Fundamenta Informaticae 82 (2008) 1–24
IOS Press

Representing Uncertainty 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 and vagueness. 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. It is detailed the important case of fuzzy rule languages. 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

[*]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). This is a revised and extended version of the paper titled: "An Approach to Representing Uncertainty Rules in RuleML", which was published in the 2nd International Conference of Rules and Rule Markup Languages for the Semantic Web (RuleML-2006).
[†]Address for correspondence: Departamento de Informtica, Faculdade de Cincias e Tecnologia, Universidade Nova de Lisboa, Quinta da Torre, 2829-516 Caparica, Portugal

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[1] 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 normalised 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. For this purpose, RuleML sets up the Fuzzy RuleML Technical Group to tackle this issue. In this paper, we intend to provide a general framework of uncertainty rules, which hopefully will serve as the underpinning of the coming Fuzzy RuleML language.

The need of representing uncertainty information in rules has been identified in various practical applications. For instance, in order to enable the use emotion in photo search in the Web, such as searching for photos of "happy people", uncertainty is needed in representing the relationships between human emotions and facial expression in image analysis about emotion recognition. We can capture such relationships by rules (such as the following), but not those without uncertainty support (e.g., to some degree some assertion is true):

- If someone has her eyebrows raised with a degree larger than 0.4 and has her mouth stretched with a degree larger than 0.8, then she is happy with a degree larger than 0.7.

- If someone has her eyebrows raised with a degree larger than 0.7 and has her mouth open with a degree larger than 0.9, then she is surprised with a degree larger than 0.7.

- If someone has her eyebrows squeezed with a degree larger than 0.8 and has her teeth visible with a degree larger than 0.7, then she is angry with a degree larger than 0.7.

Using the above rules together with the following assertions: (1) Mary has her eyebrows raised with a degree larger than 0.8, (2) Mary has her mouth stretched with a degree larger than 0.9, (3) Mary has her mouth open with a degree larger than 0.6, we can draw the conclusion that Mary is more happy than surprised.

The main contributions of the paper are two-folded. (1) On the one hand, it proposes 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

[1]Here we are using the term "uncertainty" in the wide sense, covering probability theory, possibility theory, fuzzy logic, and ignorance.

provide some insights on how to cover uncertainty in the RIF framework. (2) On the other hand, it investigate how to integrate such a framework in the ongoing Rule Interchange Format (RIF). In particular, we will focus on how to accommodate different uncertainty semantics in RIF and how to extend the RIF Core language to support uncertainties.

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 extension is explored in Section 4 to illustrate the encoding of several of the languages previously described. Section 5 discusses default interpretation of connectives for the fuzzy extension of RuleML, as a particular example of the proposed uncertainty extension. The proposal of a RIF dialect for the language is proposed in Section 6. Section 7 concludes the paper by briefly analysing the encodings and summarising the alternatives.

2. Illustrative uncertainty rule languages

The literature in Logic Programming contains a spate of semantics and languages for handling uncertainty; we designate them uncertainty rule languages (including probabilistic, fuzzy, and possibilistic ones). One of the first well known related languages is Van Emden's quantitative deduction rules [21], which are an extension of Horn clauses with a numeric "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 ordinary first-order atoms A and $B_1 \ldots B_n$, and the attenuation factor q, a real number in the interval $(0, 1]$. Truth-values of ground atoms, i.e. without variables, are interpreted in the closed unit interval $[0, 1]$. A ground quantitative deduction rule is satisfied by 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, fuzzy and many-valued, possibilistic-based and probabilistic-based approaches) uncertainty rule languages, in order to illustrate the existing diversity of proposals and syntactical constructs. We will neglect, as possible, the important semantic aspects, which can be rather involved. 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. Fuzzy and many-valued approaches

Fuzzy and many-valued 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 programs

A typical language is Vojtáš and Paulík [48]'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 [21] 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 [13], in continuation of the initial work of Vojtáš and Paulík.

2.1.2. f-SWRL

The more recently proposed fuzzy Semantic Web rule language (f-SWRL) language [38] provides OWL DL [35] axioms (but with fuzzy interpretation) as well as fuzzy rule axioms of the following form:

$$A * w \leftarrow B_1 * w_1 \wedge \ldots \wedge B_n * w_n \tag{3}$$

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. The f-SWRL language 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 concepts and role axioms from fuzzy OWL, they are not special forms of Vojtáš and Paulík [48]'s fuzzy logic programs, nor are they special forms of the annotated logic programs to be introduced below.

A more general framework has been proposed by Thomas Lukasiewicz in [34] which allows for the integration of ontologies with rule bases, allowing also default negation. Syntactically, fuzzy dl-programs allow the combination of arbitrary t-norms in the body with arbitrary negation-like operators. Furthermore, DL-atoms can be used in the bodies of fuzzy dl-rules to query ordinary description logic knowledge bases.

