On the Semantics of Rule-Based Expert Systems with Uncertainty

Michael Kifer and Al Li
Department of Computer Science
SUNY at Stony Brook, NY 11794, U.S.A.

ABSTRACT

We present a formal semantics for rule-based systems with uncertainty (this field has also become known as "qualitative logic programming"). Unlike previous works, our framework is general enough to accommodate most of the known schemes of reasoning with uncertainty found in the existing expert systems. We provide a rigorous treatment of the issues of evidential independence, and study its impact on the semantics. To the best of our knowledge, this issue has not been addressed before in the literature on qualitative logic programming. In expert systems evidential independence received only an ad hoc treatment, while the approaches found in the theory of evidential reasoning are feasible only in small scale systems. We discuss the problem of query optimization and, as a first step, present a quantitative semantics for query evaluation. An algorithm for generalization is a method well-known in deductive databases. Treatment of negation and conflicting evidence based on, so called, support logic is given in the last part of the paper, where we extend the semantics of stratified programs to deal with uncertainty.

1. Introduction

In recent years rule-based systems became popular tools for developing a wide range of knowledge-based applications [6, 20]. By supporting the very concepts of a rule, forward/backward chaining, certainty factors, etc., these systems are of a great help to the application developer. Despite that, building expert systems remains a largely an ad hoc and a very labor-intensive activity [13, 19, 20].

Expert systems shells are environments built around rule-based languages for programming knowledge based systems. These languages normally require the users to be aware of such low-level details as backtracking, cuts, and even whether rules are to be applied in forward or backward chaining. As programming languages, these systems inherit most of the disadvantages of Prolog [11], aggravating them even further by adding new features intended to cope with knowledge representation idiosyncrasies. For instance, as in Prolog, queries to expert systems are not guaranteed to terminate and/or retrieve all answers, the notion of query/program optimization is virtually nonexistent, and computational efficiency is almost solely a user’s responsibility. Many expert system languages allow rules to be fired both forwards and backwards, making it likely that some answers to a query will be lost. All this renders expert system development a very time-consuming and error-prone task.

Our contention is that this situation can be improved dramatically by utilizing the recent achievements in the deductive database research (e.g., [7, 15, 23, 38]). We believe that expert system programming can be made more declarative, thereby relieving the developer from the burden of specifying most of the nitty-gritty details of the control strategy. This burden should be taken over by the system which will use various query optimization techniques to achieve desirable response time.

This is not to say that the user necessarily has to relinquish all his control over program execution. Some query optimization strategies permit users to specify certain kinds of control information (e.g., *sideways propagation graphs* [23], or *sideways information passing strategy* [7]). However, it is our thesis that the purpose of control information should be improving the efficiency of a program, not changing its semantics (as is currently the case in rule-based systems, or languages such as Prolog). Thus, it should be possible to add or delete control information at any time without affecting correctness of a program. As we commented earlier, none of the commercial or experimental expert system shells we are aware of attempt any serious query optimization. Instead, they supply numerous ways of specifying control information, heavily relying on user’s experience in the art of what became known as "knowledge engineering".

In many problem domains (medical, military, etc.) expert systems have to deal with pieces of knowledge which are uncertain or incomplete. A well known example of such system is MYCIN [8]. Not surprisingly, the aforesaid problems become only worse when users have to face uncertain data or imprecise reasoning. In this paper we concentrate on such systems, making first couple of steps towards our declared goal of rendering expert systems programming a less demanding enterprise.

First, we propose a declarative semantics for general rule-based systems which utilize certainty factors as a means of representing incomplete or imprecise knowledge. At this point we would like to stress that the theory of evidential, probabilistic, or fuzzy reasoning is not our concern here. That is, we do not address the issue of how different evidences for the same fact can be combined together to yield a new (possibly stronger) evidence. Surveys of this field can be found in [6: Chapters 13-14: Chapter 7, 22, 27]. Instead, a model-theoretic semantics is described for a rather large class of theories of evidential reasoning, which is specified by a number of requirements to how strength of causal links is computed. In particular, all the theories considered in [27] fall under this category. Neither do we want to get into a discussion of the controversial issue of where do the certainty numbers come from. Our goal here is more modest. It consists in providing formal grounds for a class of practical problems which so far received only an ad hoc treatment.

