then $first(\overline{u}^n, k)$ is completed with k-n empty trees at the right hand side and $last(\overline{u}^n, k)$ is completed with k-n empty trees at the left hand side.

Proposition 3.1
$\preceq_T$ is a partial order (reflexive, antisymmetric and transitive).

Proof Reflexivity and transitivity can be easily proved. The relation is antisymmetric reasoning by induction as follows. Let us suppose $t \preceq_T s$ and $s \preceq_T t$ then, when t and s are empty then trivially $s = t$; otherwise $root(t) \leq root(s)$ and $root(s) \leq root(t)$ therefore $root(t) = root(s)$; in addition (a) there exists a subsequence $\overline{s}^m$ of $\overline{u}^n$ such that $s_j \preceq_T w_i$, $1 \leq i \leq m$, where $ch(t) = \overline{u}^n$ and $ch(s) = \overline{w}^m$; analogously, (b) there exists a subsequence $\overline{l}^n$ of $\overline{w}^m$ of such that $l_k \preceq_T u_k$, $1 \leq k \leq n$. We can reason now that $n = m$: from (a) we have that $n \geq m$, and from (b) we have that $m \geq n$. Now, from (a) $\overline{u}^n \preceq_T \overline{w}^n$ and from (b) $\overline{w}^n \preceq_T \overline{u}^n$, thus by hypothesis $\overline{u}^n = \overline{w}^n$, and therefore $s = t$. ■

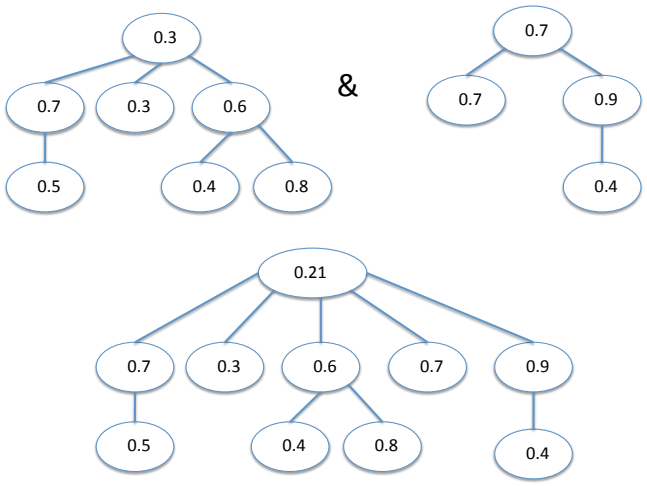

Fig. 6 Conjunction on TV trees (according Product logic)

Now, we can define the following operations in the set $\mathcal{T}$.

Definition 3.2
(Conjunction on TV trees) The operation $\&_O^{\mathcal{T}}(t,s)$, $O \in \{\mathtt{P},\mathtt{G},\mathtt{L}\}$ is defined as the tree whose root is $\&_O(root(t),root(s))$, and children $ch(t)+ch(s)$; whenever t (resp. s) is not empty, and otherwise the result is s (resp. t), where $\&_O$ is the corresponding operator over $[0,1]$ and $+$ is the concatenation of sequences.

Basically, the conjunction is defined as the application of the corresponding conjunction operator to the roots of the trees, and the concatenation of the children. We can see in Figure 6 an example of conjunction between TV trees.

Definition 3.3
(Implication Operator on TV trees) The operation $\leftarrow_O^{\mathcal{T}}(t,s)$, $O \in \{\mathtt{P},\mathtt{G},\mathtt{L}\}$ is defined in the case of non-empty trees t and s as the tree whose root is $\leftarrow_O (root(t),root(s))$, and the children are $first(\overline{u}^n, n-m)+\bar{t}$ if $n > m$ and $\bar{t}$ otherwise, where $\bar{t} = last(\overline{u}^n, m)\backslash\overline{w}^m$ where $ch(t)=\overline{u}^n$ and $ch(s)=\overline{w}^m$. In the case t is empty is defined as the empty tree and in the case s is empty as t.

In the previous definition, we use the difference of two sequences of TV trees: $\overline{v}^n\backslash\overline{w}^n$ defined as the sequence $\overline{u}^n$ in which $\overline{u}^n$ is empty whenever $w_i \preceq_{\mathcal{T}} v_i$ for all $1 \leq i \leq n$; and $\overline{u}^n = \overline{v}^n$, otherwise. Basically, the implication

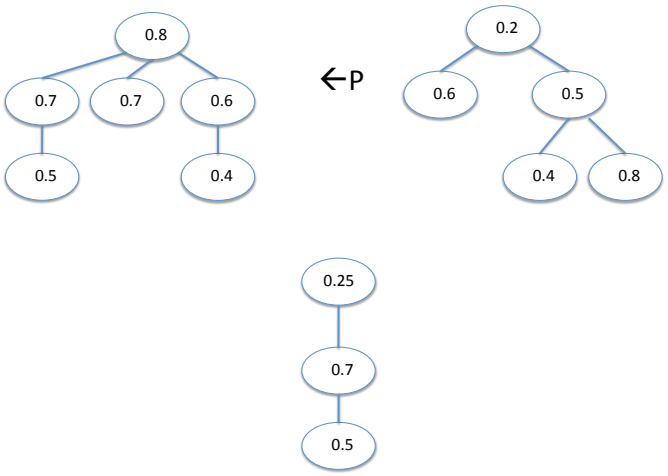

Fig. 7 Implication on TV trees (according Product logic)

operator $\leftarrow_O^{\mathcal{T}}(t,s)$ computes the corresponding adjoint operator to the roots of the trees, and the children of t are replaced by empty TV trees (from right to left) whenever they are greater or equal than the corresponding element of s. We can see in Figure 7 an example of use of the implication, in the case of product logic, between TV trees.

Our goal now is to prove that TV trees conform a multi-adjoint lattice defined as follows.

Definition 3.4
Let $(L, \preceq)$ be a lattice. A multi-adjoint lattice is a tuple:

$$\langle L, \preceq, \leftarrow_1, \&_1, \ldots, \leftarrow_n, \&_n \rangle$$

such that:

1. $(L, \preceq)$ is complete, namely, $\forall S \subset L$ non empty $\exists inf(S), sup(S)$. Then, it is a bounded lattice, i.e. it has bottom and top elements, denoted by $\top$ and $\bot$, respectively.
2. $(\&_i, \leftarrow_i)$ is an adjoint pair in $(L, \preceq)$, i.e.:
   a. $\&_i$ is non decreasing in both arguments, for all i, $i = 1, \ldots, n$.
   b. $\leftarrow_i$ is non decreasing in the first argument and non increasing in the second, for all i.
   c. $x \preceq (y \leftarrow_i z)$ if and only if $(x \&_i z) \preceq y$, for any $x, y, z \in L$ (adjoint

property).

3. $\top \&_i v = v \&_i \top = v$ for all $v \in L$, $i = 1, \ldots, n$, where $\top = sup(L)$.

TV trees are a multi-adjoint lattice based on truth values over $[0, 1]$. In order to prove this property, we need the following auxiliary proposition.

Proposition 3.2
$\langle [0,1], \leq, \leftarrow_{\mathtt{P}}, \&_{\mathtt{P}}, \leftarrow_{\mathtt{G}}, \&_{\mathtt{G}}, \leftarrow_{\mathtt{L}}, \&_{\mathtt{L}} \rangle$ is a multi-adjoint lattice.

Note that even when the previous claim holds too for many other arbitrary multi-adjoint lattices $(L, \preceq)$, we prefer to accommodate it to the $[0, 1]$ case because our notion of TV trees simply relies on this concrete kind of basic truth degrees (i.e., real numbers in the unit interval). Now, we are ready to establish the following theorem proving our intended result.

