Expressive Power of Logic Frameworks with Certainty Constraints

Nematollaah Shiri
Concordia University
Dept. of Computer Science & Software Engineering
Montreal, Quebec, Canada
shiri@cse.concordia.ca

Abstract

Uncertainty reasoning has been a challenging issue in AI and database research. During the last two decades, numerous frameworks have been proposed by extending the standard logic programming and deductive database systems. These frameworks vary in different ways including the way in which uncertainties are associated with the facts and rules in programs. On the basis of this, we have classified the approaches of these frameworks into "annotation based" (AB) and "implication based" (IB). In the AB approach, uncertainties are associated with the atoms whereas in the IB approach they are associated with implications. Except for some particular cases, the exact relationship between the two approaches has not been studied previously, in that, the two are treated as somewhat orthogonal and unrelated. In this paper, we investigate this issue and introduce the notion of certainty constraint. The purposes are two fold. First, this notion supports the operations of selection and join by certainty, which are often useful in query formulation and processing in the context of uncertainty. Second and more importantly, this notion "relates" the expressive power of the two approaches. To be precise, we propose a transformation technique which establishes that the AB and IB approaches are equally expressive when the rule bodies are extended with certainty constraints. A consequence of this result is that, using the transformation technique, we can adopt and use successful developments in either approach.

Introduction

Many real-life applications require an ability to represent, manage, and reason with uncertain information. Uncertainty arises when the information is of limited reliability, i.e., the truth of information is not established definitely. Uncertainty is measured by some individual or device, and is represented by associating with the information a value coming from an appropriate domain. Silberschatz et al. (Silberschatz, Stonebraker, & Ullman 1991) identified uncertainty management as an important challenge: "further research in *uncertainty* is essential, as we must learn not only to cope with data of *limited reliability*, but do so *efficiently* with *massive amounts of data.*"

Logic database programming with the advantage of being declarative and modular, and with its powerful top-down and bottom-up query processing techniques has been a primary choice for modeling uncertainty for which numerous frameworks have been proposed by extending the standard logic programming and deductive databases with uncertainty (Shapiro 1983; van Emden 1986; Subrahmanian 1987; Kifer & Li 1988; Ng & Subrahmanian 1992; Dubois, Lang, & Prade 1991; Fitting 1991; Kifer & Subrahmanian 1992; Lakshmanan & Sadri 1994; ; Lakshmanan & Shiri 1996).

These proposals typically offer a framework in which deduction is combined with some form of uncertainty, including certainty values, fuzzy, probabilities, possibilities, hybrid of numeric and symbolic formalisms, etc. As in the standard case, these frameworks offer a declarative semantics of programs. On the operational side, this is supported by a sound and complete (or weakly complete) proof theory and a corresponding fixpoint semantics. On their approaches, these frameworks may differ in several ways, e.g., (1) in the mathematical foundation of the underlying uncertainty notion, (2) in the way they manipulate uncertainties, and (3) in the way uncertainties are associated with the facts and rules in a program. On the basis of the third, these frameworks are classified into *annotation based* (AB, for short) and *implication based* (IB) approaches, as described below (Lakshmanan & Shiri 2001b).

Two operations often useful in the context of uncertainty for query formulation and processing are *selection by certainty* and *join by certainty.* For example, the first operation may be used in queries such as: "list all objects whose degree of redness is at least 0.8". For the join by certainty operation, an example would be restricting the joins in a rule body to tuples whose certainties stand in particular relationships. To support these two operations, we introduce the notion of *certainty constraint*, which may be added to rule bodies to restricts their evaluations. In addition to support the two useful operations, it turns out that this notion is more important in that it "relate" the expressive power of the AB and IB approaches, which were viewed as somewhat orthogonal and not so related formalisms so far, except in some special cases (For instance, the generalized AB framework (GAP) can simulate, through the annotation variables, the computation of the IB framework of van Emden.) In this paper, we investigate the relative expressive power of the AB and IB approaches. For our result to be general in this study, we take a framework independent approach by first introducing two generic AB and IB frameworks, each of which would include the main characteristics and common features of the corresponding approach it represents. We will then extend these generic frameworks with certainty constraints in the rule bodies. As our main contribution in this paper, we show that the generic AB and IB frameworks are equally expressive, when extended with certainty constraints.