2.1.3. Annotated logic programs

In this family of languages, rules keep a classical interpretation while uncertainty is associated with atoms, not with the implication. Generalized annotated logic programs (GAPs) is the fundamental formalism [26] in this approach. Annotated rules are of the form:

$$A : f(\mu_1, \ldots, \mu_n) \leftarrow B_1 : \mu_1 \,\&\, \ldots \,\&\, B_n : \mu_n \tag{4}$$

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 $B_1 \succeq \mu_1$, ..., $B_n \succeq \mu_n$ 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) [30], which theoretically has the same expressive power as GAPs [30].

2.2. Possibilistic-based approaches

Possibilistic logic is a logic of uncertainty designed for reasoning under incomplete evidence and partially inconsistent knowledge [18]. The logic is non truth-functional, and its integration with logic programming has been proposed firstly in [17] have been proposed. A *possibilistic logic program* is a finite set of (first-order) possibilistic Horn clauses annotated only with necessity degrees of the following form:

$$A \leftarrow B_1 \wedge \ldots \wedge B_n \qquad (N\alpha) \tag{5}$$

where A, $B_1, \ldots, B_n$ are propositional symbols and $(N\alpha)$ is the necessity degree of the clause with $0 \leq \alpha \leq 1$. The intended minimal model for possibilistic logic programming [17] can be captured by a straightforward translation into quantitative deduction rules, where the underlying implication and conjunctor are the corresponding Gödel's connectives.

Possibilistic logic has been extended with fuzzy constants and fuzzy quantifiers [20] and Horn fragments have been studied in the literature, namely in [2, 3, 4]. However the use of implication and conjunction is not equivalent to the use of disjunction and negation, and both variants are described in the literature. Syntactically, rules or clauses have attached a valuation function (see [2] for more details).

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. Probabilistic logic programs

Probabilistic knowledge bases of Lukasiewicz [31] are sets of conditional constraints of the form:

$$(H \mid B)[c_1, c_2] \tag{6}$$

where H and B are ordinary first-order formula, and $c_1 \leq c_2$ are rational numbers in the interval $[0, 1]$. Probabilistic logic programs of are sets of conditional constraints where H is restricted to a conjunction of atoms and B is either a conjunction of atoms or $\top$. These conditional constraints express that the conditional probability of H given B is between c_1 and c_2 or that the probability of the antecedent is 0. A semantics and complexity of reasoning are exhaustively studied, and in most cases is intractable and not truth-functional. However, for a special kind of probabilistic logic programs the author provides relationships to "classical" logic programming. Ordinary probabilistic logic programs are probabilistic logic programs where the conditional constraints have the restricted form

$$(A \mid B_1 \wedge \ldots \wedge B_n)[c, 1] \text{ or } (A \mid \top)[c, 1] \tag{7}$$

Under positively correlated probabilistic interpretations (pcp-interpretations), reasoning becomes tractable and truth-functional, and can be embedded into quantitative deduction rules. This framework has been further explored by integrating probabilistic reasoning with description logics [33].

A theory of probabilistic deductive databases is described in Lakshmanan and Sadri's work [27] where belief and doubt can both be expressed explicitly with equal status. Probabilistic programs (p-programs) are finite sets of triples of the form:

$$\left(A \overset{c}{\leftarrow} B_1, \ldots, B_n; \mu_r, \mu_p\right) \tag{8}$$

As usual, A, B_1, …, B_n are atoms, which may not contain complex terms, c is a confidence level, and μ_r (μ_p) is the conjunctive (disjunctive) mode associated with the rule. For a given ground atom A, the disjunctive mode associated with all the rules for A must be the same. Confidence levels are pairs of intervals $\langle[\alpha, \beta]\rangle[\gamma, \delta]$, the first component denoting the belief and the second doubt. This terminology is borrowed from Fitting's bilattices [22]. This was further generalized by Loyer and Straccia [29] within the framework of normal parametric programs

$$A \overset{\alpha}{\leftarrow} L_1, \ldots, L_n; \langle f_d, f_p, f_c\rangle \tag{9}$$

The function f_d is the disjunction function used to combine information from the several rules for predicate A, and f_p and f_c are the propagation function and the conjunction function. The conjunction function is used to combine the information from literals in the body of the rule $L_1, \ldots, L_n$, while f_p is used to combine this result with the certainty of the rule α, taken from a lattice. The truth-value of atoms are intervals in some lattice. The body can also contain truth-values in the lattice, besides literals.

2.3.2. Hybrid probabilistic logic programs

Hybrid probabilistic logic programs [16] 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 ($e_1 \wedge e_2$) where e_1 and e_2 are primitive events [16]. We refer the reader to [16] for more details about probabilistic strategies.

A hybrid probabilistic logic program over the set $\mathcal{S}$ of probabilistic-strategies is a finite set of hp-rules of the form:

$$F_0 : \mu_0 \leftarrow F_1 : \mu_1 \wedge \ldots \wedge F_k : \mu_k \tag{10}$$

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 ($B_1^i \wedge_s \ldots \wedge_s B_{n_i}^i$) or disjunctive strategies ($B_1^i \vee_s \ldots \vee_s B_{n_i}^i$) to finite sets of distinct atoms ($B_1^i, \ldots, B_{n_i}^i$), 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 [15], in particular annotations are more complex since they contain a time dimension.