Theorem 3.1
$\langle \mathcal{T}, \preceq_{\mathcal{T}}, \leftarrow_{\mathtt{P}}^{\mathcal{T}}, \&_{\mathtt{P}}^{\mathcal{T}}, \leftarrow_{\mathtt{G}}^{\mathcal{T}}, \&_{\mathtt{G}}^{\mathcal{T}}, \leftarrow_{\mathtt{L}}^{\mathcal{T}}, \&_{\mathtt{L}}^{\mathcal{T}} \rangle$ is a multi-adjoint lattice.

Proof

1. Let be a non empty $S \subset \mathcal{T}$, then $inf(S)$ is defined as the empty TV tree whenever all the elements of S are empty, otherwise as the TV tree whose root is the infimum of the roots of the non-empty elements of S, and the children are the sequence $\overline{u}^n$ where $u_i = inf(\{w_i | \overline{w}^n \in perm(s)_n, s \in S\})$, where $n = maxChildren(S)$ represents the maximum number of children of the elements of S, and $perm(s)_n$ is the set of all sequences of length n including $ch(s)$ as subsequence and empty trees in the other positions. Now, $inf(S) \preceq_{\mathcal{T}} s$, for all $s \in S$ by construction: when $s \in S$ then $root(inf(S)) \leq root(s)$ and $inf(\{w_i | perm(s)_n = \overline{w}^n, s \in S\}) \preceq_{\mathcal{T}} ch(s)$ given that $ch(s)$ is a subsequence of a sequence of $perm(s)$. Now, $inf(S)$ is the greatest lower bound: let us suppose $t \preceq_{\mathcal{T}} s$ for all $s \in S$; now, $root(t) \leq root(s)$ for all $s \in S$; thus $root(inf(S)) \leq root(t)$; in addition $ch(t) \preceq_{\mathcal{T}} ch(s)$ for all $s \in S$, and thus, for all $p \in perm(t)_n$ and $q \in perm(s)_n$ we have that $p \preceq_{\mathcal{T}} q$; and then $inf(\{w_i | \overline{w}^n \in perm(s)_n, s \in S\}) \preceq_{\mathcal{T}} p \preceq_{\mathcal{T}} ch(t)$, concluding that $inf(S) \preceq_{\mathcal{T}} t$. Analogously, $sup(S)$ is defined as the empty TV tree whenever all the elements of S are empty, otherwise as the TV tree whose root is the supremum of the roots of the non-empty elements of S, and the children are the sequence $\overline{u}^n$ where

$u_i = sup(\{w_i | \overline{w}^n \in perm(s)_n, s \in S\})$, where $n = maxChildren(S)$. $\top$ of $\mathcal{T}$ is the empty tree and $\bot$ is the (infinite) tree with 0 in all nodes.

2. a. $\&_O^{\mathcal{T}}$, $O \in \{\mathtt{P}, \mathtt{G}, \mathtt{L}\}$ is non decreasing: by Proposition 3.2 whenever $v \preceq_{\mathcal{T}} s$, $v' \preceq_{\mathcal{T}} s'$ then $root(v)\&_O root(v') \leq root(s)\&_O root(s')$; in addition, $c_i \preceq_{\mathcal{T}} b_i$, $c'_k \preceq_{\mathcal{T}} b'_k$, for subsequences $\overline{c}^n$ of $\overline{a}^p$ and $\overline{c'}^m$ of $\overline{a'}^q$, where $ch(v) = \overline{a}^p$, $ch(s) = \overline{b}^n$, $ch(v') = \overline{a'}^q$ and $ch(s') = \overline{b'}^m$. Now, the children of $\&_O^{\mathcal{T}}(v, v')$ is $\overline{a}^p + \overline{a'}^q$, and the children of $\&_O^{\mathcal{T}}(s, s')$ is $\overline{b}^n + \overline{b'}^m$, thus $\&_O^{\mathcal{T}}(v, v') \preceq_{\mathcal{T}} \&_O^{\mathcal{T}}(s, s')$ taking the subsequence $\overline{c}^n + \overline{c'}^m$ of $\overline{a}^p + \overline{a'}^q$.

b. $\leftarrow_O^{\mathcal{T}}$, $O \in \{\mathtt{P}, \mathtt{G}, \mathtt{L}\}$ is non decreasing in the first argument and non increasing in the second.

Let us see the first case: non decreasing w.r.t. the first argument; let $v \preceq_{\mathcal{T}} s$ be non empty trees; then by Proposition 3.2: $root(v) \leq root(s)$ and therefore $\leftarrow_O (root(s), root(t)) \leq \leftarrow_O (root(v), root(t))$; in addition, there exists a subsequence $\overline{a'}^m$ of $\overline{a}^n$ such that $a'_i \preceq_{\mathcal{T}} b_i$, $1 \leq i \leq m$, where $ch(v) = \overline{a}^n$ and $ch(s) = \overline{b}^m$. Now, let us suppose $ch(t) = \overline{c}^k$, and $c_j \not\preceq_{\mathcal{T}} b_j$ of $last(\overline{b}^m, k)$. $c_j \not\preceq_{\mathcal{T}} a'_j$ is satisfied from $a'_j \preceq_{\mathcal{T}} b_j$, otherwise by transitivity we have a contradiction. Therefore $\leftarrow_O^{\mathcal{T}}(v, t)$ contains as children a'_j, and this happens for each element b_j of $last(\overline{b}^m, k)$. Therefore, we can prove $\leftarrow_O^{\mathcal{T}}(v, t) \preceq_{\mathcal{T}} \leftarrow_O^{\mathcal{T}}(s, t)$ taking the subsequence $\overline{a'}^m$. Similarly the case $k \geq m$ taking $\overline{a'}^m$ since empty trees are added to the left hand side. When either v or s are empty then the property holds trivially.

Let us see the second case: non decreasing in the second argument; let $s \preceq_{\mathcal{T}} v$ be then $root(v) \leq root(s)$ and therefore by Proposition 3.2 $\leftarrow_O (root(t), root(s)) \leq \leftarrow_O (root(t), root(v))$; in addition, there exists a subsequence $\overline{a'}^m$ of $\overline{a}^n$ such that $a'_i \preceq_{\mathcal{T}} b_i$, $1 \leq i \leq m$, where $ch(v) = \overline{a}^n$ and $ch(s) = \overline{b}^m$. Now, let us suppose $ch(t) = \overline{c}^k$, and $a'_j \not\preceq_{\mathcal{T}} c_j$, c_j of $last(\overline{c}^k, n)$. Thus, $b_j \not\preceq_{\mathcal{T}} c_j$ is satisfied from $a'_j \preceq_{\mathcal{T}} b_j$, otherwise by transitivity we have a contradiction. Therefore $\leftarrow_O^{\mathcal{T}}(t, s)$ contains as children c_j, when $\leftarrow_O^{\mathcal{T}}(t, v)$ does. Thus, $\leftarrow_O^{\mathcal{T}}(t, s) \preceq_{\mathcal{T}} \leftarrow_O^{\mathcal{T}}(t, v)$ taking the subsequence $\overline{c}^k$. Similarly the case $n \geq k$ taking $\overline{c}^k$ since empty trees are added to the left hand side. When either s or v are empty the property holds trivially.

c. $x \preceq_{\mathcal{T}} (y \leftarrow_O^{\mathcal{T}} z)$ if and only if $(x \&_O^{\mathcal{T}} z) \preceq_{\mathcal{T}} y$:

($\Rightarrow$):

