Implementing Uncertainty in a Logic Programming Framework

Trevor Martin

Artificial Intelligence Group,
Department of Engineering Mathematics
University of Bristol
Bristol BS8 1TR, UK
email: trevor.martin@bris.ac.uk

Abstract. We briefly outline the need to incorporate uncertainty and flexibility into the semantic web knowledge representation, and argue that s*upport logic programming* - a combination of soft computing and logic programming – is one way to address this problem. To retain consistency with emerging frameworks, we show how a support logic program modelling uncertainty in relations and rules can be compiled into crisp Horn clauses suitable for reasoning within a system which does not explicitly model uncertainty. In cases where there is uncertainty in attribute values, an extension to the inference mechanism is required.

Keywords: Soft semantic web, support logic, probabilistic rules, Fril, program transformation.

1 Introduction

Recent development of the semantic web has included the adoption of logic programming (in various guises) as a key element in processing semantic web rules. In turn, this has led to a re-emergence of the question of whether it is necessary to include an explicit mechanism for handling uncertainty in the representation, and how best to process the uncertainty. During the "boom" years of logic programming, much research effort (see e.g. [1-4]) went into combining logic programming with various uncertainty calculi, both from the theoretical and implementation viewpoints. Fril is a practical implementation of logic programming with uncertainty, which has been used in a number of applications (see for example chapter 8 of [5]).

In this paper, we outline some of the design features of Fril in the belief that they contain relevant lessons for the semantic web. We focus on implementation aspects, not on properties such as expressibility or model-theoretic semantics which are covered elsewhere [4, 6-8]. Perhaps the key issue in the implementation of an uncertain logic programming language is whether to incorporate the processing of uncertainty as a core part of the system or to build a meta-level on top of an existing (crisp) implementation. The latter approach tends to be slower (as one is effectively building an interpreter) but has the advantage of compatibility with existing systems;

the former approach gives the implementer more control and flexibility. Fril was implemented as an extended Warren Abstract Machine, and took the first route above. In section 3 we outline the use of transformations to run uncertain logic programs within a standard system (without uncertainty in attribute values). This may be a more productive route, given the increasing commercial interest in implementing semantic-web related systems (e.g. Oracle 10 includes RDF management).

1.1 The Need for Uncertainty

The semantic web aims to bridge the human-machine communication gap "… *the theme is human beings doing the thinking and machines helping it work on a larger scale* …" [9], p 187-8). In order to achieve this, the semantic web provides definitions for terms used in natural language, and a logic-style framework for reasoning. Representations based on first order logic have many advantages - logic is easily understood and provides a powerful computing formalism through logic programming. However, human language is far more subtle and expressive than a formal model, and deals with many concepts that are defined by common usage rather than by necessary and sufficient conditions. Such vague concepts are widespread in attribute values (Mary's age is *young*), relations (80mph is a *safe speed* on an open highway, Star Wars Episode II is a *romantic movie)* and rules (person X *is likely to* have *high* blood pressure IF person X is *overweight*) where the italicised terms are vague. The inherent vagueness used in many natural language terms means there is often a mis-match between natural language and crisp logic-based representations.

In common with many knowledge representational formalisms, it is clear that the semantic web requires a mechanism that is able to cope with this type of knowledge.

1.2 How should we deal with uncertainty ?

From a mathematical perspective, it is difficult to argue against the use of probability as a tool for handling uncertainty. Indeed, it is relatively easy to show that if one is prepared to bet as an indication of one's level of certainty about an event then it is not rational to base one's behaviour on anything other than the laws of probability. However, it is worth remembering the words of Einstein:

> *As far as the laws of mathematics refer to reality, they are not certain;*
> *and as far as they are certain, they do not refer to reality*
> *("Geometry and Experience" 1921)*

The assumptions underlying probability include

- the availability of precise definitions for events and
- procedures to determine whether or not a given event has occurred.

Much human knowledge and communication is based on natural language, and one of the strengths of natural language is its capacity to efficiently convey a large amount of information relatively compactly. This relies on a shared understanding of terms, without necessarily sharing precisely the same definition of terms. For instance, an Englishman announcing that he is "travelling to Europe" would be understood to mean somewhere on the opposite side of the channel to England; an American saying

the same thing would probably include the UK as a possible destination. The word "Europe" denotes a collection of countries, but its precise definition is elusive - is it a set of countries marked as Europe on a particular map, members of the European union (now? in 1970? in 1975? in 2010?), countries eligible to enter European Championship football, countries eligible to enter the Eurovision song contest .... ?

