An Approach to Representing Uncertainty Rules in RuleML[*]

Carlos Viegas Damásio
Centro de Inteligência 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 present a list of several 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 [33], which are an extension of Horn clauses with an "attenuation factor":

$$A \leftarrow\!\boxed{q}\!- B_1 \& \ldots \& B_n, \quad n \geq 0 \tag{1}$$

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 quantitative 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

2.1.1 Fuzzy Logic Program

A typical implication-based language is Vojtáš and Paulík [36]'s fuzzy logic programming. It is a generalization of definite logic programming, where programs are constructed from an implication connective (say $\leftarrow_1$), 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:

$$A \leftarrow_1 B_1 \otimes_2 \ldots \otimes_2 B_n \text{ with-cf } q \tag{2}$$

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 [33] can be seen as a special form of fuzzy rules, where $\leftarrow_1$ 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:

$$A \leftarrow B_1 \wedge \ldots \wedge B_n \qquad (N\alpha) \tag{3}$$

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 [22] 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 [27]. f-SWRL provides OWL DL [24] axioms (but with fuzzy interpretation) as well as fuzzy rule axioms of the following form:

$$A * w \leftarrow B_1 * w_1 \wedge \ldots \wedge B_n * w_n \tag{4}$$

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. Due to some of above characteristics (among others), f-SWRL knowledge bases are not special forms of Vojtáš and Paulík [36]'s fuzzy logic programs, nor are they special forms of the annotated logic programs to be introduced below.

2.2 Annotation-based Approaches

Another kind of approach of uncertain rule languages is based on annotations which not only allow constant, but also variables and functions. Generalized Annotated Logic Programms (GAPs) is the fundamental formalism [17] in this approach. Annotated rules are of the form:

$$A : f(\mu_1, \ldots, \mu_n) \leftarrow B_1 : \mu_1 \,\&\ldots\&\, B_n : \mu_n \tag{5}$$

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 $B_1 \succeq \mu_1, \ldots, B_n \succeq \mu_n$ then $A \succeq f(\mu_1, \ldots, \mu_n)$. Annotations denote elements in such given upper-semilattice, used as underlying truth-value space, where $\succeq$ 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) [21], which theoretically has the same expressive power as GAPs [21].

2.3 Probabilistic-based Approaches

A third kind of approach of uncertain rule languages is based on probability. In this approach, the notion of probabilistic strategy is introduced because there is no single "formula" for computing the probability of a complex event $(e_1 \wedge e_2)$ 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.

2.3.1 Hybrid Probability Logic Program

A Hybrid Probabilistic Logic Program [12] over the set $\mathcal{S}$ of probabilistic-strategies is a finite set of hp-rules of the form:

$$F_0 : \mu_0 \leftarrow F_1 : \mu_1 \wedge \ldots \wedge F_k : \mu_k \tag{6}$$

where each $F_i : \mu_i$ is an hp-annotated basic formula over $\mathcal{S}$. Intuitively, an hp-rule means that "if the probability of F_1 falls in the interval μ_1 and … and the probability of F_k falls within the interval μ_k, then the probability of F_0 lies in the interval μ_0". The F_is are designated hybrid basic formulas and are either applications of conjunctive $(B^i_1 \wedge_s \ldots \wedge_s B^i_{n_i})$ or disjunctive strategies $(B^i_1 \vee_s \ldots \vee_s B^i_{n_i})$ to finite sets of distinct atoms $(B^i_1, \ldots, B^i_{n_i})$, encoding complex events. Intervals are pairs $[c_1, c_2]$ 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 [16] which consist of a (finite) set of Bayesian rules of the form:

$$A \mid B_1, \ldots, B_n, \qquad n \geq 0 \tag{7}$$

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 [18], 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, 28, 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 [29]. 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). When used the attribute `@mapKind` in the performatives, it expresses the default interpretation of the connectives inside. 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 [15]. We prefer to use the normalized striped syntax, and therefore ignoring stripe

`Assert`

attributes:	@mapDirection, @mapClosure, **@mapKind**
content:	(oid?, (formula)*)

`Query`

attributes:	@closure, **@mapKind**
content:	(oid?, (formula)*)

`Protect`

attributes:	@closure, @mapDirection, @mapClosure, **@mapKind**
content:	(oid?, (warden)+, (formula)*)

`@mapKind`

[optional]	(default:fo | lp | *list to be completed*)