By Proposition 3.2 we have that if $root(x) \leq \leftarrow_O(root(y), root(z))$ then $(root(x)\ \&_O root(z)) \leq root(y)$; in addition, there exists a subsequence $\overline{a'}^m$ of $\overline{a}^n$ such that $a'_i \preceq_\mathcal{T} b_i$ $1 \leq i \leq m$ where $ch(x) = \overline{a}^n$ and $ch(y \leftarrow^\mathcal{T}_O z) = \overline{b}^m$. Now, let $ch(y) = \overline{c}^k$ and $ch(z) = \overline{d}^p$ be, by hypothesis we have that there exists $\overline{a''}^{k-p}$ subsequence of $\overline{a'}^m$ such that $a''_j \preceq_\mathcal{T} c_j$, for each $c_j \in first(\overline{c}^k, k-p)$, $1 \leq j \leq k-p$. In addition, when $c'_j \in last(\overline{c}^k, p)$, $1 \leq j \leq p$ then either $last(\overline{c}^k, p) \backslash \overline{d}^p = last(\overline{c}^k, p)$ and there exists $\overline{a'''}^p$ subsequence of $\overline{a'}^m$ such that either $a'''_j \preceq_\mathcal{T} c'_j$ for all $1 \leq j \leq p$ or $last(\overline{c}^k, p) \backslash \overline{d}^p$ is empty and $d_j \preceq_\mathcal{T} c'_j$ for all $1 \leq j \leq p$. Now, we can prove $(x \&^\mathcal{T}_O z) \preceq_\mathcal{T} y$ by taking the subsequence $\overline{a''}^{k-p} + \overline{a'''}^p$ in the first case, and $\overline{a''}^{k-p} + \overline{d}^p$ in the second one. Similarly the case $p \geq k$ taking $\overline{a'''}^p$ and $\overline{d}^p$ in each one of the cases. When either x or y are empty the result is trivial.

($\Leftarrow$):

By Proposition 3.2 we have that if $(root(x) \&_O root(z)) \leq root(y)$ then $root(x) \leq \leftarrow_O(root(y),\ root(z))$; in addition, there exists a subsequence $\overline{a'}^m$ of $\overline{a}^n$ such that $a'_i \preceq_\mathcal{T} b_i$ $1 \leq i \leq m$ where $ch(x \&_O z) = \overline{a}^n$ and $ch(y) = \overline{b}^m$. Now, we have two cases: either $last(\overline{b}^m, k) \backslash \overline{c}^k$ is empty where $ch(z) = \overline{c}^k$ and therefore for all $b'_j \in last(\overline{b}^m, k)$ $c_j \preceq_\mathcal{T} b'_j$, $1 \leq j \leq k$ or $last(\overline{b}^m, k) \backslash \overline{c}^k = last(\overline{b}^m, k)$ and therefore there exists a subsequence $\overline{a''}^k$ of $\overline{a'}^m$ such that $a''_j \preceq_\mathcal{T} b'_j$, $1 \leq j \leq k$. Now, we can prove that $x \preceq_\mathcal{T} (y \leftarrow_O z)$ taking the subsequence $first(\overline{a'}^m, m-k)$ in the first case, and $first(\overline{a'}^m, m-k) + \overline{a''}^k$ in the second case. Similarly the case $k \geq m$ taking the empty set in the first case, and $\overline{a''}^k$ in the second one. since empty trees are added to the left hand side.

3. $\top \&^\mathcal{T}_O v = v \&^\mathcal{T}_O \top = v$ given that $\top$ is the empty tree.

■

Fuzzy XPath will be grounded in multi-adjoint logic programming with TV trees as truth values, *Product*, *Łukasiewicz* and *Gödel* operators, and two extra monotonic hybrid operators, particularly, *@avg* and *@fuse*.

Now, we would like to show how TV trees are used for representing computations of results in our fuzzy variant of XPath, and how MALP rules are used for

```
[element(hotels, [],
 [element(hotel, [name='Melia'],
    [element(close_to, [],
        [
                'Gran Via',
                element(close_to,[],['Callao']),
                element(close_to,[],['Plaza de Espana'])
                ],
     element(services,[],
          [
                element(pool,[],[]),
                element(metro,[],[150])
                ],
     element(price,[],[100])
    ]
  element(hotel,[name="NH"],
....
]
```

Fig. 8 Example of XML data represented in MALP

computing results from XPath expressions.

1. XML documents are represented with MALP terms, which are identical to Prolog terms. MALP represents XML trees with a term of the form:

   $$\texttt{element}(\texttt{tag}, \texttt{attributes}, \texttt{children})$$

   where *tag* is the root of the tree, *attributes* is a list of attribute/value pairs, and *children* is a list of children. For instance, the example of Figure 2 is represented with the MALP term of Figure 8.
2. XPath expressions are also represented as MALP terms, particularly, with a MALP list. For example, */hotel/services/pool* is represented by `[hotel,services,pool]`, where some elements of the list can be also either a list (for nested XPath expressions), an attribute name (like *attr(name)*), a label *relativePath* for representing // or a fuzzy condition (represented by a MALP term of the form `tree(op,xpath,xpath)`).
3. TV trees are represented as MALP terms of the form:

   $$\texttt{tv}(\texttt{truthvalue}, [\texttt{nodecontent}, \texttt{siblingtv}, \texttt{childrentv}])$$

   where *truthvalue* is the truth value of the current node, *nodecontent* is the content of the node when it is included in the answer of the query, *siblingtv* is the TV tree of the first sibling node, and *childrentv* is the TV tree of the first children. We have to remark the following consideration: TV trees are trees of truth values, according to the definition of previous section, however TV trees in MALP include the content of the selected

nodes in order to make easier the implementation. Moreover, TV trees in MALP are represented as binary trees.

Figure 9 shows an example of TV tree computed by the *fuzzyXPath* predicate for the query:

$$<< [DEEP = 0.5; DOWN = 0.9]//hotel[//$$
$$close\_to/text() = \text{“}Gran\ Via\text{”}]/@name >>$$

```
tv(1.0, [[],
    tv(0.5, [[tv(1.0,[]),['Melia']], [],
    tv(0.9, [[tv(1.0,[]),['NH']], [],
    tv(0.9, [[tv(1.0,[]),['Hilton']], [],
    tv(0.9, [[tv(1.0,[]),['Tryp']], [],
    tv(0.9, [[tv(1.0,[]),['Sheraton']],[],[]]) ]) ]) ])]), []])
```

Fig. 9 Example of a *TV* structure in MALP

4. Now, a MALP predicate called *fuzzyXPath* that takes as input (a) an XPath expression represented by a list, (b) an XML tree represented by a MALP term, and (c) *DEEP* and *DOWN* values as TV trees (by default they are *tv(1,[])*. It returns the TV tree associated to the query. For instance, a call to *fuzzyXPath* for solving the previous query and thus returning the TV tree of Figure 9, could have the following shape (where *«XMLdata»* refers to the MALP term of Figure 8 representing the XML document in Figure 2):

$$\texttt{fuzzyXPath}([\texttt{relativePath}, \texttt{hotel},$$
$$\texttt{tree}(\texttt{op}(=, [\texttt{relativePath}, \texttt{close\_to}, \texttt{text}], \text{“}\texttt{Gran Via}\text{”}), \texttt{nil}, \texttt{nil}),$$
$$\texttt{attr}(\texttt{name})], << \texttt{XMLdata} >>, \texttt{tv}(0.5, []), \texttt{tv}(0.9, []))$$

5. The definition of the *fuzzyXPath* predicate distinguishes cases with regard to the XPath expression to be evaluated. Such MALP predicate basically traverses the XML tree represented by the MALP term, and recursively computes for each node the associated RSV.

