An Approach to Representing Uncertainty Rules in RuleML*

Carlos Viegas Damásio
Centro de Inteligêncıa Artificial,
Universidade Nova de Lisboa, Portugal,
cd@di.fct.unl.pt

Jeff Z. Pan
Department of Computing Science,
University of Aberdeen, UK
jpan@csd.abdn.ac.uk

Giorgos Stoilos
National Technical University of Athens, Greece
Electrical and Computer Engineering
gstoil@image.ece.ntua.gr

Umberto Straccia
ISTI - Italian National Research Council
Pisa, Italy
Umberto.Straccia@isti.cnr.it

Abstract

The RuleML initiative defines a normalized markup for expressing and exchange rules in the Semantic Web. However, the syntax of the language is still limited and lacks features for representing rule-based languages capable of handling uncertainty. It is desirable to have a general extension of RuleML which accommodates major existing languages proposed in the latest two decades. The main contribution of the paper is to propose such a general extension, showing how to encode many of the existing languages in this extension. We hope this work can also provide some insights on how to cover uncertainty in the RIF framework.

1 Introduction

Rules in the Web have become a mainstream topic these days. On the one hand, inference rules can be marked up for e-applications, such as e-commerce and e-science; on the other hand, transformation can be used for document generations and ontology reuse. Recently, rule interchange has been widely considered as an important issue - the World Wide Web Consortium (W3C) has set up a Rule Interchange Format (RIF) Working Group to tackle this issue.

Representing and handling uncertainty has always been a fundamental issue in Knowledge Representation and Artificial Intelligence. This research effort resulted in a plethora of formalisms with different motivation and applications. For example, the size as well as the dynamic aspect of the Web indicate the usefulness of rules handling uncertainty information. Indeed, the charter of the RIF Working Group requires an extensible format to handle uncertainty rules.

The RuleML initiative is probably the earliest effort that defines a normalized markup for expressing and exchanging rules in the Semantic Web. It is a modular markup language designed for expressing knowledge bases in XML and XML/RDF. Currently, RuleML has syntactic mechanisms for encompassing a series of rule languages, ranging from Datalog to HILOG. However, the support for representing and associating uncertainty to rules and facts is rather limited. In particular, the current framework lacks a general mechanism to accommodate major existing languages proposed in the latest two decades.

The main purpose of the paper is to propose such a general extension of uncertainty rules for RuleML. On the syntax aspect, it is adopted the existing markup. On the semantic aspect, we try to be as general as possible, and to cover several categories of general mechanisms, in which languages can be parameterized by the user in order to convey specific semantic information. Different illustrative rule languages are briefly presented and aligned with the proposed syntax, in order to discuss the pros and cons of the several alternatives. We hope this work can also provide some insights on how to cover uncertainty in the RIF framework.

The rest of the paper is organised as follows. The next section briefly presents an illustrative set of uncertainty rule languages. Section 3 proposes a general extension of RuleML to accommodate these uncertainty rule languages. Such an extension is used in Section 4 to illustrate the encoding of the previous languages. Section 5 discusses default interpretation of connectives in our uncertainty extension of RuleML, and Section 6 concludes the paper by briefly analysing the encodings and summarising the alternatives.

*This research has been partially funded by European Commission and by the Swiss Federal Office for Education and Science within the 6th Framework Programme projects REWERSE (IST-2004-506779) and Knowledge Web (IST-2004-507842).
2 Some Existing Uncertainty Rule Languages

The literature in Logic Programming contains a spate of semantics and languages for handling uncertainty; we call them uncertainty rule languages. One of the first well known related languages is Van Emden’s quantitative deduction rules [34], which are an extension of Horn clauses with an “attenuation factor”:

\[ A \leftarrow q \cdot B_1 \land \ldots \land B_n, \quad n \geq 0 \]  

A quantitative deduction rule (1) is formed by atoms \( A \) and \( B_1 \ldots B_n \), and the attenuation factor \( q \), a real number in the interval \([0, 1]\). Truth-values of atoms are interpreted in the closed unit interval \([0, 1]\). A ground quantitave deduction rule is true in an interpretation \( I \) iff \( I(A) \geq q \times \min \{I(B_i) \mid i \in \{1, \ldots, n\}\} \).