Figure 1. Content Models for Performatives

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 dependant, and the reader is referred to [15] for the allowed combinations. Concrete fragments of XML markup can be 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>
    <Atom><op><Rel>prop</Rel></op></Atom>
  </head>
  <body>
    <Constant>
      <degree>
        <Data xsi:type="xsd:decimal">0.5
        </Data>
      </degree>
```

`Atom`

attributes:	@closure, **@kind**
content:	(oid)?, degree?, op, (slot)*, (arg)+, (slot)*

`Implies`

attributes:	@closure, @direction, @kind
content:	(oid?, **degree?**, ((head, body) | (body, head)))

`Integrity`

attributes:	@closure, @direction, **@kind**
content:	(oid?, **degree?**, formula)

`Equivalent`

attributes:	@closure, **@kind**
content:	(oid?, **degree?**, torso, torso)

`And, Or`

attributes:	**@kind** (@closure within Query only)
content:	(oid?, **degree?**, (formula)*)

`Naf`

attributes:	**@kind**
content:	(oid?, **degree?**, weak)

`Neg`

attributes:	**@kind**
content:	(oid?, **degree?**, strong)

`degree`

attributes:	none
content:	(Data)

`@kind`

[optional]	(default:fo | lp | *list to be completed*)

Figure 2. Content Models for Formulas in Our Uncertainty Extension

```
    </Constant>
  </body>
</Implies>
```

However, this results in complex markup that is difficult to understand and requires a new type of formula `<Constant>` which is currently absent from RuleML; therefore it is adopted the simplest syntax with `<degree>` element in atoms. This latest encoding has the advantage that variables in annotations are not required (see [10]). The implication connective used has been proposed by Zadeh [14] and is interpreted by the function

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

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:

```
<Implies kind="...">
  <degree>
    <Data xsi:type="xsd:decimal">...</Data>
  </degree>
  <head><Atom>...</Atom></head>
  <body><And>...</And></body>
</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:

```
<Implies kind="some-implication">
  <degree>
    <Data xsi:type="xsd:decimal">...</Data>
  </degree>
  <head><Atom>...</Atom></head>
  <body><And kind="some-tnorm">...</And>
</Implies>
```

Facts are encoded as

```
<Atom>
  <degree>
    <Data xsi:type="xsd:decimal">...</Data>
  </degree>
  <op><Rel>...</Rel></op>
  ...
</Atom>
```

or, equivalently, by empty bodied implications:

```
<Implies kind="some-implication">
  <degree>
    <Data xsi:type="xsd:decimal">...</Data>
```

[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.

```
  </degree>
  <head><Atom>...</Atom></head>
  <body><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 [23] expresses that the probability of catching a traffic jam while reaching R from S by taking a south road is at least 0.9:

$$(reach(R, S) \mid road(R, S) \land south(R, S)) \quad [0.9, 1.0]$$

Under pcp-interpretations (see [23]), is equivalent to the following quantitative deduction rule

$$reach(R, S) \leftarrow\!\boxed{0.9}\!- road(R, S) \;\&\; south(R, S)$$

which can be represented in our uncertainty extension as

```
<Implies kind="goguen">
  <degree>
    <Data>0.9</Data>
  </degree>
  <head>
    <Atom><op><Rel>reach</Rel></op>
          <Var>R</Var>
          <Var>S</Var>
    </Atom>
  </head>
  <body>
    <And kind="minimum">
      <Atom><op><Rel>road</Rel></op>
            <Var>R</Var>
            <Var>S</Var>
      </Atom>
      <Atom><op><Rel>south</Rel></op>
            <Var>R</Var>
            <Var>S</Var>
      </Atom>
    </And>
</Implies>
```

For possibilistic logic programming, the encoding is simpler since the implication used is Gödel's one and conjunction is the corresponding minimum t-norm.

The language f-SWRL has more cases to be taken care due to the use of weights in the body of rules. The interpretation of combination of atoms with weights in bodies can be seen as a generalization of implications with a constant (weight) in the antecedent and atom in the consequent, which is very similar to the meaning of an annotated atom. Therefore, the same syntax is used.

Example 2 Consider the following f-SWRL rule

$$\begin{aligned}&Happy(?a) * 0.7 \leftarrow \\ &\quad EyebrowsRaised(?a) * 0.9 \land MouthOpen(?a) * 0.8\end{aligned}$$

This can be encoded as follows, where Gödel's implication is used and Goguen implication is used as weight function:

```
<Implies kind="goedel">
  <degree><Data>0.7</Data></degree>
  <head>
    <Atom>
      <op><Rel>Happy</Rel></op>
      <Var>a</Var>
    </Atom>
  </head>
  <body>
    <And kind="minimum">
      <Atom kind="goguen">
        <degree><Data>0.9</Data></degree>
        <op><Rel>EyebrowsRaised</Rel></op>
        <Var>a</Var>
      </Atom>
      <Atom kind="goguen">
        <degree><Data>0.8</Data></degree>
        <op><Rel>MouthOpen</Rel></op>
        <Var>a</Var>
      </Atom>
    </And>
  </body>
</Implies>
```

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 [22] 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 [34], which can be translated as per 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.

$$(heads(Coin) : 0.5) \vee (tails(Coin) : 0.5) \\ \leftarrow toss(Coin), \neg biased(Coin)$$

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:

```
<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 [19, 18] and normal-parametric programs [20] 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

$$(paper\_accepted \vee_{pc} go\_conference) \colon [0.85, 0.98] \longleftarrow \\ (good\_work \wedge_{ind} good\_referees) : [0.7, 0.9] \ \& \\ have\_money : [0.9, 1.0]$$

The translation into our uncertainty extension is rather complex:

```
<Implies kind="hplp">
  <head>
    <Or kind="positive-correlation">
      <degree><Data>0.85 0.98</Data></degree>
      <Atom>
        <op><Rel>paper_accepted</Rel></op>
        <Var>Coin</Var>
      </Atom>
      <Atom>
        <op><Rel>go_conference</Rel></op>
        <Var>Coin</Var>
      </Atom>
    </Or>
  </head>
  <body>
    <And>
      <And kind="independence">
        <degree><Data>0.7 0.9</Data></degree>
        <Atom>
          <op><Rel>good_work</Rel></op>
          <Var>Coin</Var>
        </Atom>
        <Atom>
          <op><Rel>good_referees</Rel></op>
          <Var>Coin</Var>
        </Atom>
      </And>
      <And kind="independence">
        <degree><Data>0.9 1.0</Data></degree>
        <Atom>
          <op><Rel>have_money</Rel></op>
        </Atom>
      </And>
    </And>
  </body>
</Implies>
```

The remaining annotation-based languages are treated similarly. A summary of the proposed encodings can be found in Table 1.

Table 1. Summary of encodings of uncertainty rule languages

Language	`@kind`	`<head>`	`<body>`	`<degree>`
QD [33]	`goguen`	`<Atom>`	`<And kind="minimum">`	$[0, 1]$
FLP [35]	r-impl	`<Atom>`	`<And kind="tnorm">`	$[0, 1]$
Poss [13]	`goedel`	`<Atom>`	`<And kind="minimum">`	$[0, 1]$
f-SWRL [27]	r-impl	`<Atom>`	`<And kind="tnorm"> of <Atom><degree>`	$[0, 1]$
GAP	`gap`	`<Atom><degree>` (complex annot.)	`<And>` of `<Atom><degree>` (var and const. annot.)	none
SFLP [21]	`signed`	(as in GAP)	(as in GAP)	sets of formulas or 2^{Δ}
PKB [22]	`pkb`	any formula	any formula	$\mathcal{C}[0, 1]$
PLP [22]	`plp`	`<And>` of `<Atom>`	`<And>` of `<Atom>`	$\mathcal{C}[0, 1]$
PP [18]	`p-p`	`<Atom kind="`μ_p`">`	`<And kind="`μ_r`">`	$\mathcal{C}[0, 1] \times \mathcal{C}[0, 1]$
NPP [20]	f_p	`<Atom kind="`f_d`">`	`<And kind="`f_c`"> of` `<Atom>` and `<Neg><Atom>`	α in lattice
HPLP [12]	hplp	`<And kind="strat"> <degree> or <Or kind="strat"> <degree>` constant annotations $\mathcal{C}[0, 1] \times \mathcal{C}[0, 1]$	`<And>` of formulas like in the head constant annotations $\mathcal{C}[0, 1] \times \mathcal{C}[0, 1]$	none
LPAD [34]	`lpad`	`<Or> of <Atom><degree>` degree in $[0, 1]$	`<And>`	none
SLP [26, 6]	`slp`	`<Atom>`	`<And>`	$[0, \infty]$
BLP [16]	`blp`	`<Atom kind="cr">`	`<And>`	cpd

There is still the need to integrate these semantics in a common algebraic framework, like Multi-adjoint, Residuated or Monotonic Logic Programming [9, 25]. However, due to their very general abstract syntax of rules, they cannot be encoded in our current proposal.