Second, we define a fixpoint semantics for programs with uncertainty, and show that it coincides with the model-theoretic semantics. The fixpoint semantics is especially important to us, because, as in the database theory, it can become a basis for an efficient implementation. Although on the surface the proposed fixpoint semantics seems quite similar to that of Horn programs [39], a number of difficult problems arise concerning the computation of certainties of answers to the queries. We then propose ways of solving that problem.

In the rest of this section we survey the relevant work on declarative semantics of logic programs with uncertainty. Lee [24] describes a model-theoretic semantics and the resolution principle for one of the fuzzy logics, called K-standard sequence logic [12]. Essentially the same semantics is described in [30], although in the latter paper the formalism of Post algebras is used instead of models.

van Emden [27] also deals with K-standard sequence logic, providing both model-theoretic and fixpoint semantics for that special case of inferenceing, which he calls quantitative reasoning. Shapiro [32] considers essentially the same logic, although his framework for computing certainties is more general than in [37]. He describes a model-theoretic semantics and discusses some implementational issues, but does not provide the fixpoint semantics.

Subrahmanyan [34] extends the work of van Emden [27] by allowing negative information to be represented explicitly. For instance, in this extended logic it is perfectly acceptable for both p and ~p to be strong evidences for a fact. Unfortunately, the solutions proposed in [34] is somewhat counterintuitive. For instance, if p is known with a high degree of certainty while the certainty of ~p is low, then, according to [34], this should result in a contradiction - an unlikely course of action in most problem domains (cf. MYCIN [8]). We propose a different solution based on *support logic* [4, 6].

Work has also been reported on fuzzy relational calculus [39, 40]. These studies have also concentrated on the aforesaid K-standard sequence logic. In addition, being based on first-order (fuzzy) calculus, no recursively defined predicates can be allowed in these query languages.

**Example 1.** Consider the following program:

1. close (X, Y) →
2. edge (X, Y) →
3. line (Line(X, Y, z) → edge (Edge(X, Y), z)
4. close (X, Y), alpha * beta → close (X, Z), close (Z, Y), beta
5. line (Line(X, Y, z) * beta → edge (Edge(X, Y), z), line (Line(X, Y), z) * beta, aligned (Ell, X, Y, z)
6. edge (Edge(X, Y) * beta → line (Line(X, Y), z) * alpha, close (X, Y), beta

The fourth statement says that any two points are "close" to each other if there is an intermediate point Z which is close to both X and Y. Statements 3, 5, and 6 describe some simplified version of the line detection procedure in image recognition. In addition, there is a database of facts about close points and simple lines in the image. Variables alpha, beta, and gamma denote the certainty of the assertions they are attached to. In this example certainties range from 0 (unknown) to 1 (known to be true).

Example 1 highlights one drawback of the previous approaches to quantitative reasoning. For instance, [34, 37] do not allow such relatively complex certainty functions as those used in the example. A closer look at [33] reveals that the allowed certainty functions are required to have a (single) set-valued argument. This entails that such functions should not depend on the order of elements in the set argument, i.e., that they should be commutative. On the other hand, the certainty functions in rules 5 and 6
do not have this property.