The rest of this paper is organized as follows. We next review the basic concepts and techniques of logic programming and deductive databases and introduce some notations. In section 3, we introduce two generic AB and IB frameworks, and then extend both certainty constraints in the rule bodies. In section 4, we introduce a transformation technique, using which we establish the equivalence of these generic AB and IB frameworks which allow certainty constraints in the rule bodies. This is followed by a discussion on the continuity of the fixpoint operators. We conclude the paper with some remarks and and future work.

Background and Notations

In this section, we review basic of logic programming and deductive databases with uncertainty. This also helps introducing the concepts and notations we use in this paper. We assume the reader is familiar with the foundations of logic programming (Lloyd 1987) and deductive databases (Ceri, Gottlob, & Tanca 1989).

In his seminal work on quantitative deduction, van Emden (van Emden 1986) proposed an IB framework, by considering the subset of Prolog language (without negation) in which the implications are associated with certainty factors. Examples of some IB frameworks include Fitting (Fitting 1988; 1991), Dubois et al. (Dubois, Lang, & Prade 1991), and Lakshmanan and Sadri (Lakshmanan & Sadri 1994;). Each rule r in an IB framework is an expression of the form

$$r: \quad A \overset{\alpha}{\longleftarrow} B_1, \ldots, B_n.$$

where A and B_i's are atomic formulas, and $\alpha \in (0, 1]$ is the rule certainty. As in the standard case, when $n = 0$, we call r as *fact*. Intuitively, rule r asserts that "the certainty that the rule body implies the head is α." In a sense, α controls the "propagation" of truth from the rule body to the head. Rule evaluation in (van Emden 1986) is as follows. Given an assignment of certainties to the B_i's, we first obtain the certainty of the rule body as a whole, using min. This certainty is then multiplied by the rule certainty α, which yields a certainty of the ground atom defined by the rule. Using max, alternate derivations of this atom are combined into a single certainty. This basically explains the notion of rule evaluation in IB frameworks, modulo the fact that the certainty domain and/or certainty functions could be different from one framework to another. For rule r above, we associate the triplet of certainty functions $\langle f_d, f_p, f_c \rangle$ in which f_d is the disjunction, f_p is the propagation, and f_c is the conjunction function. The order of functions listed in this triplet indicates, from right to left, the order in which the functions are applied when evaluating the rule, as described above. We will refer to the collection of these functions as *certainty functions*.

In the AB approach, on the other hand, implication is similar to the standard case, however, each atom in the rule is associated with a certainty value or variable. To be precise, a rule in an AB framework is an expression of the form:

$$r: \quad A : f(\beta_1, \ldots, \beta_n) \leftarrow B_1 : \beta_1, \ldots, B_n : \beta_n.$$

where f is a computable n-ary function, which plays the role of conjunction and propagation functions in the IB frameworks mentioned above, and β_i's are annotation constants or variables ranging over an appropriate certainty domain. This rule asserts "the certainty of A is at least (or is in) $f(\beta_1, \ldots, \beta_n)$, whenever the certainty of B_i is at least (or is in) β_i, for $1 \leq i \leq n$." Examples of the AB frameworks include the annotated logic programming of Subrahmanian (Subrahmanian 1987), Kifer and Li (Kifer & Li 1988), the probabilistic logic programming of Ng and Subrahmanian (Ng & Subrahmanian 1992; 1993), and the generalized theory of annotated logic programming (GAP) of Kifer and Subrahmanian (Kifer & Subrahmanian 1992).