   For instance, Figure 10 shows the case in which the current root matches with the current path. We can see that $\&_{\texttt{p}}^{\mathcal{T}}$ is used as fuzzy operator, to compute *DEEP* and *DOWN* values for each recursive call. In addition,

$\leftarrow_{\mathrm{P}}^{\mathcal{T}}$ is used as implication connective, and $@fuse$ is used for re-building the answer. The weight of the MALP rules is always *tv(1,[])*.

```
fuzzyXPath([Label|LabelRest],[element(Label,_,Children)|Siblings],Deep,Down)
            ←ₚ^T
            @fuse (
              tv(1,[element(Label,Attr,Children), [], []]),
              &ₚ^T(Deep,fuzzyXPath(LabelRest,Children,Deep,Down)),
              &ₚ^T(Down,fuzzyXPath([Label|LabelRest],Siblings,Deep,Down))
            ) with tv(1,[])
```

Fig. 10 Example of MALP rule

6. Figure 11 shows the case of MALP rules called from *fuzzyXPath* for evaluating *avg*, where fuzzy conditions are handled by a predicate called *execute_fcond*.

```
fuzzyXPath([Label,tree(A,B,C)],[element(Label,Attr,Children)|Siblings],Deep,Down)
            ←ₚ^T
            @fuse(
             execute_fcond(Label,tree(A,B,C),element(Label,Attr,Children)),
             &ₚ^T(Down,fuzzyXPath([Label,tree(A,B,C)],Siblings,Deep,Down))
            ) with tv(1,[])

execute_fcond(Label,tree(avg,T1,T2),element(Label,Attr,Children)) ←ₚ^T
            @avg(
             execute_fcond(Label,T1,element(Label,Attr,Children)),
             execute_fcond(Label,T2,element(Label,Attr,Children))
            ) with tv(1,[])
```

Fig. 11 Examples of MALP rules for evaluating conditions

For instance, given the query:

$$<< //hotel[services/pool\ avg\ services/metro]/@name >>$$

the corresponding call to the *fuzzyXPath* predicate would be:

```
fuzzyXPath([relativePath, hotel, tree(avg,
            tree(exist([services, pool]), nil, nil),
```

$$\texttt{tree}(\texttt{exist}([\texttt{services}, \texttt{metro}]), \texttt{nil}, \texttt{nil})),$$

$$\texttt{attr}(\texttt{name})], << \texttt{XMLdata} >>, \texttt{tv}(1.0, []), \texttt{tv}(1.0, []))$$

which eventually will invoke predicate *execute_fcond* (for evaluating the fuzzy condition containing the *avg* operator) as follows:

$$\texttt{execute\_fcond}(\texttt{hotel}, \texttt{tree}(\texttt{avg},$$

$$\texttt{tree}(\texttt{exist}([\texttt{services}, \texttt{pool}]), \texttt{nil}, \texttt{nil}),$$

$$\texttt{tree}(\texttt{exist}([\texttt{services}, \texttt{metro}]), \texttt{nil}, \texttt{nil})),$$

$$<< \texttt{XMLdata} >>, \texttt{TV\_Cond})$$

Figure 12 shows the resulting TV tree, which clearly resembles the XML file previously reported in Example 2.5.

```
tv(1.0,[[],
    tv(1.0,[[tv(1.0, []), [Melia]], [],
    tv(1.0,[[tv(1.0, []), [Tryp]], [],
    tv(1.0,[[tv(1.0, []), [Sheraton]], [],
    tv(1.0,[[tv(0.5, []), [NH]], [],
    tv(1.0,[[tv(0.5, []), [Hilton]], [], []])])])]), [])
```

Fig. 12 Example of a *TV* obtained after evaluating a fuzzy condition

§4 Implementation

We have developed a prototype of our fuzzy XPath which is publicly available from `http://dectau.uclm.es/fuzzyXPath/`, equipped with a Web interface from which XPath queries can be tested. The implementation has been developed with our FLOPER tool. In what follows, we will give some details about FLOPER and the implementation of the fuzzy XPath in FLOPER.

Firstly, we would like to summarize the main elements of the FLOPER tool [41, 42, 43] which manages MALP programs. The parser of our FLOPER tool has been implemented by using the classical DCG's (*Definite Clause Grammars*) resource of the Prolog language, since it is a convenient notation for expressing grammar rules. Once the application is loaded inside a Prolog interpreter, it shows a menu which includes options for loading/compiling, parsing, listing and saving fuzzy programs, as well as for executing/debugging fuzzy goals. All these actions are based on the *compilation* of the fuzzy code into standard Prolog code.

```
member(X):- number(X),0=<X,X=<1.
bot(0).  top(1).  leq(X,Y):- X=<Y.
or_prod(X,Y,Z):- pri_prod(X,Y,U1),pri_add(X,Y,U2),pri_sub(U2,U1,Z).
and_prod(X,Y,Z):- pri_prod(X,Y,Z).
pri_prod(X,Y,Z):- Z is X * Y.
pri_add(X,Y,Z):- Z is X+Y.
pri_sub(X,Y,Z):- Z is X-Y.
```

Fig. 13 Example of multi-adjoint lattice

The key point of the compilation is to extend each atom with an extra argument, called *truth variable* of the form "$\_TV_i$", which is intended to contain the truth degree obtained after the subsequent evaluation of the atom. For instance, the first clause of Figure 3 is translated into:

```
p(X,_TV0):-q(X,Y,_TV1),r(Y,_TV2),or_prod(_TV1,_TV2,_TV3),and_prod(0.8,_TV3,_TV0).
```

Moreover, the remaining rules, become the pure Prolog facts "*q(a, Y, 0.9)*" and "*r(b,0.7)*", whereas the corresponding lattice is expressed by the clauses of Figure 13, where the meaning of the mandatory predicates *member*, *top*, *bot* and *leq* is obvious.
Finally, a fuzzy goal like "*p(X)*", is translated into the pure Prolog goal:

$$p(X, Truth\_degree)$$

(note that the last truth degree variable is not anonymous now) for which, after choosing option "*run*", the Prolog interpreter returns the desired fuzzy computed answer:

$$[Truth\_degree = 0.776, X = a]$$

Note that all internal computations (including compiling and executing) are pure Prolog derivations, whereas inputs (fuzzy programs and goals) and outputs (fuzzy computed answers) have always a fuzzy taste, thus producing the illusion on the final user of being working with a purely fuzzy logic programming tool. By using option "*lat*" ("*show*") of FLOPER, we can associate (and visualize) a new lattice to a given program. As seen before, such lattices must be expressed by means of a set of Prolog clauses (defining predicates *member, top, bot, leq* and the ones associated to fuzzy connectives) in order to be loaded into FLOPER.