Each "source" has its own definition of the term, but we are able to understand and use the concept in communication without the inconsistency causing a problem. Thus when considering the integration of different sources, we may need to consider uncertainty in class memberships, in relations and in rules. To illustrate, let us consider two ontologies which define the class *youngPerson*. Since *young* is not a precisely defined concept, it is likely that the two will adopt different standards – let us say ontology *o1* defines all people aged 20 and under as belonging to the class *o1:youngPerson* and ontology *o2* defines all people aged 25 and under as belonging to the class *o2:youngPerson*. How should we combine these definitions? Two extremes would be to use the union or intersection of the extensions

$$c{:}\ youngPerson = o1{:}youngPerson \cap o2{:}youngPerson$$

or

$$c{:}\ youngPerson = o1{:}youngPerson \cup o2{:}youngPerson$$

where *c* denotes the combined ontology. A more natural choice would be to acknowledge that the source of the problem is vagueness in the (combined) class definition and say that cases where the separate ontologies disagree should be modelled by an intermediate class membership or probability, related to the proportion of ontologies that would classify a particular instance as included or excluded from the class. In the case above, there is no disagreement in the case of instances with age 20 or less (i.e. *o1*'s definition), or in ages above 25. However for instances with age in the interval (20, 25] there is intermediate support for their membership in the combined class.

This combination is not based on similarities between the labels, but on the fact that the ontologies may classify the same underlying instances into different classes. In the same way, one movie database might class a particular movie as "*horror*" whilst another labels it as "*suspense*". The degree of overlap in the categories can give support for rules integrating both sources [10].

2 Fril and Support Logic Programming

Fril [5] is a logic programming language incorporating uncertainty at a fundamental level. Whilst not (yet) available as a web tool, the framework and implementation are relevant to the handling of uncertainty within the semantic web. The calculus for handling uncertainty is based on mass assignment theory [5, 11, 12], which gives a coherent framework for dealing with both fuzzy and probabilistic uncertainty. Both the language and underlying theory are dealt with in detail elsewhere [5, 13, 14] ; the essential extension compared to a logic program is that clauses may be quantified by support pairs. These are interpreted as intervals containing conditional probabilities, so for the simple rule "*p if q*" we could write

p IF *q* : *([u1 v1] [u2 v2])*

which is interpreted as stating that the conditional probabilities obey

$u1 \leq \Pr(p|q) \leq v1$

$u2 \leq \Pr(p|\neg q) \leq v2$

The symbols *p* and *q* represent predicates with arguments, not simple propositions. A more general form of rule is also allowed, in which more complex interactions in the body of the rule can be taken into account. For example, given a rule

p IF *q* AND *r*

the general rule takes account of different probability intervals for $p|qr$, $p|q\neg r$, $p|\neg qr$, $p|\neg q\neg r$. This can be used to model Bayesian nets, although the inference is limited to the direction imposed by the rule structure, i.e. conclude rule head from rule body. For simplicity we only consider the basic Fril rule here, which in the above case would only consider intervals for $p|qr$ *and* $p|\neg(qr)$. Clearly this leads to considerable efficiency savings as the number of goals in the rule body increases, and has been found adequate in most practical problems. By convention *[u2 v2]* defaults to [0, 1] in which case it may be omitted, and *[u1 v1]* defaults to [1, 1]

Unit clauses (facts) may also be quantified by support pairs, in the case that tuples do not completely satisfy a predicate - for example, take the relation

Fluent-Speaker $\subseteq$ *Person* $\times$ *Language*

and consider whether tuples *(John, English)*, *(John, French)*, *(John, Spanish)* satisfy this relation. If John is a native English speaker, and knows no French, then the first two tuples respectively belong and do not belong to the relation. However, if John knows a little Spanish then the final tuple lies between these two extremes. In the voting model [12], each voter must decide whether or not a tuple belongs to the relation and the (point valued) support pair is then the proportion of voters who vote yes. This can be extended to give general support pairs *[u v]* where *u* is the proportion voting in favour, *1-v* is the proportion voting against and *v-u* is the proportion of abstentions. Clearly it is meaningless to include a second support pair for unit clauses.

Inference in Fril proceeds by constructing a proof tree, in the same depth-first manner as Prolog. The supports of the clauses used in the proof tree are then combined to give a support for the solution.