We do not enter into a debate of which approach is the best. Rather, we emphasize that different frameworks may be appropriate for different applications. Furthermore, different forms of manipulating uncertainty may be required in some applications. See (Lakshmanan & Shiri 2001a) for a survey of uncertainty in logic programming and deductive databases. Further remarks on the AB and IB approaches are as follows. While the way implication is treated in the AB approach is similar to the standard case, the way rules are fired in the IB approach has a definite intuitive appeal (Kifer & Subrahmanian 1992). The AB approach in general is strictly more expressive than the IB. For instance, as shown in (Kifer & Subrahmanian 1992), the GAP framework can simulate the IB framework of van Emden, through annotation variables. The down side is that query processing (particularly resolution and fixpoint evaluation) in the AB approach is more complicated than the IB approach. For instance, unlike in the IB approach, unification is more involved in the AB approach, and resolution requires constraint solving. With respect to the fixpoint computation, we remark that the fixpoint operator in the IB approach is continuous while it is not necessarily the case in the AB. We will further comment on the continuity issue later when we present our results.

Generic AB and IB Frameworks

As different AB or IB frameworks may have different mathematical foundation of uncertainty and different combination functions, for their comparison to make sense, we introduce two generic frameworks, and AB and an IB, each of which includes common features of the corresponding framework it represents. We refer to these generic frameworks as GAB and GIB, respectively. We will introduce the notion of *certainty constraint* and further extend the GAB and GIB frameworks to allow certainty constraints in the rule bodies. We will then establish that these extended frameworks are equally expressive – an important result which "connects" the AB and IB approaches.

A useful operation in deductive databases and logic programming with uncertainty is to select from a relation, every tuple whose associated certainty value is not "less" than a specified threshold, e.g., as in the query "find all the red objects whose degree of redness is at least 0.75." We call this operation as *selection by certainty.* Another useful operation in this context is *join by certainty*, which amounts to prescribing that a pair of tuples from two relations can be joined provided their associated certainties stand in a certain relationship. We can view the selection/join by certainty as filtering mechanisms, by which we determine a (possibly empty) subset of tuples to be considered for the selection/join operations.

Neither of these operations is supported by any IB IB framework, including the parametric framework. The AB frameworks on the other hand enjoy a limited form of these operations through annotation constants and variables. We thus extend the AB and IB frameworks to support these operations. This is done by allowing *certainty constraints* in the rule bodies, defined formally as follows. Borrowing the syntax proposed by Halpern (Halpern 1990), given an atom A, we use the term $wt(A)$, read as *weight of A*, to denote the certainty of A. A *certainty constraint* is a boolean expression or a conjunction of such expressions each of which is of the form "$wt(A)\ \ \theta\ \nu$" or "$wt(A)\ \theta\ wt(B)$", where A and B are atoms, ν is a certainty constant or variable over the certainty domain, and $\theta \in \{\prec, \preceq, =, \neq, \succeq, \succ\}$. We can think of $wt(A)$ as a function call, returning the "current" certainty value associated with A, as the rule evaluation continues.

As is customary, we assume that the underlying certainty domain, denoted $\mathcal{T}$, is a complete lattice, partially ordered by $\preceq$; we may also use its dual $\succeq$. Also some frameworks, e.g., (Kifer & Li 1988), use multiset as the semantics structure. A multiset M is an annotated set whose elements are of the form $x : k$, where k denotes the multiplicity of x in M. Obviously, M reduces to a set if $k \in \{0, 1\}$. We will use $\{\!|\cdots|\!\}$ to denote multisets. Following our development of the parametric framework (Lakshmanan & Shiri 1996), the semantics of the generic AB and IB frameworks we introduce here is based on multisets. Furthermore, in order for the comparison of the AB and IB frameworks to be fair and make sense, we further assume that the certainty functions employed are subject to the same collection of reasonable properties, including monotonicity, continuity (w.r.t. Scott's topology), associativity, and commutativity. See (Lakshmanan & Shiri 1996) for the complete list of these properties and their justifications.