Although the core of our implementation is written with (fuzzy) MALP rules, we have reused/adapted several modules of our previous Prolog-based implementation of (crisp) XPath described in [2, 1], which make use of the SWI-

```
and_prod(tv(X1,X2),tv(Y1,Y2),tv(Z1,Z2)):-
                          pri_prod(X1,Y1,Z1),pri_app(X2,Y2,Z2).

and_luka(tv(X,Elem1),tv(Y,Elem2),tv(Z,Elem0)):-
            pri_add(X,Y,U1), pri_sub(U1,1,U2),
            pri_max(U2,0.0,Z), pri_app(Elem1, Elem2, Elem0).

and_godel(tv(X,Elem1),tv(Y,Elem2),tv(Z,Elem0)):-
                  pri_min(X,Y,Z), pri_app(Elem1,Elem2,Elem0).

or_prod(tv(X1,X2),tv(Y1,Y2),tv(Z1,Z2)):- pri_prod(X1,Y1,U1),
                          pri_add(X1,Y1,U2),pri_sub(U2,U1,Z1),
                          pri_app(X2,Y2,Z2).

or_luka(tv(X,Elem1),tv(Y,Elem2),tv(Z,Elem0)):-
                  pri_add(X,Y,U1), pri_min(U1,1,Z),
                  pri_app(Elem1,Elem2,Elem0).

or_godel(tv(X,Elem1),tv(Y,Elem2),tv(Z,Elem0)):-
                  pri_max(X,Y,Z), pri_app(Elem1,Elem2,Elem0).

agr_aver(tv(X1,X2),tv(Y1,Y2),tv(Z1,Z2)):-  pri_add(X1,Y1,Aux),
                          pri_div(Aux,2,Z1),pri_app(X2,Y2,Z2).

agr_avg(tv(V1,E1),tv(V2,E2),tv(T1,F1),tv(T2,F2),tv(V3,G)):-
                  V3 is (V1*T1 + V2*T2)/(T1+T2),pri_app(E1,E2,A1),
                  pri_app(F1,F2,A2),pri_app(A1,A2,G).

pri_add(X,Y,Z)  :- Z is X+Y.          pri_sub(X,Y,Z) :-Z is X-Y.

pri_prod(X,Y,Z) :- Z is X * Y.        pri_div(X,Y,Z) :- Z is X/Y.

pri_app([],[],[]):-!.

pri_app([],A,A):-!.

pri_app(A,[],A):-!.

pri_app([Element,TV_Son,[]],TV_SibA,[Element,TV_Son,TV_SibA]):-!.

pri_app([Element,TV_Son,TV_sib],TV_SibA,[Element,TV_Son,TV_SibR]):-
                            pri_app(TV_sib,TV_SibA,[Element,TV_Son,TV_SibR]),!.
```

Fig. 14 Multi-adjoint lattice for *FuzzyXPath* (file "tv.pl")

```
fuzzyXPath([Label|LabelRest],[element(Label,_,Children)|Siblings],Deep,Down,TV_Iam):-
        fuzzyXPath(LabelRest,Children,Deep,Down,TV_Son),
        and_prod(Deep,TV_Son,TV_Son0),
        fuzzyXPath([Label|LabelRest],Siblings,Deep,Down,TV_Bro),
        and_prod(Down,TV_Bro,TV_Bro0),
        agr_fuse(tv(1,[element(Label,Attr,Children),[],[]]),TV_Son0,TV_Bro0,TV_body),
        and_prod(tv(1,[]),TV_body,TV_Iam).
```

Fig. 15 Prolog clause obtained by FLOPER after compiling a MALP rule

Prolog library for loading XML files in order to store each XML document by means of a Prolog term. For loading XML documents in our implementation we use the predicate `load_xml(+File,-Term)`, and similarly, we have a predicate `write_xml(+File,+Term)` for writing a data-term representing an XML

document into a file. And, of course, the parser of our application has been extended to recognize the new keywords *DEEP, DOWN, avg*, etc... with the proper arguments.

Furthermore, we have incorporated a lattice definition according to the definitions of Section 3.2, which is shown in Figure 14, where *and*, *or* and *avg* operators are defined by Prolog rules. Figure 15 shows the Prolog compilation from FLOPER of the rule of Figure 10. Finally, we have defined a predicate `tv_to_elem` to show the result in a pretty way which transforms the returned TV tree to an XML tree.

§5 Dynamic Filtering for Improving Efficiency

In [32, 31] we have reported some *thresholding* techniques specially tailored for the MALP language, where the main idea consists in to dynamically create and evaluate filters for prematurely disregarding those superfluous computations leading to non-significant solutions. Somehow inspired by the same guidelines, we have recently equipped our fuzzyXPath interpreter with a command with syntax «[FILTER=r]» (being r a real number between 0 and 1) which can be used just at the beginning of a query for indicating that only those answers with RSV greater of equal than r must be generated and reported. As we have described in [5], when «[FILTER=r]» precedes a fuzzy query, the interpreter *lazily* explores an input XML document for dynamically disregarding as soon as possible those branches of the XML tree leading to irrelevant solutions with an RSV degraded below r, thus allowing the possibility of efficiently managing large files without reducing the set of answers for which users are mainly interested in.

In order to explain the benefits of using the FILTER command, let us consider in this section the XML document shown in Figure 16, for which the execution of query «[//book[@year<2000 avg @price<50]/title» produces the following set of solutions:

```
<result>
  <title rsv="1.0">La Galatea</title>
  <title rsv="1.0">Los trabajos de Persiles y Sigismunda</title>
  <title rsv="1.0">La Celestina</title>
  <title rsv="1.0">El remedio en la desdicha</title>
  <title rsv="1.0">La Dragontea</title>
  <title rsv="0.5">Don Quijote de la Mancha</title>
  <title rsv="0.5">Hamlet</title>
  <title rsv="0.5">Romeo y Julieta</title>
```

```
<bib>
   <book year="2001" price="45.95">
      <title>Don Quijote de la Mancha</title>
      <author>Miguel de Cervantes Saavedra</author>
      <publications> <book year="1997" price="35.99">
                     <title>La Galatea</title>
                     <author>Miguel de Cervantes Saavedra</author>
                     <publications>
                              <book year="1994" price="25.99">
                              <title>Los trabajos de Persiles y Segismunda</title>
                              <author>Miguel de Cervantes Saavedra</author></book>
                     </publications></book>
      </publications></book>
   <book year="1999" price="25.65">
      <title>La Celestina</title>
      <author>Fernando de Rojas</author></book>
   <book year="2005" price="29.95">
      <title>Hamlet</title>
      <author>William Shakespeare</author>
      <publications>
         <book year="2000" price="22.5">
            <title>Romeo y Julieta</title>
            <author>William Shakespeare</author></book>
      </publications></book>
   <book year="2007" price="22.95">
      <title>Las ferias de Madrid</title>
      <author>Felix Lope de Vega y Carpio</author>
      <publications>
         <book year="1996" price="27.5">
            <title>El remedio en la desdicha</title>
            <author>Felix Lope de Vega y Carpio</author> </book>
         <book year="1998" price="12.5">
            <title>La Dragontea</title>
            <author>Felix Lope de Vega y Carpio</author></book>
      </publications></book>
</bib>
```

Fig. 16 Input XML document collecting books

```
  <title rsv="0.5">Las ferias de Madrid</title>
</result>
```

If we consider now the new, quite similar query «[FILTER=0.4]//book[@year <2000 avg @price<50]/title», we clearly obtain again nine answers, but only five if we fix «[FILTER=0.8]». Obviously, we would hope that the runtime of the second case should be lower than the first one since, as our approach does, there is no need for computing all solutions and then filtering the best ones. This desired dynamic behaviour when avoiding useless computations is reflected in Figure 17 which considers the effort needed for executing (excluding parsing/-compiling time) a query like «[FILTER=r]//book[(@price>25 and @price<30) avg (@year<2000 or @year>2006)]»where each row represents the size of several XML files accomplishing with the same structure of our running example (but considering different nesting levels of tags *book*, *title*, *author* and *publications*),

Records	FILTER								
	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
1000	1.766	1.696	1.734	0.842	0.469	0.268	0.221	0.087	0.056
2000	6.628	6.432	6.998	3.242	1.439	0.677	0.599	0.168	0.122
3000	14.532	14.023	14.059	6.306	2.831	1.257	1.101	0.253	0.179
4000	25.535	24.684	24.722	10.883	4.827	1.918	1.794	0.345	0.242
5000	41.522	37.782	37.166	16.201	7.242	2.993	2.516	0.427	0.281
6000	58.905	55.354	55.596	24.411	10.993	4.207	3.554	0.554	0.373
7000	85.167	85.652	82.733	37.748	14.436	5.083	4.653	0.649	0.460
8000	137.737	102.816	102.763	69.401	26.680	8.273	5.894	0.690	0.481
9000	175.272	131.828	131.021	56.937	22.601	7.869	7.329	0.824	0.549
10000	195.613	185.201	167.676	95.286	26.649	9.516	9.595	0.973	0.742