Another weakness of the aforementioned approaches is that they all use K-standard fuzzy logic. In this logic, given a pair of evidences of strength α and β for the same fact p, the combined evidence to is assumed to be of strength max{α,β}. This may be inappropriate in some problem domains. For instance, MYCIN [8] uses, so called, stochastic logic [12] in which the combined evidence would be α + β - αβ. All this suggests considering more general theories of quantitative reasoning. Our approach is not restricted to any of the above logics. We only assume that evidences are combined subject to certain natural restrictions, which are general enough to accommodate several popular methods. In this general situation, new difficulties arise, which are mostly due to the problem of independence of different evidences supporting the same fact. We show in Section 4.1 that in the special situation considered in [32, 34, 37] this problem does not arise, which makes it, in this sense, the easiest case of quantitative reasoning. However, as mentioned earlier, this case is not general enough to accommodate the most popular reasoning systems, such as MYCIN.

We describe the general semantics in Section 3, and show the impact of evidential independence on the meaning of a program. The need to verify independence of evidences makes query evaluation more involved. In Section 4 we discuss these difficulties, and generalize the well-known semi-naive query evaluation algorithm [5, 18, 35] to deal with general quantitative (Horn) logic programs. The semantics and the semi-naive algorithm described in Sections 3 and 4 are decoupled from the choice of any specific notion of evidential independence, adding another degree of generality to our approach. Of course, the cost of query evaluation does depend on this choice, which is also discussed in Section 4.2.

It is often argued that associating a single certainty factor with each fact in the database is not enough for many applications. For instance, in MYCIN each fact has a measure of belief and a measure of disbelief. This technique is often used to cope with contradictory information commonly arising in medical, military, and other domains. A promising approach utilizing this idea is based on, so called, support logic. Recently several calculi for computing combined evidences in such logics have been proposed [4, 5, 16]. However, no formal logical semantics exists for such logics, and their implementations still share most of the drawbacks of the rule-based languages mentioned earlier. This lack of semantics prevents any systematic study of optimisation in support logic systems. In Section 5 we outline a declarative semantics for such systems.

2. Preliminaries

We assume some knowledge of Logic Programming as a level of the first few chapters of [28]. Certain familiarity with database terminology and query optimization (e.g., [7, 23, 33]) is also required.

Literals in our logic are constructs of the form p, r where p is a literal in the usual predicate calculus, and r is a term representing certainty information about p (called certainty term of p). When confusion is possible, we will call the usual first-order literals, the d-literals (for data literals). D-literals are of the form p[x1, ..., xn] or ¬p[x1, ..., xn] where p is a predicate symbol and x1, ..., xn are variables or constants. We thus do not allow function symbols in such literals. Ground d-literals will be called d-facts.

We also assume a finite collection of interpreted certainty functions (usually some arithmetic functions). A k-ary such function maps [0, 1]k into [0, 1]. Certainty terms are built in a usual way out of these functions, certainty variables (which are different from the usual logic variables), and certainty constants which will be taken from the domain of real numbers in the interval [0, 1].

D-literals annotated with certainty 1 are considered as definitely true facts. Certainty values between 0 and 1 mean that there is some inconclusive evidence that the fact is true, while the facts with certainty 0 have no supporting evidence whatsoever. This does not imply anything about the falseness of the fact, contrary to the assumption made, say, in [34, 37]. Dealing with negative information, and representing falsehood of facts is postponed until Section 5.

A (Horn) rule is a statement of the form

p1 → q1; ..., ; pn → qn

(1)

where p1, ..., pn are positive d-literals, and q1, ..., qn are certainty terms. All the variables in (1), including the certainty variables, are assumed to be universally quantified outside the clause. We also assume that variables in the head of the rule (including certainty variables) also appear in the body of that rule. In practice there does not seem to be many applications for the full generality of rule (1); in this paper we will be considering only the rules of the following simpler form:

p(x1, ..., xk) → q1; ..., ; qn

(2)

where q1, ..., qn are certainty variables or constants and f is a certainty function associated with this rule. We will be also talking about d-versions, or d-instances of rules, etc., meaning the rules (or instances thereof) with the certainty information deleted.