2.3.3. Logic programs with annotated disjunctions

Stochastic logic programs are a generalization of Hidden Markov Models [37, 10] and are constituted by range restricted Horn clauses labeled with non-negative numbers:

$$p : A :- B_1, \ldots, B_n \tag{11}$$

The original framework by Muggleton [37] enforced that the numbers are probability labels in the unit interval, which should sum to 1 for every predicate symbol defined in the program. Stochastic logic programs have a distributional semantics based on SLD-derivations, assigning a probability distribution to each predicate symbol. The dependency on a particular proof mechanism has been criticized in [46], mostly because of its non-declarative character. Logic programs with annotated disjunctions (LPADs) are more declarative and are formed by rules of the form:

$$(A_1 : p_1) \vee \ldots (A_m : p_m) \leftarrow L_1 \ \ldots, L_n \tag{12}$$

where each A_i is an atom, $\sum_{1 \leq i \leq n} p_i = 1$ and $L_1, \ldots, L_n$ are literals (atoms or their default negation). Each p_i is a probability in the unit interval, and the semantics defines again a probability distribution on Herbrand interpretations. Logic programs with annotated disjunctions generalise Poole's independent choice logic programs [39].

2.3.4. Bayesian logic programs

Finally, we consider Bayesian logic programs [25] which consist of a (finite) set of Bayesian rules of the form:

$$A \mid B_1, \ldots, B_n, \qquad n \geq 0 \tag{13}$$

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 can be embedded into Bayesian Logic Programs.

3. A general uncertainty extension for RuleML

In this section, we propose an uncertainty 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 [6]. This relevance measure is used in [6] 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 RuleML framework in particular to represent and reason with fuzzy data. Some proposals already support these notions, like the ones described in [5, 40, 9].

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 [41]. 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 [24]. We prefer

`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

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 dependant, and the reader is referred to [24] 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. The annotation might also be a variable or a complex annotation (only in head of rules). Similarly, signed formula logic programming [8] 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>
```

`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

```
        <Data xsi:type="xsd:decimal">0.5
        </Data>
      </degree>
    </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 [14]). The implication connective used has been proposed by Zadeh [19] 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[2]). 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>
  </body>
</Implies>
```

Facts are encoded as

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

or, equivalently, by empty body implications:

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

[2]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.

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 [19] for a definition). The property guarantees the existence of a unique least model for the above programs (see [13]). 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 4.1. For instance, the following ordinary probabilistic logic programming rule [31] expresses that the probability of catching a traffic jam while reaching R from S by taking a south road is at least 0.9:

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

Under pcp-interpretations (see [31]), 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 4.2. Consider the following f-SWRL rule

$$\begin{aligned}&Happy(?a) * 0.7 \leftarrow \\ &\quad EyebrowsRaised(?a) * 0.4 \wedge MouthOpen(?a) * 0.8\end{aligned}$$

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>0.4</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. Regarding, fuzzy description logic programs [34] the encoding requires the introducing of a special form of atoms, DL-atoms, which is not tackled by the previous syntax, requiring additional investigation.

Similar ways of encodings probabilistic knowledge bases, probabilistic logic programs [32] 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 [46], which can be translated as shown in the example below.

Example 4.3. Consider the LPAD rule, expressing that the probability of obtaining heads and tails after tossing a non-biased coin is equiprobable.

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

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 [28, 27] and normal-parametric programs [29] 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 4.4. Consider the following hp-rule

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

The translation into our uncertainty extension is:

```
<Implies kind="hplp">
 <head>
  <Or kind="positive-correlation">
   <degree><Data>0.85 0.98</Data></degree>
   <Atom><op><Rel>paper_accepted</Rel></op>
   </Atom>
   <Atom><op><Rel>go_conference</Rel></op>
   </Atom>
  </Or>
 </head>
 <body>
  <And>
   <And kind="independence">
    <degree><Data>0.7 0.9</Data></degree>
    <Atom><op><Rel>good_work</Rel></op>
    </Atom>
    <Atom><op><Rel>good_referees</Rel></op>
    </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, where the syntax of `<Implies>` is specified for some of the existing languages. Notice that for annotation-based languages the `<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 [13, 36]. However, due to their very general abstract syntax of rules, they cannot be encoded in our current proposal for the RuleML extension.