Fig. 17 Performance of fuzzy XPath by using FILTER on XML files with growing sizes

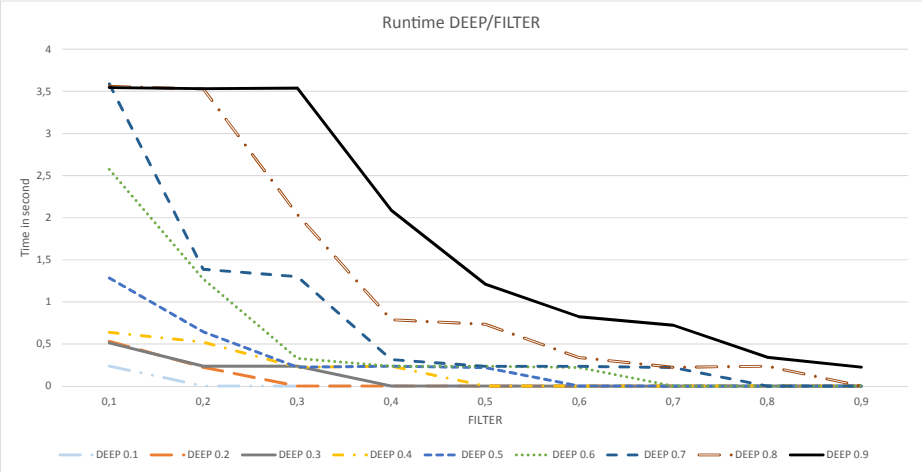

Fig. 18 Runtime for several fuzzy XPath queries varying DEEP and FILTER

and each column refers to a different degree of the FILTER command. Here, the runtime is measured in seconds (the benchmarks have been performed using a computer with processor Intel Core Duo, with 2 GB RAM and Windows Vista) and each record in the input file refers to a different *book* (that is, the number of records coincides with the number of occurrences of tag *book*) which might contain other books inside its *publications* tag.

Moreover, in Figure 18 we continue with a similar query to the previous one, but also considering the DEEP command[2]. Here, for a large XML document with a fixed size, we express the number of seconds needed for executing such query when varying FILTER and DEEP, where it is easy to see that the

[2] This kind of statistics can be produced on-line for several XML files and fuzzy XPath queries via the following URL that we have just prepared for the interested reader: `http://dectau.uclm.es/fuzzyXPath/fuzzyXPathEstatistic2.php\#testing`.

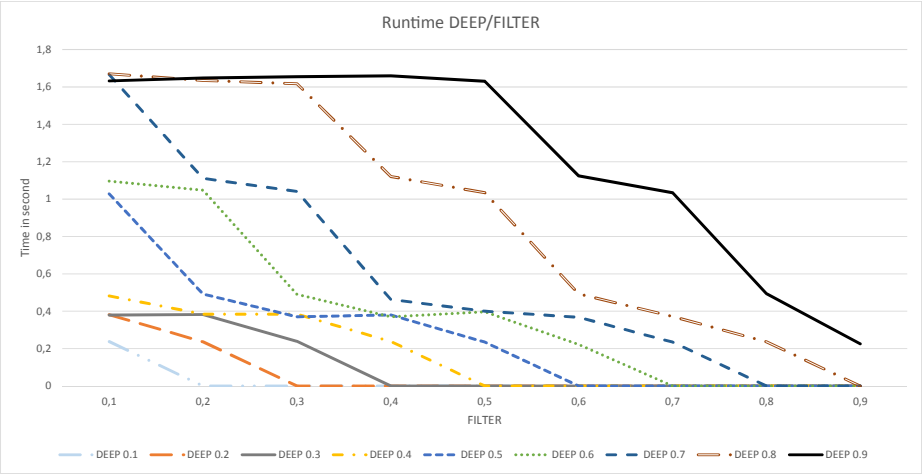

Fig. 19 Varying DEEP and FILTER in a query using $avg\{30,1\}$

behaviour is more and more improved whenever FILTER grows and DEEP decreases, as wanted. In contrast with the previous query which is based on the *avg* command, we can now use the *priorized version* of such operator which let us to give different degrees of importance to its arguments (remember that, for two given *RSV*'s r_1 and r_2, $avg\{p_1,p_2\}$ is computed by $r_3 = (r_1{*}p_1{+}r_2{*}p_2)/(p_1{+}p_2)$) and hence, if in the previous query we use $avg\{30,1\}$ instead of standard *avg*, we indicate that the first sub-condition (i.e., @price>25 and @price<30) is 30 times more important than the second one (i.e., @year<2000 or @year>2006), whereas $avg\{1,30\}$ represent the inverse criterion. In Figures 19 and 20 we provide statistics in the same way than in Figure 18, but using now *average with priorities* 30-1 and 1-30, respectively.

In order to implement the FILTER command in our interpreter, the key point consists in introducing two extra parameters on the main predicate of our application, which now looks like: `fuzzyXPath(+ListXPath,+Tree,+Deep,+Down,+Filter,+Accum)`. So, whereas *ListXPath* is the Prolog representation of an XPath expression and *Tree* is the term representing an input XML document, the values for DEEP/DOWN/FILTER have the obvious meaning, and the last argument *Accum* (which is appropriately updated -maybe decreased- when going deeper in the exploration of the file) accumulates the sequence of penalties produced till reaching a concrete node, and it is very useful for deciding when performing a recursive call to the children of such node whenever the value of *Accum* is better than the one fixed by FILTER.

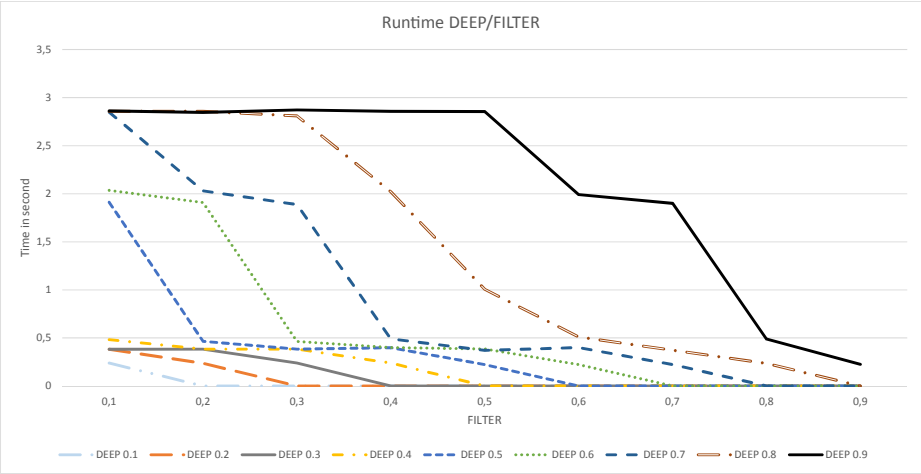

Fig. 20 Varying DEEP and FILTER in a query using $avg\{1, 30\}$

§6 Conclusions and Future Work

In this paper we have described the foundations and implementation of a flexible extension based on fuzzy logic programming of the well-known XPath language. The new fuzzy XPath dialect takes profit of the underlying source MALP language for easily modeling a wide range of flexible operators representing different versions of conjunctions, disjunctions and other highly expressive hybrid operators for retrieving data from XML documents, as well as for constraining queries with structural and thresholding conditions. Fuzzy XPath has been integrated in the MALP framework, by providing semantics to fuzzy logic programs that work with trees with truth values. We have described the implementation which is publicly available from `http://dectau.uclm.es/fuzzyXPath/` and has been coded as a set of MALP (*Multi-Adjoint Logic Programming*) rules developed under the FLOPER (*Fuzzy LOgic Programming Environment for Research*) system built in our research group. Additionally, we have shown benchmarks of performance of our system, improved by dynamic filtering. In `http://dectau.uclm.es/fuzzyXPath/` there are examples of use of our language which can be executed online.