We assume that bodies of the rules are nonempty. Particularly, unlike in Prolog, facts are not viewed just as rules with empty bodies: they are not rules according to our definition at all. All predicate symbols are partitioned into two categories. The first category consists of the base predicates. These predicates can appear only in the bodies of rules, and their extensions (sets of facts) are known in advance. The second category consists of the derived predicates. Extensions of these predicates are initially empty, and the corresponding facts are derived by the rules. Derived predicates can appear in the heads of the rules as well as their bodies. This assumption is common in deductive databases, and is known to be equivalent to the general case (used in Logic Programming).

An expert system is a combination of a Horn program P (i.e., a collection of rules) and of a set of facts, D, for the base predicates of P. In keeping with the database tradition, we will sometimes call P the intentional database (IDB) and D the extensional database (EDB). For simplicity we assume that each ground d-fact appears in D at most once, i.e., for each p(x) in D, there is at most one fact of the form [p(x)] in D. If no [p(x)] appears in D for some p(x), then we assume that p(x) ∈ D.

As suggested by the form of the rules, certainty function associated with a rule is a measure of strength of a link between the rule premises and the consequent. We impose the following natural restrictions on these functions:

1. Monotonicity: if r1 ≤ r2 ≤ r3 then r1 ≤ r3. In other terms this means that higher certainty of premises should yield a higher certainty of the consequent.

2. Boundedness: r1 ≤ r2 for i = 1, ..., n. This states that conclusion of a rule can be only as good as its premises.

3. Continuity: for all ε there exists δ > 0 such that if r1 = r2 then r1 = r2. This requires that small changes of the input do not lead to large changes of the output.

In expert systems, strengths of evidences obtained from different sources all supporting the same fact are combined to determine strength of the overall support for the fact. In different problem domains and, perhaps, for different types of facts, combination methods may be different. Furthermore, different researchers are fond of different theories of evidential reasoning. In order to be independent from any such theory, we postulate that associated with each predicate symbol, p, there is a unique combination function, F p, used to calculate strength of combined evidences for p. Combination functions accept a single multiset-valued argument, which implies that the combined strength of evidences for a fact is in fact independent from the order in which these evidences are obtained. Some researchers have argued that it should not be that way, i.e., evidences obtained prior to some other evidences may increase significance of the latter evidences [21]. Acknowledging controversy of this issue, we note, however, that the majority of the known theories of evidential reasoning (see [22]) satisfy the commutativity requirement.

We thus impose the following natural restrictions on combination functions:

1. Commutativity: Each combination function has a single argument which is a multiset of certainty factors (i.e., multiset of values in the range [0, 1]).

2. Monotonicity: If p(S) ≤ p(S') then F p(S) ≤ F p(S'). The order on multisets is defined in the usual way: S ≤ S' if there is a 1-1 mapping t from S into S' s.t. for every x ∈ S, x ≤ t(x).

We postpone the discussion of conflicting evidences until Section 5.

1 We postpone the discussion of conflicting evidences until Section 5.

2 In a multiset, the same element can have several occurrences.
condition ensures that stronger evidence yields stronger overall support for a fact.

3. Association: \( F(S \cup S') = F(S) F(S') \). Here \( \cup \) is a union of multisets which retains duplicate occurrences of the same element. According to this requirement, the order in which evidence is combined is immaterial. In order to obtain the overall strength of support for a fact there is no need to wait until all evidence for that fact is obtained. Instead, one can evaluate the support incrementally. Strictly speaking, this requirement is not needed in order to describe the semantics of programs. However, without associativity, evidential support cannot be evaluated incrementally, burying any hope for efficient query evaluation. We will use this property in Section 4.2.

4. No-Information Rule: \( F(S \cup \{0\}) = F(S) \). That is, a non-evidence cannot change the overall support.

5. Correctness: \( F(\{\alpha\}) = \alpha \). This means that the combined support provided by a single evidence is exactly as strong as the evidence itself.

6. Continuity: \( F(\_\_\_\_) \) is continuous w.r.t. the order \( \geq \) on multisets defined in (2). Continuity of combination functions is needed for the same reason as in the case of certainty functions.