A Generic IB Framework

As the generic IB framework in this paper, we consider the parametric framework (Lakshmanan & Shiri 1996), which unifies and/or generalizes all the IB frameworks. We further extend this generic language with certainty constraints, denoted C_r, and defined below. Let us refer to this extended generic IB framework as EGIB. Each rule r in EGIB is an expression of the form:

$$r:\ p(\overline{Y})\ \xleftarrow{\alpha_r}\ q_1(\overline{Y}_1), \ldots, q_n(\overline{Y}_n), C_r; \langle f_d, f_p, f_c\rangle.$$

where p and q's are atoms, $\alpha_r \in \mathcal{T}$ is the rule certainty, C_r is (a possibility empty) conjunction of certainty constraints in the rule, and the combination functions from right to left indicate conjunction, propagation, and disjunction functions.

The notion of program satisfaction is similar to that of the parametric framework except that here certainty constraints should also be taken into account. Let P be a program in the EGIB framework. A valuation v of P is a mapping from the Herbrand base B_P of P to the certainty domain $\mathcal{T}$, which assigns to each ground atom A in B_P, a certainty value in $\mathcal{T}$, that is, $v(A) \in \mathcal{T}$. We use P^* to denote the ground instantiation of P, i.e., the collection of variable-free instances of every rule in P. Let $C_r \equiv C_1, \ldots, C_k$ be the conjunction of certainty constraints specified in the body of a rule $r \in P$, where C_ℓ, $1 \leq \ell \leq k$, is either an expression of the form (1) $wt(q_i(\overline{Y}_i))\, \theta\, \sigma$ or (2) $wt(q_i(\overline{Y}_i))\, \theta\, wt(q_j(\overline{Y}_j))$, θ is a comparison operator, and $\sigma \in \mathcal{T}$.

Given a valuation v, we say v *satisfies* a certainty constraint C_i, denoted $\models_v C_i$, provided $v(B_i)\, \theta\, \sigma$ is true in case (1), and $v(B_i)\, \theta\, v(B_j)$ is true in case (2), where B_i is a ground instance of the atomic formula $q_i(\overline{Y}_i)$. We say that v satisfies $C_1, \ldots, C_k$, provided $\models_v C_\ell$, for all ℓ, $1 \leq \ell \leq k$. We use these definitions to formalize the notion of program satisfaction, as follows.

Let r be any rule in an EGIB program P and v be any valuation of P. For any ground instance ρ of r, we say that v satisfies ρ, denoted $\models_v\ \ \rho$, iff $f_p(\alpha_r, f_c(\{\!|v(B_1), \ldots, v(B_n)|\!\})) \preceq v(A)$ and $\models_v\ \ C_r$. We say that v satisfies r, denoted $\models_v r$, iff v satisfies every instance of r. Finally, we say v satisfies P, denoted $\models_v P$, iff (1) $\forall r \in P : \models_v r$, and (2) $\forall A \in B_P : v(A) \succeq f_d(X)$, where $X = \{\!|f_p(\alpha_r, f_c(\{\!|v(B_1), \ldots, v(B_n)|\!\}))\ \ |\ (A \xleftarrow{\alpha_r} B_1, \ldots, B_n, C_r; \langle f_d, f_p, f_c\rangle) \in P^*$ and $\models_v C_r|\!\}$.

Unlike in standard logic programming and deductive databases, when uncertainty is present, it is possible that a rule r in P is satisfied by a valuation v while P itself is not satisfied. The last condition above is concerned with this case stating that in order for v to also satisfy for P, the certainty assigned to any ground atom A by v is not less than A's certainty derived from P, which is obtained by applying the disjunction function f_d to the multiset X of certainties derived from every instance ρ of $r \in P$ whose head is A such that $\models_v \rho$.