In what follows, we briefly present some categories (namely, implication-based, annotation-based and probabilistic-based) of uncertainty rule languages, in order to illustrate the existing diversity of proposals and syntactical constructs. Interestingly, some of the following languages actually generalise Van Emden’s quantitative deduction rules. Note that, however, the section is not intended to be an exhaustive survey of such languages, which will be considered in a forthcoming publication.

2.1 Implication-based Approaches

Implication-based approaches usually depart from a many-valued logic and associate truth-values, weights or degrees to rules and facts. Normally the underlying logic is truth-functional.

2.1.1 Fuzzy Logic Program

A typical implication-based language is Vojtáš and Paulík [37]’s fuzzy logic programming. It is a generalization of definite logic programming, where programs are constructed from an implication connective (say \( \leftarrow \)), with corresponding t-norm adjunction (resp. \( \otimes_1 \)), and another t-norm operator denoted by \( \otimes_2 \). A t-norm is a generalization to the many-valued setting of the conjunction connective. In their setting, a fuzzy rule is of the form:

\[ A \leftarrow B_1 \otimes_2 \ldots \otimes_2 B_n \text{ with-cf } q \]  

where a rational number \( q \) in \([0,1]\] expresses a confidence factor, and \( A \) and \( B_1, \ldots B_n \) are atoms with truth-value in the unit interval \([0, 1]\]. Van Emden’s quantitative deduction rules [34] can be seen as a special form of fuzzy rules, where \( \leftarrow \) is Goguen implication, with \( \otimes_1 \) as its adjunction (product), and \( \otimes_2 \) is Gödel t-norm (minimum). Note that the restriction to the carrier \([0, 1]\] is not essential and was lifted in [9], continuing the initial work of Vojtáš and Paulík.

2.1.2 Possibilistic Logic Program

Possibilistic Logic Programs [13] can be seen as a special form of fuzzy logic programs. A possibilistic logic program is a finite set of (first-order) possibilistic Horn clauses annotated only with necessity degrees of the following form:

\[ A \leftarrow B_1 \land \ldots \land B_n \quad (N\alpha) \]  

where \( A, B_1, \ldots, B_n \) are propositional symbols and \( (N\alpha) \) is the necessity degree of the clause with \( 0 \leq \alpha \leq 1 \). Possibilistic Logic Programming [13] and Ordinary Probabilistic Logic Programs [23] are all particular instances of Quantitative Deduction; thus they can also be captured by Fuzzy Logic Programming. For instance, the embedding of Possibilistic Logic Programming is straightforward where the underlying implication and conjunctor are the corresponding Gödel’s connectives.

2.1.3 f-SWRL

Another implication-based language is the recently proposed f-SWRL language [28]. f-SWRL provides OWL DL [25] axioms (but with fuzzy interpretation) as well as fuzzy rule axioms of the following form:

\[ A \ast w \leftarrow B_1 \ast w_1 \land \ldots \land B_n \ast w_n \]  

where \( A, B_1, \ldots, B_n \) are either concepts (unary predicates) or properties (binary predicates) used in OWL DL axioms, and the weights \( w_1, \ldots w_n \) and \( w \) are real numbers in the unit interval. f-SWRL provides a framework to accommodate different operations (such as fuzzy intersection, union, negation, implication as well as weight operations) as long as they conform to the key constraints of f-SWRL, such as that the degree of fuzzy implication should be no less than the weight of the head, and that fuzzy assertions are special forms of fuzzy rule axioms, which requires allowing the consequent to be a constant. As f-SWRL knowledge bases also contains fuzzy concept and role axioms from fuzzy OWL, they are not special forms of Vojtáš and Paulík [37]’s fuzzy logic programs, nor are they special forms of the annotated logic programs to be introduced below.