The following useful properties of combination functions can be derived from the aforementioned ones:

7. Support enhancement: \( F(S) \geq \alpha \) for every \( \alpha \in S \). This follows directly from (2) and (5). It means that every new evidence is potentially useful, because it may increase the overall support for a fact.

8. Unconditional Support: \( F(S \cup \{1\}) = 1 \). This property follows form (7) it postulates that if an evidences establishes a fact beyond any doubt then the fact is unconditionally true and the rest of the evidence do not matter.

Our next step will be defining interpretations and models in which we follow the outline of [37]. For our purposes it suffices to consider only Herbrand interpretations, although general interpretations can be also defined. Given an expert system \( E \subseteq P \cup D \), the domain \( D \subseteq P \) of any Herbrand interpretation \( E \) is a collection of all the constants mentioned in \( E \). A Herbrand base of \( E \) is a collection of all ground facts of the form \( p(a_1, \ldots, a_n) \), where \( p \) is an n-ary predicate symbol in \( P \), \( a_i \) all belong to \( D \), and \( \alpha \) is a certainty factor of \( p(a_1, \ldots, a_n) \). A Herbrand interpretation \( E \) of \( E \) is a subset of \( D \) and \( \alpha \in [0, 1] \) is a certainty factor of \( p(a_1, \ldots, a_n) \). A Herbrand interpretation \( E \) of \( E \) is consistent if and only if the rule \( p \) is true in \( E \) if \( \alpha \geq \beta \). The fact \( p(a_1, \ldots, a_n) \) is then true under \( E \) if there is \( p(a_1, \ldots, a_n) \in E \). This postulate that whenever a fact is known with a higher certainty, \( \beta \), it is also known with any other certainty which is smaller than \( \beta \).

A ground rule \( p(x_1, \ldots, x_n) \rightarrow q(x_1, \ldots, x_n) \) is true in \( E \) if and only if the head of the rule, \( q(x_1, \ldots, x_n) \), is also true in \( E \). A (nonground) rule \( p(x_1, \ldots, x_n) \rightarrow q(x_1, \ldots, x_n) \) is true in \( E \) if all its ground instances are true in \( E \). In our framework, ground rules are used as evidence to the facts in their heads. However, not every true rule can serve as an evidence. We say that the above rule supports \( \alpha \) of \( p(x_1, \ldots, x_n) \) in \( E \) if all its literals (head and body) are true in \( E \). We will also refer to such rules as evidences.

A program, \( P \), is true if all its rules are true, and, in addition, the following combination requirement is satisfied:

For every set of independent (explained later) ground instances of the rules of \( P \) with the same head

\[
\begin{align*}
F(p) &\leq F(p) F(p) \\
F(p) &\leq F(p) F(p) \\
F(p) &\leq F(p) F(p) 
\end{align*}
\]

such that each individual rule supports its head literal in \( E \), the literal \( F(p) F(p)(\{f(a_1, \ldots, a_n), \ldots, f(a_1, \ldots, a_n)\}) \) should be true in \( E \). Notice that \( f_1 \) and \( f_2 \) would be identical certainty functions if the i-th and the j-th rules above are instances of the same rule of \( P \); they may be different otherwise. 8

8 Observe that, since base predicates cannot appear in the rule heads, \( F(p) \) cannot be an EDB fact. Hence, no EDB fact used with the above evidence in \( F(p) \).
On the other hand, we could compute the certainty of $q(s, b)$ in the intended model using the following argument. Let the certainty of this fact after the $n$-th iteration be $\alpha_n$ and its certainty after the $n+1$-st iteration be $\alpha_{n+1}$. It is easy to see that these two numbers are related by the following equation: $\alpha_{n+1} = 0.5 \alpha_n + 0.5 \alpha_n$. After the simplification, we obtain $\alpha_{n+1} = 0.5 + 0.2 \alpha_n$. In the limit we get $\alpha = 0.6 + 0.2 \alpha$, where $\alpha$ is the certainty of $q(s, b)$ in the intended model. Solving this equation yields 0.75, which is the desired certainty.