A Generic AB Framework

In this section we introduce a generic AB framework, GAB, that includes essential features of the AB approach. To this end, following our approach in the development of the parametric framework (Lakshmanan & Shiri 2001b). In order for the comparison of GAB and GIB to make sense, we require that the GAB framework uses multiset as its underlying semantics structure. We further assume that the combination functions allowed in GAB are subject to the same postulates in the GIB. Note that the annotation function used in the rule head in an AB framework plays the role of both conjunction and propagation functions. Also note that the GAB framework so defined does not exactly correspond to any existing AB framework. A technical issue here to note is that while the immediate consequence operator T_P used to define the fixpoint semantics of logic programs P is continuous in standard logic programming and IB frameworks, it is not the case for almost all AB frameworks, indicating that the least fixpoint evaluation of programs in AB frameworks may not terminate in finite time. We elaborate on this issue in the closing section.

Next we allow certainty constraints in the GAB framework. The syntax of a rule r in the extended GAB framework, called EGAB, is an expression of the form:

$p(\overline{Y}) : f(V_1, \ldots, V_n) \leftarrow q_1(\overline{Y}_1) : V_1, \ldots, q_n(\overline{Y}_n) : V_n, C_r.$

where V_i is a *certainty constant* or *variable*, for $1 \leq i \leq n$, and the constraint C_r is a conjunction of boolean expression of the form $V_i \, \theta \, V_j$ or $V_i \, \theta \, \sigma$, where θ is a comparison operator, and σ is a certainty value in $\mathcal{T}$.

The following example, adopted from (Lakshmanan 1994), illustrates the use of certainty constraints.

Consider a medical application where uncertain knowledge about particular diseases and symptoms is represented as the following program, in which we assume the triplet of certainty functions associated with each rule is $\langle max, *, min \rangle$.

$$\begin{aligned} r_1 : disease(X, D) &\xleftarrow{0.8} has(X, S),\ symptom(D, S),\\ &\quad wt(has(X, S)) \succeq 0.8,\\ &\quad wt(symptom(D, S)) \succeq 0.9.\end{aligned}$$

$$\begin{aligned} r_2 : disease(X, D) &\xleftarrow{0.9} family\_history(X, D),\\ &\quad hereditary(D),\\ &\quad wt(family\_history(X, D)) \succeq 0.8,\\ &\quad wt(hereditary(D)) \succeq 0.7.\end{aligned}$$

The use of constraints is viewed as a filtering mechanism. For instance, the likelihood of X having disease D is computed (by r_1) only when X has a symptom S with certainty $\succeq 0.8$, and S is associated with D with certainty $\succeq 0.9$. In particular, conclusions with "insignificant" certainties are automatically pruned.

Note that the extended GAB framework so defined is strictly more expressive than the GAB, and hence more expressive that any basic AB framework. This is because in the former, we can express a relationship between two annotations, while this could not be done in existing AB frameworks. For instance, suppose we want to perform a join operation on relations q and r for those tuples $t_q \in q$ and $t_r \in r$ such that the certainty of t_q is not "less" than the certainty of t_r. While this cannot be expressed in any existing AB framework, it can be expressed as the following rule in the extended GAB:

$$p(\overline{Y}) : f(V_1, V_2) \leftarrow q(\overline{Y}_1) : V_1, r(\overline{Y}_2) : V_2, V_1 \succeq V_2.$$

where f is any allowable annotation function that satisfies the postulates.

Equivalence of EGAB and EGIB

In this section, we show that the generic EGAB and EGIB frameworks with certainty constraints are equally expressive. This is formally stated as follows. We use D to denote the input set of facts, also called extensinal database, which is essentially a collection of atom-certainty pairs. For a program P and input collection of facts D, we use $P(D)$ to denote the set of atom-certainty paris obtained by a fixpoint evaluation of P on D.

Theorem: Given any program P_A in the EGAB framework, there exists a program P_I in EGIB such that for every input database D, program P_I computes the same atom-certainty pairs as P_A computes on D. That is, $P_A(D) = P_I(D)$, for all input database D.