2.2 Annotation-based Approaches

In annotation-based approaches, rules keep a classical interpretation while uncertainty is associated with atoms, not with the implication. Generalized Annotated Logic Programs (GAPs) is the fundamental formalism [18] in this approach. Annotated rules are of the form:

\[ A : f(\mu_1, \ldots, \mu_n) \leftarrow B_1 : \mu_1 \& \ldots \& B_n : \mu_n \]
where $\mu_1, \ldots, \mu_n$ are either annotation constants or annotation variables and $f$ is a total, continuous and computable function. The intuitive reading of annotated rules is if $B_1 \geq \mu_1, \ldots, B_n \geq \mu_n$, then $A \geq f(\mu_1, \ldots, \mu_n)$. Annotations denote elements in such given upper-semilattice, used as underlying truth-value space, where $\geq$ is the corresponding order. Note that annotation variables cannot be used as object variables in the atoms, and vice-versa. Van Emden’s quantitative deduction rules can be represented as annotated rules of the following forms:

$$A : q \times \min(\mu_1, \ldots, \mu_n) \leftarrow B_1 : \mu_1 \& \ldots \& B_n : \mu_n,$$

where $\mu_1, \ldots, \mu_n$ are annotation variables.

Other annotation-based approaches include, e.g., Signed Formula Logic Programming (SFLP) [22], which theoretically has the same expressive power as GAPs [22].

### 2.3 Probabilistic-based Approaches

A third kind of approach of uncertain rule languages is based on probability theory. These are the most complex languages since the underlying connectives are not truth-functional, requiring complex definitions and approaches.

### 2.3.1 Hybrid Probability Logic Program

Hybrid Probabilistic Logic Program [12] are an adaptation of Generalized Annotated Logic Programs to deal with probabilistic reasoning, but with a different semantics. In this approach, the notion of probabilistic strategy is introduced because there is no single “formula” for computing the probability of a complex event $(e_1 \land e_2)$ where $e_1$ and $e_2$ are primitive events [12]. Due to the limitation of space, we refer the reader to [12] for more details about probabilistic strategies.

A Hybrid Probabilistic Logic Program over the set $S$ of probabilistic-strategies is a finite set of hp-rules of the form:

$$F_0 : \mu_0 \leftarrow F_1 : \mu_1 \land \ldots \land F_k : \mu_k$$

where each $F_i : \mu_i$ is an hp-annotated basic formula over $S$. Intuitively, an hp-rule means that “if the probability of $F_1$ falls in the interval $\mu_1$ and . . . and the probability of $F_k$ falls within the interval $\mu_k$, then the probability of $F_0$ lies in the interval $\mu_0$”. The $F_i$s are designated hybrid basic formulas and are either applications of conjunctive ($B_1^l \land \ldots \land B_n^l$) or disjunctive strategies ($B_1^l \lor \ldots \lor B_n^l$) to finite sets of distinct atoms ($B_1^{l_1}, \ldots, B_n^{l_1}$), encoding complex events. Intervals are pairs $[e_1, e_2]$ of reals numbers in the unit interval. Hybrid Probabilistic Logic Programs have been further generalized to capture temporal aspects in real-world applications [11], in particular annotations are more complex since they contain a time dimension.

#### 2.3.2 Bayesian logic program

Finally, we consider Bayesian logic programs [17] which consist of a (finite) set of Bayesian rules of the form:

$$A \mid B_1, \ldots, B_n, \quad n \geq 0$$

The distinctive feature of Bayesian Logic Programs is that for each clause $c$ there is exactly one conditional probability distribution $cpd(c)$, and for each Bayesian predicate $p/l$ there is exactly one combining rule $cr(p/l)$. It is usually assumed that $cpd(c)$ is represented as a table; other possible representations are decision trees and rules! The distribution $cpd(c)$ generically represents the conditional probability distributions associated with each ground instance of the corresponding clause, while the combining rule expresses how the different probability distributions of clauses for a given predicate are combined; an often used combining rule is noisy-or. Bayesian networks are a particular case of Bayesian Logic Programs.