Table 1. Encoding of uncertainty rule languages in RuleML

Language	`@kind`	`<head>`	`<body>`	`<degree>`
QD [21]	`goguen`	`<Atom>`	`<And kind="minimum">`	$[0,1]$
FLP [47]	r-impl	`<Atom>`	`<And kind="tnorm">`	$[0,1]$
Poss [17]	`goedel`	`<Atom>`	`<And kind="minimum">`	$[0,1]$
f-SWRL [38]	r-impl	`<Atom>`	`<And kind="tnorm">` of `<Atom><degree>`	$[0,1]$
GAP [26]	`gap`	`<Atom><degree>` (complex annot.)	`<And>` of `<Atom><degree>` (var and const. annot.)	none
SFLP [30]	`signed`	(as in GAP)	(as in GAP)	sets of formulas or 2^{Δ}
PKB [32]	`pkb`	any formula	any formula	$\mathcal{C}[0,1]$
PLP [32]	`plp`	`<And>` of `<Atom>`	`<And>` of `<Atom>`	$\mathcal{C}[0,1]$
PP [27]	`p-p`	`<Atom kind="`μ_p`">`	`<And kind="`μ_r`">`	$\mathcal{C}[0,1] \times \mathcal{C}[0,1]$
NPP [29]	f_p	`<Atom kind="`f_d`">`	`<And kind="`f_c`"> of <Atom>` and `<Neg><Atom>`	α in lattice
HPLP [16]	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 [46]	`lpad`	`<Or> of <Atom><degree>` degree in $[0,1]$	`<And>`	none
SLP [37, 10]	`slp`	`<Atom>`	`<And>`	$[0,\infty]$
BLP [25]	`blp`	`<Atom kind="cr">`	`<And>`	cpd

5. Fuzzy RuleML

So far we have introduced the optional `<degree>` element and the `@kind` attribute into RuleML. The `<degree>` element can be attached to atoms (or rules) to specify confidence degrees, truth-values, probabilities, weights, annotations, etc. The `@kind` attribute can be used to indicate the semantics of connectives. In order to make our uncertainty extension compatible with RuleML, and more importantly, to simplify the writing of uncertainty programs, some default settings should be provided for the degree elements and the semantics of the connectives. Due to its simplicity and ease of implementation, it is adopted a fuzzy interpretation of the connectives in the spirit of the languages described in Section 2.1.1.

- 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.

- The default semantics for conjunction is minimum t-norm.
- The default semantics for disjunction is maximum s-norm.
- The default semantics for negation is $1 - x$.
- Consequently, Gödel implication is used by default for interpreting implication, where

$$x \rightarrow_G y = \begin{cases} 1 \text{ if } x \leq y \\ y \text{ otherwise} \end{cases}$$

  This means an imply element `<Implies kind="goedel">` can be simplified as `<Implies>`.

There are some remarks here. Firstly, what we are trying to propose here is a reasonable default setting, rather than the *best* default setting (which might be impossible to achieve, since different people have different preferences). Like f-SWRL, we regard it an important feature that the uncertain programs will be used together with some ontologies. It has been pointed out [42] that, under the above semantics of the connectives, ontology consistency checking of fuzzy ontologies can be reduced to that of classic (Description Logic-based) ontologies. This indicates classic Description Logic reasoners can be reused to provide reasoning support for fuzzy ontologies. On the other hand, 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 [23, 11, 12, 43, 45]. It should be noticed that for the default interpretation of connectives, the proof procedures in [11, 12] do terminate in polynomial time in the data complexity for DATALOG programs.

Secondly, the selection of default connectives also is compatible with classical logic programming, and the use of the minimum t-norm guarantees that the resulting intended model of the program will be the greatest, amongst all t-norms (minimum is the less conservative t-norm).

Also, the distinction between strong and naf negation in this framework is not immediate (see for instance [49]) and might require more complex truth-value lattices, namely bilattices [1, 29]. Further discussions on this will be part of our future work.

Last but not the least, if the default setting is not the preferred one for some applications, the users should be able to overwrite the default setting for their uncertain programs. For instance, they could specify an URL of an RDF file, which uses some pre-defined properties to specify the defaults.

6. Uncertainty dialect for RIF

In this section, we investigate how to integrate the above uncertainty framework into the W3C Rule Interchange Format (RIF). Although RIF is still a moving target at the moment, the RIF Working Group has come up with a preliminary version of the core language [7], which is used as the base for our uncertainty extension.

The RIF Core language is essentially Horn rules with equality, with its abstract syntax specified in `asn06` and has a concrete human-readable syntax, described in EBNF. The abstract syntax can be found in Figure 3, corresponding to lines not in bold format. Our extension is depicted in the same figure, where newly introduced features appear in bold type. The only non-standard feature of the presentation, is that class `ATOMIC` has properties `kind` and `degree` which are inherited by subclasses `Equal` and `Uniterm`.

```
class Ruleset
  property formula : list of RULE

class RULE
  subclass Forall
    property declare : list of Var
    property formula : CLAUSE

class CONDITION
  subclass And
    property formula : list of CONDITION
    property kind: xsd:anyURI
    property degree: Uniterm
  subclass Or
    property formula : list of CONDITION
    property kind: xsd:anyURI
    property degree: Uniterm
  subclass Exists
    property declare : list of Var
    property formula : Uniterm
    property kind: xsd:anyURI
    property degree: Uniterm
  subclass ATOMIC

class CLAUSE
  subclass ATOMIC
  subclass Implies
    property if: CONDITION
    property then: CONDITION
    property kind: xsd:anyURI
    property degree: Uniterm

class ATOMIC
  subclass Equal
    property side: list of TERM
  subclass Uniterm
  property kind: xsd:anyURI
  property degree: Uniterm

class TERM
  subclass Var
    property name: xsd:string
  subclass Const
    property name: xsd:string
  subclass Uniterm
    property op: Const
    property arg: list of TERM
```