Proof Sketch. The idea of the proof is based on simulation of programs in one formalism within the other. We use a transformation technique to establish this equivelence. That is, given any rule r in EGAB or EGIB, our transformation yields an equivalent rule r' in the other framework, in the sense that on every input database D, r and its transformed expression r' define the same *multiset* of atom-certainty pairs. This is done in two parts, by first assuimg r is in EGAB and then consider the case where r is a rule in EGAB. The transformation procedure is defined as follows.

First, consider the EGAB rule r in P_A defined earlier, repeated here for convenience.

$$r : p(\overline{Y}) : f(V_1, \ldots, V_n) \leftarrow q_1(\overline{Y}_1) : V_1, \ldots, q_n(\overline{Y}_n) : V_n, C_r.$$

This rule can be expressed as the following EGIB rule in which $\top$ is the top value in the certainty domain $\mathcal{T}$.

$$r' : p(\overline{Y}) \xleftarrow{\top} q_1(\overline{Y}_1), \ldots, q_n(\overline{Y}_n), C'_r; \langle f_d, f_p, f_c \rangle.$$

where C'_r is C_r with annotation variable V_i replaced by $wt(q_i(\overline{Y}_i))$, for $1 \leq i \leq n$. Here f_d is the disjunction function associated with the predicate p defined by r, and f_p is any propagation function satisfying the postulates which, among others, is required to satisfy $f_p(\alpha, \top) = \alpha$, $\forall \alpha \in \mathcal{T}$.

Now suppose r is the following EGIB rule in P_I.

$$r: \; p(\overline{Y}) \overset{\alpha_r}{\longleftarrow} q_1(\overline{Y}_1), \ldots, q_n(\overline{Y}_n), \; C_r; \; \langle f_d, f_p, f_c \rangle.$$

This rule can be expressed as the EGAB rule:

$$p(\overline{Y}) : f_p(\alpha_r, f_c(V_1, \ldots, V_n)) \leftarrow q_1(\overline{Y}_1) : V_1, \ldots \\ q_n(\overline{Y}_n) : V_n, C'_r.$$

where C'_r is the certainty constraint C_r in which the weight term $wt(q_i(\overline{Y}_i))$ replaced by the annotation variable V_i, for $1 \leq i \leq n$.

It follows from the transformation above that the resulting rule is equivalent to the given rule, in the sense that they compute the same atom-certainty pairs. Similarly, given any r in a EGIB program, we can express r as a rule in EGAB in which the annotation of each atom can be extracted from the weight terms wt in the constraints in r, the rest of constraints in r would become the constraints in the EGAB rule, and the annotation function in the rule is determined from the conjunction and propagation functions together used to define the head constraints. This equivalence at the rule level implies equivalence at the program level, since, by construction, the disjunction function associated with the head predicate in r and r' are identical.

The following example illustrates the transformation procedure. *Let P be the following program in EGAB.*

$$r_1 : disease(X, D) : 0.8 * min(V_1, V_2) \leftarrow \\ has(X, S) : V_1, \; symptom(D, S) : V_2, \\ V_1 \succeq 0.8, \; V_2 \succeq 0.9.$$

$$r_2 : disease(X, D) : 0.9 * min(V_1, V_2) \leftarrow \\ family\_history(X, D) : V_1, \\ hereditary(D) : V_2, V_1 \succeq 0.8, \; V_2 \succeq 0.7.$$

Following the transformation method proposed above, P would be transformed into an equivalent program in EGIB defined in Example **??**, assuming that max is the disjunction function associated with every predicate in P.

Note that although query processing in the AB framework of Kifer and Subrahmanian (Kifer & Subrahmanian 1992) may in general call for constraint solvers, constraints are not explicitly allowed in users' programs. Recall that a restricted form of constraints is implicitly allowed in the basic AB framework, i.e., associating the same annotation variable with more than one atom in the rule body, or associating an annotation constant. Adding constraints strictly increases the expressive power of the parametric and the AB frameworks. This also provides us with a common ground for comparing EGAB and EGIB. Our result above shows that when constraints are added, the extended GAB and GIB have the same expressive power.