The problem illustrated in Example 2 stems from the possibility of a cyclic-inferencing of the same fact, each time with a slightly higher evidence. We have also shown that this iterative process can be bypassed, which allows computing certainties exactly. In the next subsection we will generalize this argument to arbitrary programs. Another interesting question is finding out when the problem of Example 2 does not arise. In the rest of this subsection we deal with this issue.

Let us say that a program $P$ has finite termination property if after a certain step $n$, $T_E^k(\emptyset) = T_E^k(\emptyset)$ for every $E = P \cup D$. Obviously, the problem spotted earlier may happen to a program if and only if the program does not have finite termination property. All nonrecursive programs, or those without uncertainties, obviously possess that property. A large important class of such programs was considered in [33, 34, 37]. This class can be characterized by the fact that all recursive predicates have the same combination function $\max(\_)$.

That is, given a number of evidences for a fact, $p$, the combined support provided by these evidences equals the support provided by the strongest of the evidences.

**Theorem 1.** If all combination functions for the recursive predicates of $P$ are $\max(\_)$-functions, then $P$ has a finite termination property.

### 4.2. A Quantitative Semi-Naïve Algorithm

In this subsection we generalize the ideas of the semi-naïve bottom-up query evaluation [2, 6, 18] to the case of quantitative logic programming. The main difficulty here stems from the need in computing certainty factors and dealing with evidential independence.

As illustrated by Example 2, certainty factors cannot be computed just by accumulating new certainties and combining them with the old ones. The discussion following this example suggests that recurrent equations involving certainty factors might be helpful.

Accordingly, we associate an equation with each d-fact. At the very beginning, these equations are all of the form $r_{ij}(p, \emptyset) = 0$ for each d-fact $p(\emptyset)$, where $r_{ij}(p, \emptyset)$ is a special certainty variable associated with $p(\emptyset)$. This initial equation says that at the beginning no evidence is available for the derived facts, and therefore, their certainty is 0. As the evaluation proceeds, the equations are updated to reflect new evidence obtained during the evaluation process. In general, we end up with recursive equations (like the ones obtained in Example 2), which have to be solved to find the desired certainties. We discuss ways of solving these equations later in this section.

First we present a basic quantitative semi-naïve algorithm assuming the simplest (independence-1) notion of evidential independence. Then we show how to extend this algorithm to accommodate other notions of independence. We assume the following conventions.

Besides the unique certainty variable $r_{ij}(p, \emptyset)$ associated with each fact $p(\emptyset)$, there is an equation $r_{ij}(p, \emptyset) = e(p, \emptyset)$. Here $e(p, \emptyset)$ is an expression involving certainty variables associated with other facts, as well as, possibly, $r_{ij}(p, \emptyset)$ itself. As explained earlier, initially $e(p, \emptyset) = 0$ for all facts which are not the EDB D (i.e., for the derived facts). For the EDB facts from D, $e(p, \emptyset)$ simply equals the certainty of $p(\emptyset)$ in D. As in the regular semi-naïve algorithm (e.g., [2]), we keep track of the ground d-facts obtained in the current iteration (which are kept in the set NOW). Facts derived during the previous iteration are kept in the set LAST, and those obtained even prior to that are saved in the set OLD. The algorithm is depicted in Figure 1.