As future work we plan the following research lines. Firstly, we are interested to extend our dialect of XPath with the handling of text content in the same line as Full-text XPath [18]. Including content analysis, our fuzzy XPath will gain on expressivity and flexibility. Secondly, we are interested in top-k answering. Top-k answering has been already studied for fuzzy logic programming

[56], and can be adapted to FLOPER. Thirdly, we find that MALP and FLOPER can be used in ontologies and the Semantic Web, following [54, 37, 26].

Acknowledgment

This work was supported by the EU (FEDER), and the Spanish MINECO Ministry (*Ministerio de Economía y Competitividad*) under grants TIN2013-45732-C4-2-P and TIN2013-44742-C4-4-R, as well as by the Andalusian Regional Government (Spain) under Project P10-TIC-6114.

References

1) J. M. Almendros-Jiménez. An Encoding of XQuery in Prolog. In *Proceedings of the Sixth International XML Database Symposium XSym'09*, pages 145–155, Heildelberg, Germany, 2009. Springer, Lecture Notes in Computer Science 5679.

2) J. M. Almendros-Jiménez, A. Becerra-Terón, and Francisco J. Enciso-Baños. Querying XML documents in logic programming. *Theory and Practice of Logic Programming*, 8(3):323–361, 2008.

3) Jesús M Almendros-Jiménez, Alejandro Luna Tedesqui, and Ginés Moreno. Annotating 'fuzzy chance degrees" when debugging xpath queries. In *Advances in Computational Intelligence*, pages 300–311. Springer, 2013.

4) Jesús Manuel Almendros-Jiménez, Alejandro Luna, and Ginés Moreno. Fuzzy Logic Programming for Implementing a Flexible XPath-based Query Language. *Electr. Notes Theor. Comput. Sci.*, 282:3–18, 2012.

5) Jesús Manuel Almendros-Jiménez, Alejandro Luna Tedesqui, and Ginés Moreno. Dynamic filtering of ranked answers when evaluating fuzzy xpath queries. In *Rough Sets and Current Trends in Computing - 9th International Conference, RSCTC 2014, Granada and Madrid, Spain, July 9-13, 2014. Proceedings*, volume 8536 of *Lecture Notes in Computer Science*, pages 319–330. Springer, 2014.

6) J.M. Almendros-Jiménez, A. Luna, and G. Moreno. A Flexible XPath-based Query Language Implemented with Fuzzy Logic Programming. In *Proc. of 5th International Symposium on Rules: Research Based, Industry Focused, RuleML'11. Barcelona, Spain, July 19–21*, pages 186–193, Heidelberg,Germany, 2011. Springer Verlag, Lecture Notes in Computer Science 6826.

7) Sihem Amer-Yahia, SungRan Cho, and Divesh Srivastava. Tree pattern relaxation. In *Advances in Database Technology–EDBT 2002*, pages 496–513. Springer, 2002.

8) Sihem Amer-Yahia, Nick Koudas, Amélie Marian, Divesh Srivastava, and David Toman. Structure and content scoring for xml. In *Proceedings of the 31st international conference on Very large data bases*, pages 361–372. VLDB Endowment, 2005.

9) Sihem Amer-Yahia, Laks VS Lakshmanan, and Shashank Pandit. FleXPath: flexible structure and full-text querying for XML. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 83–94. ACM, 2004.

10) J. F. Baldwin, T. P. Martin, and B. W. Pilsworth. *Fril- Fuzzy and Evidential Reasoning in Artificial Intelligence*. John Wiley & Sons, Inc., 1995.

11) A. Berglund, S. Boag, D. Chamberlin, M.F. Fernandez, M. Kay, J. Robie, and J. Siméon. XML path language (XPath) 2.0. *W3C*, 2010.

12) P. Bosc and O. Pivert. SQLf: a relational database language for fuzzy querying. *Fuzzy Systems, IEEE Transactions on*, 3(1):1–17, 1995.

13) P. Buche, J. Dibie-Barthélemy, O. Haemmerlé, and G. Hignette. Fuzzy semantic tagging and flexible querying of XML documents extracted from the Web. *Journal of Intelligent Information Systems*, 26(1):25–40, 2006.

14) P. Buche, J. Dibie-Barthélemy, and F. Wattez. Approximate querying of XML fuzzy data. *Flexible Query Answering Systems*, pages 26–38, 2006.

15) B.P. Buckles and F.E. Petry. A fuzzy representation of data for relational databases. *Fuzzy Sets and Systems*, 7(3):213–226, 1982.

16) B.P. Buckles, F.E. Petry, and H.S. Sachar. A domain calculus for fuzzy relational databases. *Fuzzy Sets and Systems*, 29(3):327–340, 1989.

17) A. Campi, E. Damiani, S. Guinea, S. Marrara, G. Pasi, and P. Spoletini. A fuzzy extension of the XPath query language. *Journal of Intelligent Information Systems*, 33(3):285–305, 2009.

18) Pat Case, Michael Dyck, Mary Holstege, Sihem Amer-Yahia, Chavdar Botev, Stephen Buxton, Jochen Doerre, Jim Melton, Michael Rys, and Jayavel Shanmugasundaram. XQuery and XPath Full Text 1.0. *W3C*, 2011.

19) E. Damiani, S. Marrara, and G. Pasi. FuzzyXPath: Using fuzzy logic and IR features to approximately query XML documents. *Foundations of Fuzzy Logic and Soft Computing*, pages 199–208, 2007.

20) E. Damiani, S. Marrara, and G. Pasi. A flexible extension of XPath to improve XML querying. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 849–850. ACM, 2008.

21) B. Fazzinga, S. Flesca, and F. Furfaro. On the expressiveness of generalization rules for XPath query relaxation. In *Proceedings of the Fourteenth International Database Engineering & Applications Symposium*, pages 157–168. ACM, 2010.

22) B. Fazzinga, S. Flesca, and A. Pugliese. Top-k Answers to Fuzzy XPath Queries. In *Database and Expert Systems Applications*, pages 822–829. Springer, 2009.

23) Bettina Fazzinga, Sergio Flesca, and Filippo Furfaro. XPath query relaxation through rewriting rules. *Knowledge and Data Engineering, IEEE Transactions on*, 23(10):1583–1600, 2011.

24) Bettina Fazzinga, Sergio Flesca, and Andrea Pugliese. Retrieving xml data from heterogeneous sources through vague querying. *ACM Transactions on Internet Technology (TOIT)*, 9(2):7, 2009.

25) Bettina Fazzinga, Sergio Flesca, and Andrea Pugliese. Top-k approximate answers to xpath queries with negation. *IEEE Trans. Knowl. Data Eng.*, 26(10):2561–2573, 2014.

26) A. Formica. Semantic Web search based on rough sets and Fuzzy Formal Concept Analysis. *Knowledge-Based Systems*, 2011.

27) A. Gaurav and R. Alhajj. Incorporating fuzziness in XML and mapping fuzzy relational data into fuzzy XML. In *Proceedings of the 2006 ACM symposium on Applied computing*, pages 456–460. ACM, 2006.

28) P. Gursky, T. Horvath, R. Novotny, V. Vanekova, and P. Vojtas. UPRE: User preference based search system. In *Web Intelligence, 2006. WI 2006. IEEE/WIC/ACM International Conference on*, pages 841–844. IEEE, 2006.