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, Goedel 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 [37]) and might require more complex truth-value lattices, namely bilattices [1, 20]. 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 [7, 8, 30, 32]. It should be notice that for the default interpretation of connectives, the proof procedures 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, 25]. These are particularly important for capturing Fuzzy Description Logic Programs, like the ones recently proposed in [31]. 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

[1] J. Alcântara, C. V. Damásio, and L. M. Pereira. An encompassing framework for paraconsistent logic programs. *J. Applied Logic*, 3(1):67–95, 2005.

[2] J. F. Baldwin, T. P. Martin, and B. W. Pilsworth. *Fril - Fuzzy and Evidential Reasoning in Artificial Intelligence*. Research Studies Press Ltd, 1995.

[3] H. Boley, V. Bhavsar, and L. Yang. A weighted-tree similarity algorithm for multi-agent systems in e-business environments. In *Business Agents and the Semantic Web 2004 (In conjunction with the 2003 Canadian AI Conference)*, number 47089 in NRC, pages 53–72, 2003.

[4] J. Calmet, J. Lu, M. Rodriguez, and J. Schü. Signed formula logic programming: operational semantics and applications. In Z. W. Rás and M. Michalewicz, editors, *Proceedings of the Ninth International Symposium on Foundations of Intelligent Systems*, volume 1079 of *LNAI*, pages 202–211, Berlin, June 9–13 1996. Springer.

[5] T. Cao. Annotated fuzzy logic programs. *Fuzzy Sets and Systems*, 113(2):277–298, 2000.

[6] J. Cussens. Parameter estimation in stochastic logic programs. *Machine Learning*, 44(3):245–271, 2001.

[7] C. V. Damásio, J. Medina, and M. Ojeda-Aciego. Sorted multi-adjoint logic programs: termination results and applications. In *Proceedings of JELIA'04*, LNAI 3229, pages 260–273, 2004.

[8] C. V. Damásio, J. Medina, and M. Ojeda-Aciego. A tabulation procedure for first-order residuated logic programs: soundness, completeness and optimisations. In *Proceedings of FUZZ-IEEE'06*, 2006. To appear.

[9] C. V. Damásio and L. M. Pereira. Monotonic and residuated logic programs. In S. Benferhat and P. Besnard, editors, *Symbolic and Quantitative Approaches to Reasoning with Uncertainty, 6th European Conference, ECSQARU 2001, Toulouse, France, September 19-21, 2001, Proceedings*, volume 2143 of *Lecture Notes in Computer Science*, pages 748–759. Springer, 2001.

[10] C. V. Damásio and L. M. Pereira. Sorted monotonic logic programs and their embeddings. In *Proc. IPMU'04*, pages 807–814, 2004.

[11] A. Dekhtyar, M. I. Dekhtyar, and V. S. Subrahmanian. Temporal probabilistic logic programs. In D. D. Schreye, editor, *Logic Programming: The 1999 International Conference, Las Cruces, New Mexico, USA, November 29 - December 4*, pages 109–123, 1999.

[12] A. Dekhtyar and V. S. Subrahmanian. Hybrid probabilistic programs. *Journal of Logic Programming*, 43(3):187–250, 2000.

[13] D. Dubois, J. Lang, and H. Prade. Towards possibilistic logic programming. In K. Furukawa, editor, *Proceedings of the 8th International Conference on Logic Programming*, pages 581–598. MIT, June 1991.

[14] D. Dubois and H. Prade, editors. *Fudamentals of Fuzzy Sets*. Kluwer Academic Publishers, 2000.

[15] D. Hirtle. RuleML 0.9 content models, 2006. http://www.ruleml.org/0.9/xsd/content_models_09.pdf.

[16] K. Kersting and L. D. Raedt. Bayesian logic programs. In J. Cussens and A. M. Frisch, editors, *ILP Work-in-progress reports*, volume 35 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2000.