Other probabilistic-based approaches include Probabilistic programs [19], Stochastic Logic Programs [6], etc.

### 3 A General Uncertainty Extension

In this section, we propose a fuzzy extension of RuleML which attains the following “conflicting” objectives:

- to extend RuleML with a basic and modular set of constructs;
- to be general enough to accommodate main existing rule-based languages dealing with uncertainty;
- to be natural and easy usable by the user;
- to adopt language defaults that are transparent and reasonable to the user;

The existing RuleML 0.9 version already provides the attribute @weight in element <slot>. Attribute @weight is used to express a slot relative weight with respect to its siblings, and has been applied to encode in RuleML node-labeled, arc-labeled, weight-labeled trees [3]. This relevance measure is used in [3] to define semantic matching between trees, and usually these weights are normalized real numbers in the unit interval $[0, 1]$. In this paper we ignore this important issue of similarity and ranking, and possible extensions, which should be incorporated in a full-fledged Fuzzy RuleML framework in particular to represent and reason with fuzzy data. Some proposals already support these notions, like the ones described in [2, 29, 5].

More interesting for our objectives is the element <degree>, a child of element <Atom> and <Equal> in RuleML 0.9. This was originally intended to represent “an optional truth value (between 0 and 1) that may be assigned
to facts and rules,” as proposed in [30]. Also important, RuleML 0.9 defines the attribute @kind which is allowed in solely in the <Implies> element, for choosing between first-order and logic programming rules. The original terminology of RuleML is adopted and adapted to achieve the design goals of our uncertainty extension.

From an attentive analysis of the literature, and in particular of from the previous set of languages, it can be concluded that are some common features:

- most of the languages use implication symbols to represent rules;
- most of the languages, except annotated ones, attach to rules confidence degrees, probabilities, weights, conditional probability tables, etc.
- to different languages usually correspond different types of implication, conjunction and disjunction operators in the rules, some of them even allow different operators in the same rule base;
- some languages permit combination of complex formula in the body and in the head of rules, which surpass the simple conjunctions and disjunctions;
- annotation-based languages attach complex annotations to atoms, and even to formulae;
- some languages use parameters to specify the behaviour of rules;
- some languages adopt general truth-values structures, namely lattices and residuated lattices.

In order to achieve the objectives stated at the beginning of this section, our concrete proposal consists in extending the RuleML 0.9 by:

- adding @mapKind to performatives <Assert>, <Query> and <Protect>.
- permitting the use of @kind in <Atom>, besides in <Implies>, as well as in any other RuleML connective <Equivalent>, <Integrity>, <And>, <Or>, <Neg>, and <Naf>.
- the optional element <degree> is allowed in the previous RuleML connectives.

The @kind attribute is used to specify semantic information regarding the construct (e.g. t-norm or implication used). The attribute @mapKind in the performatives expresses the (default) value of the @kind attribute of the performative child element(s); this is a technique adopted in RuleML designated attribute mapping. This simplifies writing of rule bases, without requiring repetitive declarations of the intended interpretation of connectives. In order to associate weights, annotation, or probability, or truth-value associated with complex formula, the element <degree> is used. These amendments have a reduced impact in the RuleML language, and is downward compatible with the existing syntax. For the sake of completeness, the abstract syntax is presented in Figures 1 and 2, in the style of [16]. We prefer to use the normalized striped syntax, and therefore ignoring stripe skipping in the content models. The differences to RuleML 0.9 are marked in bold in the figures. Notice that some of RuleML elements are context dependent, and the reader is referred to [16] for the allowed combinations. Concrete fragments of XML markup can be found in the examples of next section.