Concluding Remarks and Future Work

In this paper we studied the expressive power of the AB and IB frameworks with uncertainty. As these frameworks differ in various ways, we first developed a generic framework that possess common features of AB frameworks. Similarly, we proposed a generic IB framework that includes common features of IB frameworks. We then extended these generic frameworks with certainty constraints, mainly to support the operations of selection/join by certainty. More importantly, this extension shed light on the expressive power of the AB and IB frameworks. More precisely, we showed the AB and IB approaches are equally expressive, when they are extended with certainty constraints.

While the fixpoint operator of the IB frameworks are continuous, the increased expressive power gained by allowing certainty constraints comes at the expense of continuity; the fixpoint operator of the EGIB is no longer continuous. This is true even for the majority of the proposed AB frameworks. The following example illustrates this point.

Let us now consider the following EGIB program P, in in which $\mathcal{T} = [0, 1]$ is the underlying certainty lattice.

$$r_1: \; p(X, Y) \overset{1}{\longleftarrow} e(X, Y); \; \langle f_{ind}, *, min \rangle.$$

$$r_2: \; p(X, Y) \overset{1}{\longleftarrow} e(X, Z), \; p(Z, Y); \langle f_{ind}, *, min \rangle.$$

$$r_3: \; r(X, Y) \overset{1}{\longleftarrow} p(X, Y), \; q(X, Y), wt(p(X, Y)) \succeq \\ wt(q(X, Y)); \langle f_{ind}, *, min \rangle.$$

where $f_{ind}(\alpha, \beta) = \alpha + \beta - \alpha\beta$ is the independence function, in the probabilistic sense. Suppose $D = \{e(1, 1) : 0.5, \; e(1, 2) : 0.5, \; q(1, 2) : 1\}$ is the set of input facts. Using a fixpoint evaluation of P, the certainty associated with atom $r(1, 2)$ will be 0, obtained only at the limit (after ω steps). This is denoted as $T_P^{\omega}(r(1, 2)) = 0$. However, we can see that $T_P^{\omega+1}(r(1, 2)) = 1$, which is different than its certainty at the limit. This indicates that the fixpoint operator T_P is not continuous, a desired property that is lost.

Another impact of adding constraints to rule bodies is on the termination and complexity properties. We can give examples of programs in the EGIB framework whose evaluation terminates, while the corresponding programs with the constraints dropped may not terminate. This is illustrated in the following example.

Consider a program in EGIB that includes the rules r_1 and r_2 defined in the above example. The fixpoint evaluation of this program does not terminate on the input set D introduced above. Now consider the EGIB program in which rule r_2 is replaced with r'_2:

$$r'_2: \; p(X, Y) \overset{1}{\longleftarrow} e(X, Z), p(Z, Y), wt(e(X, Y)) \succeq 1, \\ wt(p(X, Y)) \succeq 1; \; \langle f_{ind}, *, min \rangle$$

That is, $Q = \{r_1, r'_2\}$. It can be easily verified that the fixpoint evaluation of Q on D terminates in two iterations. This is because the first certainty constraint in r'_2 ensures that this rule never fires.

More work is required to characterize the termination and complexity of these frameworks, when recursive predicates are associated with disjunction functions, such as f_{ind}, which tend to increase indefinitely unless supplied with the top certainty value. Another interesting direction to explore is to study the containment problem in the presence of the certainty constraints along our development in the context of the parametric framework (Lakshmanan & Shiri 2001b).

An implementation of the EGIB is underway by extending our implementation (Shiri & Zheng 2004) of a fragment of the parametric framework. Query processing and unification here are more complicated due to the presence of the certainty constraints in the rule bodies.