It is easy to see that because of the associativity of combination functions, at the end of the run of the algorithm of Figure 1, the equation obtained for an arbitrary fact, say $p(\emptyset)$, is equivalent to $r_{ij}(p, \emptyset) = f_{ij}(f_{i1}(f_{i2}(\ldots f_{in}(r_{i1}, r_{i2}, \ldots, r_{in}, \emptyset))))$, where $R_1, \ldots, R_n$ are all the ground d-instances of the rules in $P$ with the same head $p(\emptyset)$ used in the

\footnote{In fact, this class is slightly more general than the one considered in [33, 34, 37].}
OLD := ϕ
LAST := D (the EDB)
repeat
NOW := Ø
for each rule R ∈ P do
(*) Let R be f(X) ≐ (a₁, ..., αₙ) → (f₁(X₁), ..., fₙ(Xₙ)) = ϕ
Suppose there are ground d-facts g₁(δ₁₁), ..., gₙ(δₙ) ∈ OLD ∪ LAST such that:
(1) f(δ₁₁) ≡ (f₁(δ₁₁), ..., fₙ(δₙ)) is a ground d-instance of R,
(2) at least one of the gᵢ(δᵢ₁) is in LAST
replace r_i(δ₁₁) = exp_r_i(δ₁₁) by r_i(δ₁₁) = F_i(f₁(δ₁₁), ..., fₙ(δₙ), exp_r_i(δ₁₁))
if r_i(δ₁₁) ∈ OLD ∪ LAST then NOW := NOW ∪ {r(δ₁₁)}
end (* for each *)
OLD := OLD ∪ LAST
LAST := NOW
until NOW = Ø

Figure 1 - The Basic Quantitative Semi-Naive Algorithm

Algorithm: Jₜ_i are certainty functions associated with these rules, and the d-version of each Rᵢ is p(δ₁₁) = r_i(δ₁₁), ..., r_i(δₙ)

Equations associated with the EDB-facts are much simpler: r_t(δ₁₁) = α, where α is the certainty of r(δ₁₁) in D. This is because base predicates cannot appear in the heads of the rules of P, and therefore certainties of the EDB-facts do not change. In summary, we obtain a system of equations

\[
\begin{align*}
{\gamma(\delta_{11})} = {\gamma(f(\delta_{11})} ... {\gamma(f(n)) \cdots}) = \{p(\delta_{11})\} \text{FACTS} \\
\end{align*}
\]

where FACTS is the set of d-facts derived by the semi-naive algorithm plus the facts in D; \(\gamma(f(\delta_{11}))\) stands for the composition of functions in (5). Since all combination and certainty functions are monotonically increasing, all \(\gamma(f(\delta_{11}))\) are too. These functions are also upper-bounded by 1. The equations in (6) can be also viewed as a definition of an operator, \(\Phi\), on \([0,1]^n\rightarrow[0,1]^n\), where \(\delta = |D|\) is the number of facts in the EDB. Because of the monotonicity and boundedness, this operator has the least fixed point in \([0,1]^n\), which obviously is a solution to the equational system (8).

THEOREM 2. Consider the semantics under the independence-1 assumption. Then certainty factors for all d-facts in the intended model of E are given by the least fixed point solution of the \(\Phi\) corresponding to the equational system (8).

It is now easy to generalize the basic algorithm to handle any notion of evidential independence other than independence-1. Let IND denote some such notion. Recall that in (4) each certainty term \(F_i(\cdots)\) corresponds to a unique ground rule used as an evidence for \(p(\delta_{11})\). Denote the collection of all such rules by S. Let S₁, ..., Sₙ be all the maximal sets of independent (according to IND) rules from S, and let T₁, ..., Tₙ be the corresponding sets of certainty terms. Then, in order to replace independence-1 by IND in the basic semi-naive algorithm, one only has to transform every equation (4) produced by that algorithm into the following form:

\[
\begin{align*}
\gamma(\delta_{11}) = \max \{F_1(\delta_{11}), ..., F_n(\delta_{11})\}
\end{align*}
\]

We will then obtain a different equational system similar to (5):

\[
\{r_t(\delta_{11}) = \max \{\psi_1(r_t(\delta_{11}), ..., \psi_n(r_t(\delta_{11})), ..., \psi_k(r_t(\delta_{11})) \cdots} \} \text{FACTS} \}
\]

Associated with system (7) is another operator, \(\Psi_{IND}\), mapping \([0,1]^{n\times k}\rightarrow[0,1]^{n\times k}\), which also possesses the monotonicity and boundedness properties. In particular, as the earlier system (5), it has the least fixed point solution.