By default, the support for a conjunction of goals is the product of the individual supports, i.e. given two facts

q1 : [x1 y1]

q2 : [x2 y2]

the support for the conjunction *q1 AND q2* is *[x1*x2 y1*y2]*. In general, one can only say that the support lies in the interval

[MAX(0, x1+x2-1), MIN(y1, y2)]

but by assuming maximum entropy we can use the product to calculate the support for the conjunction (see [5] for discussion of this point).

Consider a single basic Fril rule of the form

p IF q1 AND q2 AND … AND qn : $([u_1\ v_1]\ [u_2\ v_2])$

when the following facts are given or derivable

$qi : [u_i\ v_i] \quad 1 \leq i \leq n$

More generally the facts may not completely match the terms in the rule and the support pairs $[a_i\ b_i]$ will be determined using semantic unification. Let the combined

support for the body of the rule be *[x y]*. A generalised Jeffrey's rule for support pairs is the basic inference rule of support logic, so that the inference is

$p : [z1\ z2]$

where

$$z1 = \begin{pmatrix} u1 \cdot y + u2 \cdot (1-y)\ if\ u1 \le u2 \\ u1 \cdot x + u2 \cdot (1-x)\ if\ u1 > u2 \end{pmatrix} \qquad z2 = \begin{pmatrix} v1 \cdot x + v2 \cdot (1-x)\ if\ v1 \le v2 \\ v1 \cdot y + v2 \cdot (1-y)\ if\ v1 > v2 \end{pmatrix}$$

For example, the case discussed above can be modelled by the probabilistic rules

c:youngPerson(X) IF o1:youngPerson(X) : [1, 1]

c:youngPerson(X) IF o2:youngPerson(X) : [0.5, 1]

where *[0.5, 1]* is the support or interval probability that the rule head *c:youngPerson(X)* is true, given that the rule body *o2:youngPerson(X)* is true. The first rule has a support of [1,1] which is the default value for a rule and is normally omitted for clarity. This approach can be extended to multiple rules, (although we note that it is unable to deal with the case where two ontologies have no overlap in classification since we are then effectively dealing with two different concepts.). If all information is taken into account and negation can be properly handled within the rules, then it is possible to work without intervals e.g. we could have a rule that *x* is in *c:youngPerson* with a probability of 0.5 if *x* is in *o2:youngPerson* and not in *o1:youngPerson*. The interval in the second rule arises because we ignore the information given by *o1*, and only consider *o2*. This projection onto one ontology simplifies computation considerably at the expense of working with intervals rather than point values. Ding [15] takes a slightly different approach, allowing limited interaction between rules and compiling the uncertainty into Bayes nets.

We note in passing that if we consider that each ontology corresponds to an (equally likely) possible world, our approach has several links to probabilistic logic programming described by [16]. We also note that this treatment assumes both ontologies are concerned with the same set of instances.

To answer a query, a standard logic programming proof tree is created with extended unification to deal with uncertain attribute values; a probabilistic calculation is then carried out over the proof tree to determine the support for the conclusion. All proof paths are examined to determine the overall support.

If multiple proof paths are available for a conclusion, the overall support is the intersection of supports from the individual proof paths. This approach differs from other fuzzifications of Prolog. Uncertainty is expressed as a conditional probability of the rule head given the body. In contrast, most theoretical and practical treatments of uncertainty in logic programming associate uncertainty with the implication. As is well known from fuzzy control and multi-valued logics, there are very many plausible implication operators and much time and effort has been devoted to arguing their relative merits. The support logic programming approach avoids this controversy.

3 Implementing the Support Logic Calculus

Inference in a support logic program proceeds by constructing a proof tree. The supports of the clauses used in the proof tree are combined to give a support for the solution. Three different cases need to be considered:

- ***combination*** of multiple proof paths for an atom
- ***conjunction*** of different atoms in a rule body
- ***conditional*** support for the rule head from the rule support and support for the body

3.1 The *comb* operator

In normal logic programming, a single solution to a query is sufficient - it proves that a solution exists, and finds instantiations of the query variables corresponding to this solution. In support logic programming, the situation is different. Consider the rules

p IF *q*
p IF *r*

and let us suppose that *q* and *r* are true (with different supports). We have two proof paths for the conclusion *p*, with supports :

p : [x1 y1]

from the first proof path, and

p : [x2 y2]

from the second. By default, Fril assumes that the overall support must be consistent with *both* proof paths, and deduces

p : [max (x1, x2), min(y1, y2)]

subject to the support being a non-empty interval.