Figure 3. Uncertainty RIF dialect abstract syntax in `asn06` notation

We are using this feature to denote that these properties only make sense in the scope of `ATOMIC` class, not being present for general `Uniterm` and `Equal` instances. The concrete syntax of RIF Core is not yet finished by the W3C Working Group, but we will use the existing one for presenting our proposal. We do not address the multisorted RIF Logic, even though such a feature will be of utmost importance for a complete uncertainty RIF dialect.

Ruleset ::= **RULE***
RULE ::= `'Forall'` **Var*** `'('` **CLAUSE** `')'`
CLAUSE ::= **Implies** | **ATOMIC**
Implies ::= **CONDITION** `':-'` **CONDITION**
CONDITION ::= **CONJUNCTION** | **DISJUNCTION** | **EXISTENTIAL** | **ATOMIC**
CONJUNCTION ::= `'And'` `'('` **CONDIT*** `')'`

DISJUNCTION ::= ′`Or`′ ′`(`′ **CONDIT*** ′`)`′
EXISTENTIAL ::= ′`Exists`′ **Var**+ ′`(`′ **CONDITION** ′`)`′
ATOMIC ::= **Uniterm** | **Equal**
Uniterm ::= **Const** ′`(`′ **TERM*** ′`)`′
Equal ::= **TERM** ′`=`′ **TERM**
TERM ::= **Const** | **Var** | **Uniterm**
Const ::= **CONSTNAME** | ′`"`′ **CONSTNAME** ′`"`′ ′`^^`′ **SORTNAME**
Var ::= ′`?`′ **VARNAME** | ′`?`′ **VARNAME** ′`^^`′ **SORTNAME**

The non-instantiable classes appear with names fully capitalized. In particular, **ATOMIC** stands for an atomic formula, which can be either an universal term (**Uniterm**) or an equality of two **TERM**s. We have generalised the RIF Core language, by allowing to appear RIF **CONDITION**s in the head (then-part) and body of a rule (if-part). RIF Core restricts heads of rules to be **ATOMIC**.

In an uncertainty dialect of the RIF language, one can introduce degrees to a rule or an atom in a rule in the following modified human-readable syntax:

Implies ::= **CONDITION** ′`:-`′ [**DEGREE** ′`-`′] **CONDITION**
ATOMIC ::= **Uniterm** [′`:`′ **DEGREE**] | **Equal** [′`:`′ **DEGREE**]
DEGREE ::= **Uniterm**

Note that the degrees are optional, and for the sake of the generality they are assumed to be arbitrary Uniterms; of course, it is expected that most of the times these will be constants of sort `xsd:decimal` or `xsd:double`. For example, we can express the uncertainty rule ‘the probability of catching a traffic jam while reaching R from S by taking a south road is at least 0.9’ from Example 1 as follows:

```
Forall ?R ?S (
  reach(?R ?S) :- 0.9 -
       And( road(?R ?S)
            south(?R ?S) )
)
```

In the above rule, the degree 0.9 is set on the implication, while in the following rule the degrees are set on the atoms. The uncertainty rule ‘If someone has her eyebrows raised with a degree larger than 0.4 and has her mouth stretched with a degree larger than 0.8, then she is happy with a degree larger than 0.7.’ from Example 2 can be expressed as follows:

```
Forall ?a (
  Happy(?a) :  0.7 :-
       And( EyebrowsRaised(?a) :  0.4
            MouthOpen(?a) :  0.8 )
)
```

Besides the introduction of degrees, one can also introduce the kind attribute to specify the kind of semantics for implications, conjunctions, disjunctions, and existential quantifications in the body. The atoms may also have an attached kind information, for supporting several languages, as described in Section 4. The equality atom may also use kind information for specifying, for instance, similarity fuzzy relations. The full syntax of the uncertainty in concrete human readable syntax is:

Implies ::= **CONDITION** ′`:-`′ [**KIND**] [**DEGREE** ′`-`′] **CONDITION**
CONJUNCTION ::= ′`And`′ [**KIND**] ′`(`′ **CONDIT*** ′`)`′ [′`:`′ **DEGREE**]
DISJUNCTION ::= ′`Or`′ [**KIND**] ′`(`′ **CONDIT*** ′`)`′ [′`:`′ **DEGREE**]
EXISTENTIAL ::= ′`Exists`′ [**KIND**] **Var**+ ′`(`′ **CONDITION** ′`)`′ [′`:`′ **DEGREE**]
ATOMIC ::= **Uniterm** [**KIND**] [′`:`′ **DEGREE**] | **Equal** [′`:`′ **DEGREE**]
Equal ::= **TERM** ′`=`′ [**KIND**] **TERM**
KIND ::= ′`<`′ `xsd:anyURI` ′`>`′
DEGREE ::= **Uniterm**

The kind should be a xsd:anyURI for identifying the corresponding non-default behaviour of connective.