[17] M. Kifer and V. S. Subrahmanian. Theory of generalized annotated logic programming and its applications. *The Journal of Logic Programming*, 12(1, 2, 3 & 4):335–367, 1992.

[18] L. V. S. Lakshmanan and F. Sadri. On a theory of probabilistic deductive databases. *Theory and Practice of Logic Programming*, 1(1):5–42, Jan. 2001.

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

[20] Y. Loyer and U. Straccia. Default knowledge in logic programs with uncertainty. In *Proc. of the 19th Int. Conf. on Logic Programming (ICLP-03)*, number 2916 in Lecture Notes in Computer Science, pages 466–480, Mumbai, India, 2003. Springer Verlag.

[21] J. J. Lu. Logic programming with signs and annotations. *Journal of Logic and Computation*, 6(6):755–778, Dec. 1996.

[22] T. Lukasiewicz. Probabilistic logic programming with conditional constraints. *ACM Trans. Comput. Logic*, 2(3):289–339, 2001.

[23] T. Lukasiewicz. Probabilistic logic programming with conditional constraints. *ACM Transactions on Computational Logic*, 2(3):289–339, 2001.

[24] D. L. McGuiness and F. van Harmelen. OWL web ontology language overview, 2004. W3C Recommendation `http://www.w3.org/TR/owl-features/`.

[25] J. Medina, M. Ojeda-Aciego, and P. Vojtáš. Multi-adjoint logic programming with continuous semantics. In T. Eiter, W. Faber, and M. Truszczynski, editors, *Logic Programming and Nonmonotonic Reasoning, 6th International Conference, LPNMR 2001, Vienna, Austria, September 17-19, 2001, Proceedings*, volume 2173, pages 351–364, 2001.

[26] S. Muggleton. Stochastic logic programs. *Journal of Logic Programming*, 2001. Accepted subject to revision.

[27] J. Z. Pan, G. Stoilos, G. Stamou, V. Tzouvaras, and I. Horrocks. f-SWRL: A Fuzzy Extension of SWRL. *Journal of Data Semantic, special issue on Emergent Semantics*. To appear.

[28] M. I. Sessa. Approximate reasoning by similarity-based sld resolution. *Theoretical Computer Science*, 275:389–426, 2002.

[29] G. Stoilos, G. Stamou, V. Tzouvaras, and J. Pan. Uncertainty and ruleml rulebases:a preliminary report. In A. Adi, editor, *Proceedings of the International Conference on Rules and Rule Markup Languages for the Semantic Web*, volume 3791 of *Lecture Notes in Computer Science*, pages 199–203. Springer-Verlag, 2005.

[30] U. Straccia. Query answering in normal logic programs under uncertainty. In *8th European Conferences on Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU-05)*, number 3571 in Lecture Notes in Computer Science, pages 687–700, Barcelona, Spain, 2005. Springer Verlag.

[31] U. Straccia. Fuzzy description logic programs. In *Proceedings of the 11th International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems, (IPMU-06)*, 2006.

[32] U. Straccia. Query answering under the any-world assumption for normal logic programs. In *Proceedings of the 10th International Conference on Knowledge Representation (KR-06)*, 2006.

[33] M. H. van Emden. Quantitative deduction and its fixpoint theory. *The Journal of Logic Programming*, 3(1):37–54, Apr. 1986.

[34] J. Vennekens, S. Verbaeten, and M. Bruynooghe. Logic programs with annotated disjunctions. In B. Demoen and V. Lifschitz, editors, *Proc. of ICLP 2004*, volume 3132 of *Lecture Notes in Computer Science*, pages 431–445. Springer, 2004.

[35] P. Vojtáš. Fuzzy logic programming. *Fuzzy sets and systems*, 124(3):361–370, 2001.

[36] P. Vojtáš and L. Paulík. Soundness and completeness of non-classical sld-resolution. In R. Dyckhoff, H. Herre, and P. Schroeder-Heister, editors, *Extensions of Logic Programming, 5th International Workshop, ELP'96, Leipzig, Germany, March 28-30, 1996, Proceedings*, volume 1050 of *Lecture Notes in Computer Science*, pages 289–301. Springer, 1996.

[37] G. Wagner. A logical reconstruction of fuzzy inference in databases and logic programs. In *Proc. of 7th Int. Fuzzy Systems Association World Congress (IFSA'97)*, 1997.