If there is more than one proof path for an atom, we look for a support pair which is compatible with all of them, i.e. the intersection of the intervals. Thus if atom *b* has support [*li ui*] from proof tree *i*, the overall support for *b* is

$$\bigcap_i [l_i, u_i] = \left[\max_i(l_i), \min_i(u_i)\right]$$

If this interval is empty, there is an inconsistency in the program. We define the (binary) support combination operator, *comb*, as

$$comb([l_1, u_1], [l_2, u_2]) = [\max(l_1, l_2), \min(u_1, u_2)]$$

This is monotonic, commutative and associative.

comb(S1, S2) $\subseteq$ *comb(S3, S2)* if *S1* $\subseteq$ *S3*
comb(S1, S2) = comb(S2, S1)
comb(S1,comb(S2,S3))=comb(comb(S1,S2),S3)

for arbitrary support pairs *S1*, *S2*, *S3* where *S=[l,u]*.

3.2 The *conj* operator

By default, the support for a conjunction of goals is the product of the individual supports, i.e. given two facts

$b1 : [l_{b1}, u_{b1}]$

$b2 : [l_{b2}, u_{b2}]$

the support for the conjunction *b1 AND b2* is given by the conjunction operator

$$conj([l_{b1}, u_{b1}], [l_{b2}, u_{b2}]) = [l_{b1} \times l_{b2} \;\; u_{b1} \times u_{b2}]$$

which again is monotonic, commutative and associative. In addition, we note that *comb* distributes over *conj* i.e.

$$conj(comb(S_1,S_2),S_3) = comb(conj(S_1,S_3),conj(S_2,S_3))$$

for arbitrary support pairs *S1*, *S2*, *S3*. Thus if we have two atoms *b1* and *b2* with *k1* proof paths for *b1* (with supports S_1^i, *i=1…k1*) and *k2* proof paths for *b2* with supports S_2^j, *j=1…k2*, we can calculate the overall body support from the supports for the *k1*k2* proof paths for the conjunction *(b1 AND b2)*. Because of the associativity of *comb* and *conj*, this can be extended to any finite number of proof paths for a finite number of atoms in a rule body.

In general, one can only say that the support for a conjunction lies in the interval

$$[\text{MAX}(0, l_{b1}, +l_{b2}\text{-}1), \text{MIN}(u_{b1}, u_{b2})]$$

but by assuming maximum entropy we can use the product to calculate the support for the conjunction (see [5] for discussion of this point).

3.3 The *cond* operator

Consider a single basic Fril rule of the form

h IF b1 AND b2 AND … AND bn : Sr

where *h, bi (i=1…n)* are atoms forming the head and body of the rule respectively. Support logic allows a rule support to be of the form $[S_{hb}, S_{hnb}]$ where S_{hb} is the support for the head given that the body is true and S_{hnb} is the support for the head given that the body is not true. If

$$Sr = (S_{hb}, S_{hnb}) = ([l_{hb}, u_{hb}], [l_{hnb}, u_{hnb}])$$

when the following facts are given or derivable by single proof trees

$$b_i^j : S_i^j \qquad 1 \le i \le n, \;\; 1 \le j \le ki$$

(where the body atom *bi* has *ki* proof paths). Let the combined support for the body of the rule be *Sb*, calculated by repeated application of the *conj* and *comb* operators defined above. If *Sr* is the rule support, the support for *h* is calculated using a generalised Jeffrey's rule for support pairs

$$cond(Sr, Sb) = [l_h, u_h]$$

where

$$l_h = \begin{pmatrix} l_{hb} \times u_b + l_{hnb} \times (1 - u_b) \; if \;\; l_{hb} \le l_{hnb} \\ l_{hb} \times l_b + l_{hnb} \times (1 - l_b) \; if \;\; l_{hb} > l_{hnb} \end{pmatrix} \qquad u_h = \begin{pmatrix} u_{hb} \times l_b + u_{hnb} \times (1 - l_b) \; if \;\; u_{hb} \le u_{hnb} \\ u_{hb} \times u_b + u_{hnb} \times (1 - u_b) \; if \;\; u_{hb} > u_{hnb} \end{pmatrix}$$

(see [5] for discussion of the use of Jeffrey's rule as a special case of updating mass assignments). By case analysis of this expression, it is easy to see that the *cond* operator is monotonic in the body support, i.e.