29) E. Herrera-Viedma and G. Pasi. Fuzzy approaches to access information on the Web: recent developments and research trends. In *Proc. International Conference on Fuzzy Logic and Technology (EUSFLAT 2003)*, pages 25–31, 2003.

30) M. Ishizuka and N. Kanai. Prolog-ELF Incorporating Fuzzy Logic. In Aravind K. Joshi, editor, *Proceedings of the 9th Int. Joint Conference on Artificial Intelligence, IJCAI'85*, pages 701–703. Morgan Kaufmann, 1985.

31) P. Julián, J. Medina, P.J. Morcillo, G. Moreno, and M. Ojeda-Aciego. An unfolding-based preprocess for reinforcing thresholds in fuzzy tabulation. In Ignacio Rojas, Gonzalo Joya Caparrós, and Joan Cabestany, editors, *IWANN (1)*, volume 7902 of *Lecture Notes in Computer Science*, pages 647–655. Springer, 2013.

32) P. Julián, J. Medina, G. Moreno, and M. Ojeda. Efficient thresholded tabulation for fuzzy query answering. *Studies in Fuzziness and Soft Computing (Foundations of Reasoning under Uncertainty)*, 249:125–141, 2010.

33) P. Julián, G. Moreno, and J. Penabad. On the declarative semantics of multi-adjoint logic programs. In J. Cabestany et al., editor, *Bio-Inspired Systems: Computational and Ambient Intelligence, 10th International Work-Conference on Artificial Neural Networks, IWANN'09, Salamanca, Spain, June 10-12, 2009, Proceedings, Part I*, volume 5517 of *Lecture Notes in Computer Science*, pages 253–260. Springer, 2009.

34) D.H. Kraft, G. Pasi, and G. Bordogna. Vagueness and uncertainty in information retrieval: how can fuzzy sets help? In *Proceedings of the 2006 international workshop on Research issues in digital libraries*, page 3. ACM, 2006.

35) R.C.T. Lee. Fuzzy Logic and the Resolution Principle. *Journal of the ACM*, 19(1):119–129, 1972.

36) Hua-Gang Li, S Alireza Aghili, Divyakant Agrawal, and Amr El Abbadi. Flux: Fuzzy content and structure matching of xml range queries. In *Proceedings of the 15th international conference on World Wide Web*, pages 1081–1082. ACM, 2006.

37) T. Lukasiewicz and U. Straccia. Managing uncertainty and vagueness in description logics for the semantic web. *Web Semantics: Science, Services and Agents on the World Wide Web*, 6(4):291–308, 2008.

38) Amelie Marian, Sihem Amer-Yahia, Nick Koudas, and Divesh Srivastava. Adaptive processing of top-k queries in xml. In *Data Engineering, 2005. ICDE 2005. Proceedings. 21st International Conference on*, pages 162–173. IEEE, 2005.

39) M.J. Martín-Bautista, D.H. Kraft, MA Vila, J. Chen, and J. Cruz. User profiles and fuzzy logic for web retrieval issues. *Soft Computing-A Fusion of Foundations, Methodologies and Applications*, 6(5):365–372, 2002.

40) J. Medina, M. Ojeda-Aciego, and P. Vojtáš. Similarity-based Unification: a multi-adjoint approach. *Fuzzy Sets and Systems*, 146:43–62, 2004.

41) P.J. Morcillo and G. Moreno. Programming with Fuzzy Logic Rules by using the FLOPER Tool. In Nick Bassiliades et al., editor, *Proc of the 2nd. Rule Representation, Interchange and Reasoning on the Web, International Symposium, RuleML'08*, pages 119–126. Springer Verlag, Lecture Notes in Computer Science 3521, 2008.

42) P.J. Morcillo, G. Moreno, J. Penabad, and C. Vázquez. A Practical Management of Fuzzy Truth Degrees using FLOPER. In M. Dean et al., editor, *Proc. of 4nd Intl Symposium on Rule Interchange and Applications, RuleML'10*, pages 20–34. Springer Verlag, Lecture Notes in Computer Science 6403, 2010.

43) P.J. Morcillo, G. Moreno, J. Penabad, and C. Vázquez. Fuzzy Computed Answers Collecting Proof Information. In J. Cabestany et al., editor, *Advances in Computational Intelligence - Proc of the 11th International Work-Conference on Artificial Neural Networks, IWANN 2011*, pages 445–452. Springer Verlag, Lecture Notes in Computer Science 6692, 2011.

44) Susana Muñoz-Hernández, Victor Pablos Ceruelo, and Hannes Strass. RFuzzy: Syntax, semantics and implementation details of a simple and expressive fuzzy tool over Prolog. *Information Sciences*, 181(10):1951–1970, 2011.

45) B. Oliboni and G. Pozzani. Representing fuzzy information by using XML schema. In *Database and Expert Systems Application, 2008. DEXA'08. 19th International Workshop on*, pages 683–687. IEEE, 2008.

46) B. Oliboni and G. Pozzani. An XML Schema for Managing Fuzzy Documents. *Soft Computing in XML Data Management*, pages 3–34, 2010.

47) G. Pasi. Flexible information retrieval: some research trends. *Mathware & soft computing*, 9(1):107–121, 2008.

48) G. Pasi. Fuzzy sets in information retrieval: state of the art and research trends. *Fuzzy Sets and Their Extensions: Representation, Aggregation and Models*, pages 517–535, 2008.

49) M. Rodríguez and C. A. Romero. A declarative semantics for clp with qualification and proximity. *Theory and Practice of Logic Programming*, 10(4-6):627–642, 2010.

50) M. Rodríguez-Artalejo and C. Romero-Díaz. Quantitative logic programming revisited. In J. Garrigue and M. Hermenegildo, editors, *Functional and Logic Programming (FLOPS'08)*, pages 272–288. Springer Lecture Notes in Computer Science 4989, 2008.

51) S. Schockaert, N. Makarytska, and M. De Cock. Fuzzy methods on the web: a critical discussion. *35 Years of Fuzzy Set Theory*, pages 237–266, 2011.

52) S. Shenoi and A. Melton. Proximity relations in the fuzzy relational database model. *Fuzzy Sets and Systems*, 31(3):285–296, 1989.

53) S. Shenoi and A. Melton. An extended version of the fuzzy relational database model. *Information Sciences*, 52(1):35–52, 1990.

54) U. Straccia. Towards a fuzzy description logic for the semantic web (preliminary report). *The Semantic Web: Research and Applications*, pages 73–123, 2005.

55) U. Straccia. Managing uncertainty and vagueness in description logics, logic programs and description logic programs. In *Reasoning Web, 4th International Summer School, Tutorial Lectures*, number 5224 in Lecture Notes in Computer Science, pages 54–103. Springer Verlag, 2008.

56) Umberto Straccia and Nicolás Madrid. A top-k query answering procedure for fuzzy logic programming. *Fuzzy Sets and Systems*, 205:1–29, 2012.

57) Martin Theobald, H. Bast, Debapriyo Majumdar, Ralf Schenkel, and Gerhard Weikum. Topx: efficient and versatile top-k query processing for semistructured data. *VLDB J.*, 17(1):81–115, 2008.

58) P. Vojtáš and L. Paulík. Query answering in normal logic programs under uncertainty. In L. Godó, editor, *Proc. of 8th. European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU-05), Barcelona, Spain*, pages 687–700. Lecture Notes in Computer Science 3571, Springer Verlag, 2005.

59) L. Yan, ZM Ma, and J. Liu. Fuzzy data modeling based on XML schema. In *Proceedings of the 2009 ACM symposium on Applied Computing*, pages 1563–1567. ACM, 2009.