Acknowledgements

The author would like to thank Dr. Laks Lakshmanan and anonymous reviewers for helpful discussions and valuable comments.

References

Ceri S.; Gottlob G.; and Tanca L. 1989. What you always wanted to know about datalog (and never dared to ask). *IEEE Transactions on Knowledge and Data Engineering* 1(1).

Dubois Didier; Lang Jérôme; and Prade Henri. 1991. Towards possibilistic logic programming. In *Proc. 8th Intl. Conf. on Logic Programming*, 581–596.

Fitting M.C. 1988. Logic programming on a topological bilattice. *Fundamenta Informaticae* 11:209–218.

Fitting M.C. 1991. Bilattices and the semantics of logic programming. *Journal of Logic Programming* 11:91–116.

Halpern, J.Y. 1990. An analysis of first-order logics of probability. *Journal of AI* 46:311–350.

Kifer, M., and Li, A. 1988. On the semantics of rule-based expert systems with uncertainty. In Gyssens, M.; Paradaens, J.; and van Gucht, D., eds., *2nd Intl. Conf. on Database Theory*, 102–117. LNCS-326.

Kifer M., and Subrahmanian V.S. 1992. Theory of generalized annotated logic programming and its applications. *Journal of Logic Programming* 12:335–367.

Lakshmanan, Laks V.S., and Sadri, F. 1994. Probabilistic deductive databases. In *Proc. Intl. Logic Programming Symposium*, 254–268. Ithaca, NY, MIT Press.

Lakshmanan Laks V.S., and Sadri F. 1994. Modeling uncertainty in deductive databases. In *Proc. Intl. Conf. on Database Expert Systems and Applications (DEXA '94).* Athens, Greece, LNCS-856.

Lakshmanan, Laks V.S., and Shiri, Nematollaah. 1996. A parametric approach to deductive databases with uncertainty. In *Proc. Intl. Workshop on Logic in Databases (LID'96)*, 61–81. Italy, LNCS-1154.

Lakshmanan, Laks V.S., and Shiri, Nematollaah. 2001a. Logic programming and deductive databases with uncertainty: A survey. In *Encyclopedia of Computer Science and Technology*, Volume 45. Marcel Dekker, Inc. New York. 155–176.

Lakshmanan Laks V.S., and Shiri Nematollaah. 2001b. A parametric approach to deductive databases with uncertainty. *IEEE Transactions on Knowledge and Data Engineering* 13(4):554–570.

Lakshmanan, Laks V.S. 1994. An epistemic foundation for logic programming with uncertainty. In *Proc. 14th Conf. on the Foundations of Software Technology and Theoretical Computer Science (FST and TCS'94).* LNCS-880.

Lloyd J. W. 1987. *Foundations of Logic Programming.* Springer-Verlag, second edition.

Ng R.T., and Subrahmanian V.S. 1992. Probabilistic logic programming. *Information and Computation* 101(2):150–201.

Ng R.T., and Subrahmanian V.S. 1993. A semantical framework for supporting subjective and conditional probabilities in deductive databases. *Automated Reasoning* 10(2):191–235.

Shapiro E. 1983. Logic programs with uncertainties: a tool for implementing expert systems. In *Proc. IJCAI'83*, 529–532. William Kaufmann.

Shiri, N., and Zheng, Z. H. 2004. Challenges in fixpoint computation with multisets. In *Proc. 3rd Intl Symp. Foundations of Information and Knowledge Systems (FoIKS '04), Vienna, Austria, February 17–20.*

Silberschatz Avi; Stonebraker Michael; and Ullman J.D. 1991. Database systems: Achievements and opportunities. *Communications of the ACM* 34:110–120.

Subrahmanian V.S. 1987. On the semantics of quantitative logic programs. In *Proc. 4th IEEE Symposium on Logic Programming*, 173–182.

van Emden M.H. 1986. Quantitative deduction and its fixpoint theory. *Journal of Logic Programming* 4(1):37–53.