$$S_{b1} \subseteq S_{b2} \text{ iff } cond(Sr, S_{b1}) \subseteq cond(Sr, S_{b2})$$

for a fixed rule support Sr and body supports S_{b1}, S_{b2}. Additionally,

$$cond(Sr, comb(S_{b1}, S_{b2})) = comb(cond(Sr, S_{b1}), cond(Sr, S_{b2}))$$

hence we can calculate the support for a solution by finding a proof path, computing its support, then looking for another proof path and intersecting its support with that already found. This is equivalent to a depth-search of the SLD tree [17], recording each solution and its support. In contrast, a straightforward formulation of support logic requires a breadth-search of the tree, where the support at an or-node can only be calculated when all sub-trees rooted at that node have been evaluated.

Instead of calculating

$$\left[l_h, u_h\right] = cond\left(S_r, \underset{i=1\ldots n}{conj}\left(\underset{j=1\ldots ki}{comb}\left(S_i^j\right)\right)\right)$$

we find

$$\left[l_h, u_h\right] = comb\left(cond\left(S_r, \underset{\substack{i=1\ldots n \\ j=1\ldots ki}}{conj}\left(S_i^j\right)\right)\right)$$

where the n-ary operators are obvious extensions from the binary case.

In Prolog terms, this corresponds to an "all-solutions" query at the root node only, as compared to all-solutions queries for each goal appearing in the search tree. The efficiency gain is significant. We can transform a support logic program to a crisp logic program by adding a single argument to the head and to each body goal, to represent the support computation. Thus a rule

```
h(X,Y)  IF b1(X,Z) AND b2(Z,Y) : Sr
```

becomes

```
h(cond(Sr, conj(S1,S2)), X, Y)
                        IF  b1(S1, X,Z) AND b2(S2, Z,Y)
```

and a fact

```
b1(a, b) : Sf
```

becomes

```
b1(Sf, a, b)
```

Placing the support as the first argument is an arbitrary but useful convention. A query then changes from

$$? \; h\,(X, Y)$$

to

$$? \; h(S, X, Y)$$

followed by evaluation of the support expression *S* for all proof paths.

The compilation of supported clauses to crisp clauses with embedded support arguments is straightforward, as is the mechanism to find all solutions. Compilation (or run-time translation) of uncertain logic programs into standard logic programs has been implemented before (e.g. [13, 18] by adding extra goals to the rule body. The disadvantage of this approach is that it creates an execution overhead and may disrupt compiler optimisations. In contrast, the method outlined above uses an extra argument and only marginally affects execution speed by adding a single unification step. It fits into the crisp logic programming approach proposed for the semantic web. Hence we

can write (or learn) support logic rules and facts, and evaluate uncertain conclusions without having to extend the whole framework.

3.4 Uncertainty in Attribute Values

We have so far ignored the question of uncertain attribute values - for example, suppose we know that (i) Mary's age is 28 or 29, or (ii) Mary is in her late twenties. The first case is representable (but awkward) in logic, the second is essentially fuzzy as different values satisfy "late twenties" to a greater or lesser degree. This can be represented by a fuzzy set, interpreted as a fuzzy constraint on values of the attribute - i.e. there should be a single value, but it is not precisely known.

Fuzzy sets can also be used to represent uncertain values in rule bodies e.g.

```
company X performed well in year Y  IF
          turnover-of X in Y is high-turnover  AND
          profit-of X in Y is about10%
```

where the italicised terms are fuzzy sets.

This requires us to generalise the unification process since the presence of fuzzy attributes allows partial matching. For example, assume that the height of Bob is *above_average* and we have a query to identify *tall* people. Fril uses semantic unification, to determine a conditional probability for the matching of two terms and hence calculates a support pair for the goal. Essentially, a virtual rule is assumed

```
ht of Bob is tall IF ht of Bob is above_average  : S
```

where *S* is calculated from the probability of matching the fuzzy sets. Note that this process is asymmetric - the probability of *tall* given *above_average* is not necessarily the same as the probability of *above_average* given *tall*. This is obvious if one considers a crisp case - for example, define a small dice value as {1, 2} and even as {2, 4, 6}. Clearly $Pr(small \mid even) \neq Pr(even \mid small)$

This process of matching fuzzy sets is known as *semantic unification* to distinguish it from the normal logic unification method, which is purely syntactic. Semantic unification is a fundamental part of Fril. A support pair is automatically calculated for the match, and incorporated into the overall calculation of support for the query, using probabilistic semantic unification. Further details are given in [5] and details of fast algorithms suitable for implementing semantic unification in a Warren Abstract Machine architecture are given in [19, 20]

4 Summary

The aim of the semantic web is to make content both human and machine understandable. We claim that this means avoiding artificially precise definitions and modelling human-understandable terms by adopting a representation that handles uncertainty naturally and efficiently. Considerable existing work has been devoted to incorporating uncertainty into logic programming - uncertain facts and rules can be executed efficiently within a standard framework but uncertain attribute values require more fundamental modification.