An implicit and major design decision regards the syntactic coexistence of annotation and implication based approaches. Annotated atoms are captured by the new attribute @kind in the <Atom> element:

```
<Atom kind="gap">
  <degree>
    <Data xsi:type="xsd:decimal">0.5</Data>
  </degree>
  <op><Rel>prop</Rel></op>
</Atom>
```

Notice also the use of element <degree> to associate the corresponding annotation. Notice that this annotation might also be a variable or a complex annotation (only in head of rules). Similarly, signed formula logic programming [4] can be encoded in our uncertainty extension, but where degrees are sets of constants or even complex propositional formula. In order to be able to handle the more complex languages like Hybrid Probabilistic Logic Programs, the attribute kind and element <degree> are allowed in arbitrary formula. It should also be mentioned, that annotated atoms can always be understood as the implication

```
<Implies kind="zadeh">
  <head>
```
Figure 2. Content Models for Formulas in Our Uncertainty Extension

\[
I(x \supset y) = \begin{cases} 
1.0 & \text{if } x \leq y \\
0.0 & \text{otherwise}
\end{cases}
\]

The specific encoding of implication-based languages is straightforward, and will be analysed in detail in the next section.

4 Examples of Encoding Existing Languages

The simplest (at the syntactical level) rule-based languages (see examples in Section 2) depart from definite logic programming rules by adding a degree associated with the rule, these include quantitative deduction, possibilistic logic programming, ordinary probabilistic logic programs, and stochastic logic programs. These can be rendered according to the following general pattern:

\[
<\text{Implies kind="..."}>
\quad <\text{degree}>
\quad <\text{Data xsi:type="xsd:decimal">...</Data>
\quad <\text{degree}>
\quad <\text{head}><\text{Atom}>...</Atom></head>
\quad <\text{body}><\text{And}>...</And></body>
\quad </\text{Implies}>
\]

The kind attribute in element <Implies> could be used to specify the underlying semantics of the rule (e.g. "slp" for Stochastic Logic Programs\(^1\)). The degree is always a non-negative decimal number. However, except for Stochastic Logic Programs, these are all particular cases of the Fuzzy Logic Programming framework which follows the pattern:

\[
<\text{Implies kind="some-implication">}
\quad <\text{degree}>
\quad <\text{Data xsi:type="xsd:decimal">...</Data>
\quad <\text{degree}>
\quad <\text{head}><\text{Atom}>...</Atom></head>
\quad <\text{body}><\text{And kind="some-tnorm">...</And>
\quad </\text{body}>
\quad </\text{Implies}>
\]

Facts are encoded as

\[
<\text{Atom}>
\quad <\text{degree}>
\]

\(^1\)The exact identifiers of supported languages will be defined elsewhere. For instance, it seems preferable to assign to each of these semantics a URI, for specifying the several allowed forms of rules.
or, equivalently, by empty body implications:

\[
\text{Implies kind="some-implication"}
\]

\[
\text{degree}
\]

\[
\text{head}<\text{Atom}>...<\text{Atom}</head>
\]

\[
\text{body}<\text{And}</And></body>
\]

\[/Implies\]

In order to guarantee the equivalence of the above encodings, it is suggested to use implication connectives that obey to the property,

\[
I(x \rightarrow y) = 1.0 \text{ iff } I(x) \leq I(y)
\]

In particular, R-implications satisfy this property (see for instance [14] for a definition). The property guarantees the existence of a unique least model for the above programs (see [9]). Some usual R-implications are Lukasiewicz, Gödel, Goguen, and Fodor which are based on the corresponding t-norms bold intersection, minimum, product, and nilpotent minimum. To avoid syntactic overhead, we take the liberty of removing the attribute @xsi:type="xsd:decimal" from all <Data> elements in the remaining examples.