Using this extension, the uncertainty rule ‘the probability of obtaining heads and tails after tossing a coin is equiprobable’ can be expressed as follows:

```
Forall ?Coin (
  Or(Heads(?Coin):0.5 Tails(?Coin):0.5) :- <lpad> Toss(?Coin)
)
```

The single sorted semantics of RIF is provided by semantic structures (or interpretations) of the form $< \mathbf{D}, \mathbf{I}_C, \mathbf{I}_V, \mathbf{I}_F, \mathbf{I}_R >$ where $\mathbf{D}$ is the domain, and four mappings:

- $\mathbf{I}_C$ mapping constants to elements of the domain $\mathbf{D}$.
- $\mathbf{I}_V$ mapping variables to elements of the domain $\mathbf{D}$.
- $\mathbf{I}_F$ mapping constants to functions from $\mathbf{D}^*$ to $\mathbf{D}$, for interpreting functor symbols.
- $\mathbf{I}_R$ mapping constants to truth-valued mappings from $\mathbf{D}^* \rightarrow \mathbf{TV}$, for interpreting predicate symbols.

¿From these, it is defined the general mapping $\mathbf{I}$ for interpreting Uniterms, as usual:

- $\mathbf{I}(k) = \mathbf{I}_C(k)$ if k is a constant symbol
- $\mathbf{I}(?v) = \mathbf{I}_V(?v)$ if $?v$ is a variable
- $\mathbf{I}(f(t_1 \ldots t_n)) = \mathbf{I}_F(f)(\mathbf{I}(t_1), \ldots, \mathbf{I}(t_n))$

The truth-value space $\mathbf{TV}$ in our uncertainty RIF dialect is the unit interval $[0, 1]$, and as expected, an omitted degree is interpreted as 1.0. This truth-value space is endowed with a truth-order corresponding to the usual total order in the real numbers, and this is denoted by $x \leq_t y$ iff x is less than or equal to y. Regarding default interpretations of connectives, the definitions presented in [7] are extended below to cater for degrees. In the following, it is assumed that deg is a Uniterm such that $\mathbf{I}(deg)$ is mapped into an element of the real unit interval, and $\rightarrow_G$ is Gödel's implication. We also assume that $\mathbf{I}(deg) > 0.0$, otherwise the conditions annotated with a degree are trivially mapped into 1.0.

- Atomic Formulas: $\mathbf{I}_{Truth}(r(t_1 \ldots t_n) : deg) = \mathbf{I}(deg) \rightarrow_G \mathbf{I}_R(r)(\mathbf{I}(t_1) \ldots \mathbf{I}(t_n))$
- Equality: $\mathbf{I}_{Truth}(t_1 = t_2 : deg) = 1.0$ iff $\mathbf{I}(t_1) = \mathbf{I}(t_2)$, otherwise $\mathbf{I}_{Truth}(t_1 = t_2 : deg) = 0.0$.
- Conjunction: $\mathbf{I}_{Truth}(And(c_1 \ldots c_n) : deg) = \mathbf{I}(deg) \rightarrow_G min_t(\mathbf{I}_{Truth}(c_1), \ldots, \mathbf{I}_{Truth}(c_n))$, where min_t is minimum with respect to the truth order.
- Disjunction: $\mathbf{I}_{Truth}(Or(c_1 \ldots c_n) : deg) = \mathbf{I}(deg) \rightarrow_G max_t(\mathbf{I}_{Truth}(c_1), \ldots, \mathbf{I}_{Truth}(c_n))$, where max_t is maximum with respect to the truth order.
- Quantification: $\mathbf{I}_{Truth}(Exists ?v_1 \ldots ?v_n(c) : deg) = \mathbf{I}(deg) \rightarrow_G lub_t(\mathbf{I}^*_{Truth}(c))$, where lub_t is taken over all interpretations $\mathbf{I}^*$ of the form $< \mathbf{D}, \mathbf{I}_C, \mathbf{I}^*_V, \mathbf{I}_F, \mathbf{I}_R >$, where $\mathbf{I}^*_V$ is the same as $\mathbf{I}_V$ except possibly on the variables $?v_1, \ldots, ?v_n$ (i.e., $\mathbf{I}^*$ agrees with $\mathbf{I}$ everywhere except possibly in its interpretation of the variables $?v_1 \ldots ?v_n$.

Rule satisfaction is simply defined (equivalently) by:

$$\begin{array}{ll} & \mathbf{I} \models then :- deg \; - \; if \\ \text{iff} & (\mathbf{I}_{Truth}(if) \rightarrow_G \mathbf{I}_{Truth}(then)) \geq_t \mathbf{I}(deg) \\ \text{iff} & \mathbf{I}_{Truth}(then) \geq_t min_t(\mathbf{I}(deg), \mathbf{I}_{Truth}(if)) \end{array}$$

This defines a minimal extension of the RIF Core to handle uncertainty. Notice that with this default interpretation, there is no need to define the kind of each connective. The previous proposal can be immediately extended in order to handle other types of connectives and sorts.