References

1. Martin, T.P. and F. Arcelli Fontana, *Logic Programming and Soft Computing - an Introduction*, in *Logic Programming and Soft Computing*, T.P. Martin and F. Arcelli Fontana, Editors. 1998, RSP/Wiley: UK. p. 1-18.
2. Martin, T.P., J.F. Baldwin, and B.W. Pilsworth, *Implementation of Fprolog - a Fuzzy Prolog Interpreter*. Fuzzy Sets & Systems, 1987. **23** p. 119-129.
3. van Emden, M., *Quantitative Deduction and its Fixpoint Theory.* J.Logic Prog, 1986. **3**: p. 37-53.
4. Kifer, M. and V.S. Subrahmanian, *Theory of generalized annotated logic programming and its applications*. J Logic Prog, 1992. **12**: 335 - 367.
5. Baldwin, J.F., T.P. Martin, and B.W. Pilsworth, *FRIL - Fuzzy and Evidential Reasoning in AI*. 1995, U.K.: Research Studies Press (John Wiley). 391.
6. Krajci, S., R. Lencses, and P. Vojtas, *A comparison of fuzzy and annotated logic programming*. Fuzzy Sets and Systems, 2004. **144**(1): p. 173-192.
7. Vojtas, P., *Fuzzy logic programming.* Fuzzy Sets and Systems, 2001. **124**(3): p. 361-370.
8. Damasio, C.V. and L.M. Pereira. *Sorted Monotonic Logic Programs and their Embeddings*. in *IPMU-04*. 2004. Perugia, Italy.
9. Berners-Lee, T., *Weaving the Web*. 1999, London: Texere. 272.
10. Martin, T.P. and Y. Shen, *Improving access to multimedia using multi-source hierarchical meta-data*, in *Adaptive Multimedia Retrieval: User, Context, and Feedback*. 2006, Springer. p. 266 - 278.
11. Baldwin, J.F., *A Calculus For Mass Assignments In Evidential Reasoning*, in *Advances in the Dempster-Shafer Theory of Evidence*, M. Fedrizzi, J. Kacprzyk, and R.R. Yager, Editors. 1992, John Wiley: N.Y.
12. Baldwin, J.F., *The Management of Fuzzy and Probabilistic Uncertainties for Knowledge Based Systems*, in *Encyclopedia of AI*, S.A. Shapiro, Editor. 1992, John Wiley. p. 528-537.
13. Baldwin, J.F., *Support Logic Programming*, in *Fuzzy Sets - Theory and Applications*, A. Jones, Editor. 1986, D. Reidel. p. 133-170.
14. Baldwin, J.F. and T.P. Martin, *An Abstract Mechanism for Handling Uncertainty*, in *Uncertainty in Knowledge Bases*, B. Bouchon-Meunier, R.R. Yager, and L.A. Zadeh, Editors. 1991, Springer Verlag. p. 126-135.
15. Ding, Z. and Y. Peng. *A Probabilistic Extension to Ontology Language OWL*. in *37th Hawaii Int Conf on System Sciences (HICSS'04)*. 2004: IEEE.
16. Dekhtyar, A. and V.S. Subrahmanian, *Hybrid probabilistic programs*. Journal of Logic Programming, 2000. **43**(3): p. 187-250.
17. Lloyd, J.W., *Foundations of Logic Programming*. 2nd ed. 1987: Springer.
18. Baldwin, J.F., *Evidential Support Logic Programming.* Fuzzy Sets and Systems, 1987. **24**: p. 1-26.
19. Baldwin, J.F. and T.P. Martin. *Fast Operations on Fuzzy Sets in the Abstract Fril Machine*. in *First IEEE Int Conf on Fuzzy Systems*. 1992. IEEE Press.
20. Baldwin, J.F., J. Lawry, and T.P. Martin. *Efficient Algorithms for Semantic Unification*. in *Inf. Processing and Management of Uncertainty*. 1996. Spain.