Example 1 For instance, the following ordinary probabilistic logic programming rule [24] expresses that the probability of catching a traffic jam while reaching R from S by taking a south road is at least 0.9:

\[
(\text{reach}(R, S) \mid \text{road}(R, S) \wedge \text{south}(R, S)) \ [0.9, 1.0]
\]

Under pcp-interpretations (see [24]), is equivalent to the following quantitative deduction rule

\[
\text{reach}(R, S) \leftarrow 0.9 \wedge \text{road}(R, S) \wedge \text{south}(R, S)
\]

which can be represented in our uncertainty extension as

\[
\text{Implies kind="goguen"}
\]

\[
\text{degree}
\]

\[
\text{head}
\]

\[
\text{body}
\]

This encoding is capable of capturing all forms of weight functions since it is implicitly assumed that annotated atoms...
are viewed as special forms of implication. However, f-SWRL fuzzy assertions specifying at most conditions, of the form $(a : C) \leq m$ and $(< a, b > : r) \leq m$, require the introduction of truth-value constants in the language, which is not being proposed in the current version of the language. Alternatively, the use of integrity constraints might be an interesting alternative for representing such statements.

Similar ways of encodings Probabilistic Knowledge bases, Probabilistic Logic Programs [23] can be easily defined, by simply allowing more complex formula in the head and body of rules, and using lists of two numbers to represent the associated intervals. More simple are Logic Programs with Annotated Disjunctions [35], which can be translated as shown in the example below.

**Example 3** Consider the LPAD rule, expressing that the probability of obtaining heads and tails after tossing a non-biased coin is equiprobable.

$$(\text{heads}(\text{Coin}) : 0.5) \lor (\text{tails}(\text{Coin}) : 0.5) \leftarrow \text{toss}(\text{Coin}), \neg \text{biased}(\text{Coin})$$

Notice this is a concrete extension to the Dishornlog fragment of RuleML 0.9, namely with negation as failure and probabilistic information. This is rendered in our uncertainty extension as:

```xml
<Implies kind="lpad">
  <head>
    <Or>
      <Atom>
        <degree>Data>0.5</Data></degree>
        <op><Rel>heads</Rel></op>
        <Var>Coin</Var>
      </Atom>
      <Atom>
        <degree>Data>0.5</Data></degree>
        <op><Rel>tails</Rel></op>
        <Var>Coin</Var>
      </Atom>
    </Or>
  </head>
  <body>
    <And>
      <Atom>
        <op><Rel>toss</Rel></op>
        <Var>Coin</Var>
      </Atom>
      <Naf>
        <Atom>
          <op><Rel>biased</Rel></op>
          <Var>Coin</Var>
        </Atom>
      </Naf>
    </And>
  </body>
</Implies>
```

Regarding Bayesian logic programs, the encoding is more difficult since several predicate specific parametric information should be provided in each rule. The `<degree>` element of the implication is now a conditional probability table and the combination mode used is specified in the @kind attribute in the atom element child of `<head>`. Care should be taken in order to guarantee that the same combination mode is used in all rules for that predicate.

Other probabilistic approaches like p-programs [20, 19] and normal-parametric programs [21] require a similar technique: in the `<Implies>` element we use @kind to associate the propagation function with the implication symbol or the probabilistic combination function used; the disjunction combination mode is specified in the @kind attribute in the atom element child of `<head>`; the conjunction mode is present in the `<And>` element in the body of the rule.