7. 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. Furthermore, we have also shown how to integrate such a framework in the preliminary version of the RIF Core language.

In the future, the language will be extended to handle quantifiers and to more general monotonic operators, in the style of [13, 36]. These are particularly important for capturing Fuzzy Description Logic Programs, like the ones recently proposed in [44, 34]. 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 centralised 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 an early version of the paper. The reviewers are also thanked for the insightful comments.

References

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

[2] Alsinet, T., Godo, L.: A Complete Calcultis for Possibilistic Logic Programming with Fuzzy Propositional Variables., *UAI '00: Proceedings of the 16th Conference in Uncertainty in Artificial Intelligence, Stanford University, Stanford, California, USA, June 30 - July 3, 2000* (C. Boutilier, M. Goldszmidt, Eds.), Morgan Kaufmann, 2000.

[3] Alsinet, T., Godo, L.: Towards an automated deduction system for first-order possibilistic logic programming with fuzzy constants., *Int. J. Intell. Syst.*, **17**(9), 2002, 887–924.

[4] Alsinet, T., Godo, L.: Adding similarity-based reasoning capabilities to a Horn fragment of possibilistic logic with fuzzy constants., *Fuzzy Sets and Systems*, **144**(1), 2004, 43–65.

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

[6] Boley, H., Bhavsar, V., Yang, L.: A Weighted-Tree Similarity Algorithm for Multi-Agent Systems in E-Business Environments, *Business Agents and the Semantic Web 2004 (In conjunction with the 2003 Canadian AI Conference)*, number 47089 in NRC, 2003.

[7] Boley, H., Kifer, M.: RIF Core Design. W3C Working Draft 30 March 2007, Available at `http://www.w3.org/TR/2007/WD-rif-core-20070330`.

[8] Calmet, J., Lu, J., Rodriguez, M., Schü, J.: Signed formula logic programming: operational semantics and applications, *Proceedings of the Ninth International Symposium on Foundations of Intelligent Systems* (Z. W. Rás, M. Michalewicz, Eds.), 1079, Springer, Berlin, June 9–13 1996, ISBN 3-540-61286-6.

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

[10] Cussens, J.: Parameter Estimation in Stochastic Logic Programs., *Machine Learning*, **44**(3), 2001, 245–271.

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

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

[13] Damásio, C. V., Pereira, L. M.: Monotonic and residuated logic programs, *Symbolic and Quantitative Approaches to Reasoning with Uncertainty, 6th European Conference, ECSQARU 2001, Toulouse, France, September 19-21, 2001, Proceedings* (S. Benferhat, P. Besnard, Eds.), 2143, Springer, 2001, ISBN 3-540-42464-4.

[14] Damásio, C. V., Pereira, L. M.: Sorted monotonic logic programs and their embeddings, *Proc. IPMU'04*, 2004.

[15] Dekhtyar, A., Dekhtyar, M. I., Subrahmanian, V. S.: Temporal Probabilistic Logic Programs, *Logic Programming: The 1999 International Conference, Las Cruces, New Mexico, USA, November 29 - December 4* (D. D. Schreye, Ed.), 1999, ISBN 0-262-54104-1.

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

[17] Dubois, D., Lang, J., Prade, H.: Towards Possibilistic Logic Programming, *Proceedings of the 8th International Conference on Logic Programming* (K. Furukawa, Ed.), MIT, June 1991, ISBN 3-540-55038-0.

[18] Dubois, D., Lang, J., Prade, H.: Possibilistic Logic, in: *Handbook of Logic in Artificial Intelligence and Logic Programming, Volume 3: Nonmonotonic Reasoning and Uncertain Reasoning* (D. Gabbay, C. J. Hogger, J. A. Robinson, Eds.), Oxford University Press, Oxford, 1994, 439–513.

[19] Dubois, D., Prade, H., Eds.: *Fudamentals of Fuzzy Sets*, Kluwer Academic Publishers, 2000.

[20] Dubois, D., Prade, H., Sandri, S.: Possibilistic logic with fuzzy constants and fuzzy quantifiers, *Logic Programming and Soft Computing* (F. Arcelli, T. Martin, Eds.), Wiley, 1998.

[21] van Emden, M. H.: Quantitative Deduction and its Fixpoint Theory, *The Journal of Logic Programming*, **3**(1), April 1986, 37–54.

[22] Fitting, M.: Bilattices and the Semantics of Logic Programs, *Journal of Logic Programming*, **11**, 1991, 91–116.

[23] Guadarrama, S., Munoz-Hernandez, S., Vaucheret, C.: Fuzzy Prolog: a new approach using soft constraints propagation, *Fuzzy Sets and Systems*, **1**(144), 2004, 127–150.

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

[25] Kersting, K., Raedt, L. D.: Bayesian Logic Programs., *ILP Work-in-progress reports* (J. Cussens, A. M. Frisch, Eds.), 35, CEUR-WS.org, 2000.