The annotation-based approaches are similar, and here we illustrate one of the more complex ones, namely Hybrid Probabilistic Logic Programs, which require the use of `<degree>` element with complex formulae:

**Example 4** Consider the following hp-rule

$$(\text{paper} \_\text{accepted} \lor_{pc} \text{go} \_\text{conference}): [0.85, 0.98] \leftarrow (\text{good} \_\text{work} \land \text{good} \_\text{referees}): [0.7, 0.9] \& \text{have} \_\text{money}: [0.9, 1.0]$$

The translation into our uncertainty extension is:

```xml
<Implies kind="hplp">
  <head>
    <Or kind="positive-correlation">
      <degree>Data>0.85 0.98</Data></degree>
      <op><Rel>paper_accepted</Rel></op>
      <Atom>
        <op><Rel>good_work</Rel></op>
        <Var>good_referees</Var>
      </Atom>
    </Or>
    <And>
      <And kind="independence">
        <degree>Data>0.7 0.9</Data></degree>
        <op><Rel>have_money</Rel></op>
        <Atom>
          <op><Rel>good_referees</Rel></op>
        </Atom>
      </And>
    </And>
  </head>
  <body>
    <And>
      <And kind="independence">
        <degree>Data>0.9 1.0</Data></degree>
        <op><Rel>have_money</Rel></op>
        <Atom>
          <op><Rel>good_referees</Rel></op>
        </Atom>
      </And>
    </And>
  </body>
</Implies>
```
Table 1. Encoding of uncertainty rule languages in Fuzzy RuleML

<table>
<thead>
<tr>
<th>Language</th>
<th>@kind</th>
<th>&lt;head&gt;</th>
<th>&lt;body&gt;</th>
<th>&lt;degree&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>QD [34]</td>
<td>goguen &lt;Atom&gt;</td>
<td>&lt;And kind=&quot;minimum&quot;&gt;</td>
<td>[0, 1]</td>
<td></td>
</tr>
<tr>
<td>FLP [36]</td>
<td>r-impl &lt;Atom&gt;</td>
<td>&lt;And kind=&quot;tnorm&quot;&gt;</td>
<td>[0, 1]</td>
<td></td>
</tr>
<tr>
<td>Poss [13]</td>
<td>goedel &lt;Atom&gt;</td>
<td>&lt;And kind=&quot;minimum&quot;&gt;</td>
<td>[0, 1]</td>
<td></td>
</tr>
<tr>
<td>r-SWRL [28]</td>
<td>r-impl &lt;Atom&gt;</td>
<td>&lt;And kind=&quot;tnorm&quot;&gt; of &lt;Atom&gt;&lt;degree&gt;</td>
<td>[0, 1]</td>
<td></td>
</tr>
<tr>
<td>GAP [18]</td>
<td>gap &lt;Atom&gt;&lt;degree&gt;</td>
<td>&lt;And&gt; of &lt;Atom&gt;&lt;degree&gt; (var and const. annot.)</td>
<td>none</td>
<td></td>
</tr>
<tr>
<td>SFLP [22]</td>
<td>signed (as in GAP)</td>
<td>(as in GAP)</td>
<td>sets of formulas or $2^\Delta$</td>
<td></td>
</tr>
<tr>
<td>PKB [23]</td>
<td>pkb any formula</td>
<td>any formula</td>
<td>$C[0, 1]$</td>
<td></td>
</tr>
<tr>
<td>PLP [23]</td>
<td>plp &lt;And&gt; of &lt;Atom&gt;</td>
<td>&lt;And&gt; of &lt;Atom&gt;</td>
<td>$C[0, 1]$</td>
<td></td>
</tr>
<tr>
<td>PP [19]</td>
<td>p-p &lt;Atom kind=&quot;\mu_p&quot;&gt;</td>
<td>&lt;And kind=&quot;\mu_p&quot;&gt;</td>
<td>$C[0, 1] \times C[0, 1]$</td>
<td></td>
</tr>
<tr>
<td>NPP [21]</td>
<td>f_p &lt;Atom kind=&quot;f_d&quot;&gt;</td>
<td>&lt;And kind=&quot;f_e&quot;&gt; of &lt;Atom&gt; and &lt;Neg&gt;&lt;Atom&gt;</td>
<td>$\alpha$ in lattice</td>
<td></td>
</tr>
<tr>
<td>HPLP [12]</td>
<td>hplp &lt;And kind=&quot;strat&quot;&gt; &lt;degree&gt; of formulas like in the head</td>
<td>none</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LPAD [35]</td>
<td>lpad &lt;Or&gt; of &lt;Atom&gt;&lt;degree&gt;</td>
<td>&lt;And&gt;</td>
<td>none</td>
<td></td>
</tr>
<tr>
<td>SLP [27, 6]</td>
<td>sip &lt;Atom&gt;&lt;degree&gt;</td>
<td>&lt;And&gt;</td>
<td>[0, 1]</td>
<td></td>
</tr>
<tr>
<td>BLP [17]</td>
<td>blp &lt;Atom kind=&quot;cr&quot;&gt; &lt;And&gt;</td>
<td>cpd</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The remaining annotation-based languages are treated similarly. A summary of the proposed encodings can be found in Table 1, where the syntax of $<\text{Implies}>$ is specified for some of the existing languages. Notice that for annotation-based languages the $<\text{degree}>$ element is not present, like in the previous example. There is still the need to integrate the several languages in a common algebraic framework, like Multi-adjoint, Residuated or Monotonic Logic Programming [9, 26]. However, due to their very general abstract syntax of rules, they cannot be encoded in our current proposal for the Fuzzy RuleML extension.

5 Default interpretation of connectives

To simplify the writing of uncertainty programs, a set of reasonable defaults should be proposed for the language. First, the underlying truth-value lattice is the unit interval $[0, 1]$, which is used in the majority of uncertainty rule languages. When a degree element is omitted, it is assumed to stand for the real number 1.0. It is adopted minimum t-norm and maximum s-norm as the interpretation of conjunction and disjunction, respectively. Consequently, Gödel implication is used for interpreting implication. Negation is the usual complement defined by function $1 - x$. This extends classical logic, and minimum is the less conservative t-norm. However, the distinction between strong and naf negation in this framework is not immediate (see for instance [38]) and might require more complex truth-value lattices, namely bilattices [1, 21]. The full semantics of the default language will be described in a forthcoming paper.

Furthermore, it might be possible to soften the above default setting by allowing to specify an URL of an RDF file for the defaults.

There are already query answering procedures for fuzzy logic programming based rule languages, some supporting both strong and weak negation, which can be found in [15, 7, 8, 31, 33]. It should be notice that for the default interpretation of connectives, the proof procedures in [7, 8] do terminate in polynomial time for DATALOG programs.

6 Conclusion

This paper presents a proposal for a uncertainty extension of RuleML, which is capable of encompassing a significant number of rule languages for uncertainty handling. We hope this can serve as the underpinning of the coming Fuzzy RuleML markup language. Our proposal
is a simple extension of RuleML, namely via an orthogonal use the @kind attribute and the <degree> element. The @mapKind attribute is introduced to provide a simple mechanism of expressing the desired interpretation of rules. Logical constants might be beneficial for some particular languages, namely f-SWRL, but it is not proposed for the basic language.

In the future, the language will be extended to handle quantifiers and to more general monotonic operators, in the style of [9, 26]. These are particularly important for capturing Fuzzy Description Logic Programs, like the ones recently proposed in [32]. Moreover, there is the need to provide representation mechanism for expressing parameterized fuzzy membership functions, as well as fuzzy relations, and fuzzy constants. This will be discussed in a different paper, requiring further changes to the content models presented previously.

For a final proposal, a centralized authority should be responsible for registering the several language formats and exploiting common features, as we have tried to do in this preliminary work.

Acknowledgements. We would like to thank other members (in particular Harold Boley, Giorgos Stamou, Adrian Giurca and Gerd Wagner) of the Fuzzy RuleML Technical Group (http://image.ntua.gr/FuzzyRuleML/) and David Hirtle for helpful discussion during the preparation of the paper.

References