[26] Kifer, M., Subrahmanian, V. S.: Theory of Generalized Annotated Logic Programming and its Applications, *The Journal of Logic Programming*, **12**(1, 2, 3 & 4), 1992, 335–367.

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

[28] Lakshmanan, L. V. S., Shiri, N.: A Parametric Approach to Deductive Databases with Uncertainty, *IEEE Transactions Knowledge Data Engineering*, **13**(4), 2001, 554–570.

[29] Loyer, Y., Straccia, U.: Default Knowledge in Logic Programs with Uncertainty, *Proc. of the 19th Int. Conf. on Logic Programming (ICLP-03)*, number 2916 in Lecture Notes in Computer Science, Springer Verlag, Mumbai, India, 2003.

[30] Lu, J. J.: Logic Programming with Signs and Annotations, *Journal of Logic and Computation*, **6**(6), December 1996, 755–778.

[31] Lukasiewicz, T.: Probabilistic logic programming with conditional constraints, *ACM Transactions on Computational Logic*, **2**(3), 2001, 289–339, ISSN 1529-3785.

[32] Lukasiewicz, T.: Probabilistic logic programming with conditional constraints, *ACM Trans. Comput. Logic*, **2**(3), 2001, 289–339, ISSN 1529-3785.

[33] Lukasiewicz, T.: Probabilistic Description Logic Programs., *Symbolic and Quantitative Approaches to Reasoning with Uncertainty, 8th European Conference, ECSQARU 2005, Barcelona, Spain, July 6-8, 2005, Proceedings* (L. Godo, Ed.), 3571, Springer, 2005.

[34] Lukasiewicz, T.: Fuzzy Description Logic Programs under the Answer Set Semantics for the Semantic Web., *Rules and Rule Markup Languages for the Semantic Web, Second International Conference, RuleML 2006, Athens, Georgia, USA, November 10-11, 2006, Proceedings* (T. Eiter, E. Franconi, R. Hodgson, S. Stephens, Eds.), IEEE Computer Society, 2006.

[35] McGuiness, D. L., van Harmelen, F.: OWL Web Ontology Language Overview, 2004, W3C Recommendation `http://www.w3.org/TR/owl-features/`.

[36] Medina, J., Ojeda-Aciego, M., Vojtáš, P.: Multi-adjoint Logic Programming with Continuous Semantics, *Logic Programming and Nonmonotonic Reasoning, 6th International Conference, LPNMR 2001, Vienna, Austria, September 17-19, 2001, Proceedings* (T. Eiter, W. Faber, M. Truszczynski, Eds.), 2173, 2001, ISSN 0302-9743.

[37] Muggleton, S.: Stochastic Logic Programs, *Journal of Logic Programming*, 2001, Accepted subject to revision.

[38] Pan, J. Z., Stoilos, G., Stamou, G., Tzouvaras, V., Horrocks, I.: f-SWRL: A Fuzzy Extension of SWRL, *Journal of Data Semantic, special issue on Emergent Semantics*, **4090**, 2006, 28–46.

[39] Poole, D.: The Independent Choice Logic for Modelling Multiple Agents Under Uncertainty., *Artif. Intell.*, **94**(1-2), 1997, 7–56.

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

[41] Stoilos, G., Stamou, G., Tzouvaras, V., Pan, J.: Uncertainty and RuleML Rulebases:A Preliminary Report, *Proceedings of the International Conference on Rules and Rule Markup Languages for the Semantic Web* (A. Adi, Ed.), 3791, Springer-Verlag, 2005.

[42] Straccia, U.: Transforming fuzzy description logics into classical description logics, *Proc. of the 9th European Conf. on Logics in Artificial Intelligence (JELIA-04)*, 2004.

[43] Straccia, U.: Query Answering in Normal Logic Programs under Uncertainty, *8th European Conferences on Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU-05)*, number 3571 in Lecture Notes in Computer Science, Springer Verlag, Barcelona, Spain, 2005.

[44] Straccia, U.: Fuzzy Description Logic Programs, *Proceedings of the 11th International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems, (IPMU-06)*, 2006.

[45] Straccia, U.: Query Answering under the Any-World Assumption for Normal Logic Programs, *Proceedings of the 10th International Conference on Knowledge Representation (KR-06)*, 2006.

[46] Vennekens, J., Verbaeten, S., Bruynooghe, M.: Logic Programs with Annotated Disjunctions., *Proc. of ICLP 2004* (B. Demoen, V. Lifschitz, Eds.), 3132, Springer, 2004.

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

[48] Vojtáš, P., Paulík, L.: Soundness and Completeness of Non-classical SLD-Resolution, *Extensions of Logic Programming, 5th International Workshop, ELP'96, Leipzig, Germany, March 28-30, 1996, Proceedings* (R. Dyckhoff, H. Herre, P. Schroeder-Heister, Eds.), 1050, Springer, 1996, ISBN 3-540-60983-0.

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