THEOREM 3. Assume an arbitrary definition of evidence independence IND. Suppose also that system (7) is obtained from (5) by selecting maximal sets \(S_1, ..., S_k\) of independent rules according to IND. Then certainty factors for all d-facts in the intended model of E are given by the least fixed point solution of the \(\Phi_{IND}\) corresponding to the equational system (7).

Example 5. Consider the following system: EDB D = \{a₁, a₂, a₃, a₄, a₅, a₆\}, facts 111

\[
p(\{X, Y\} : a₁, a₂, a₃, a₄, a₅, a₆) \text{facts}
\]

\[
\begin{align*}
F_1(\{X, Y\} : a₁, a₂, a₃, a₄, a₅, a₆) & = \frac{1}{1 + 1} \text{fact}
F_2(\{X, Y\} : a₁, a₂, a₃, a₄, a₅, a₆) & = \frac{1}{1 + 1} \text{fact}
F_3(\{X, Y\} : a₁, a₂, a₃, a₄, a₅, a₆) & = \frac{1}{1 + 1} \text{fact}
\end{align*}
\]

Let us trace the execution of the algorithm of Figure 1 assuming the independence-2 notion of evidential independence. Initially OLD = NOW = Ø, and LAST = D. At the first iteration we derive \(p(\{a₁, a₂\}) = 1\) using the first rule. We thus have \(p(\{a₁, a₂\}) \in NOW\) and then OLD = NOW = LAST = \(\{p(\{a₁, a₂\})\}\).

At the next iteration rules 2 and 3 are applied yielding the fact \(p(\{a₁, a₃\}) \in NOW\) twice. This leads to creation of the following equation for \(p(\{a₁, a₃\})\): \[p(\{a₁, a₃\}) = F_1(\{a₁, a₃\}) F_2(\{a₁, a₃\}) F_3(\{a₁, a₃\}) \]

Then \(p(\{a₁, a₃\})\) moves to OLD and \(p(\{a₁, a₃\})\) to LAST. Notice that the two rules that created \(p(\{a₁, a₃\})\) were not independent according to independence-2.

During the third iteration, rules 2 and 3 are applied once again, but with slightly different data (using \(\{a₁, a₄\}\) instead of \(\{a₁, a₃\}\)). This yields the fact \(p(\{a₁, a₄\})\) two more times, updating the equation for that fact to \[p(\{a₁, a₄\}) = F_1(\{a₁, a₄\}) F_2(\{a₁, a₄\}) F_3(\{a₁, a₄\}) \]

At this point the derivation terminates since no new ground rule can be applied. However, not all evidences do provide independent evidences. To take this into account we transform the above equation into the form (6):

\[
\begin{align*}
r(\delta_{11}) = \max \{F_1(\delta_{11}), ..., F_n(\delta_{11})\}
\end{align*}
\]

Values of all variables except \(r(\delta_{11})\) and \(r(\delta_{11})\) are known from the initial equations \(r(\delta_{11}) = 0.2, r(\delta_{11}) = 0.3, r(\delta_{11}) = 0.7, r(\delta_{11}) = 0.1\). Substituting, we obtain a set of linear equations, which can be easily solved, yielding certainties of all facts involved in the query evaluation.

Unfortunately, the situation is not always that simple as the above example might suggest. First, under a more sophisticated independence requirement (than independence-1), we may have to solve the equational system (7) instead of (5). Solving such a system usually amounts to breaking up the max(\cdots) functions in (7) and creating a number of simpler equational sub-systems of the form (5). These systems are then solved independently. The solution of (7) is the maximum of solutions to the sub-systems. The problem here is that breaking up (7) may create an exponential number of sub-systems to solve. Second, the number of equations in (5) or (7) may be as large as the number of relevant facts used in the query evaluation.

Another, perhaps more serious, problem is that, in general, functions \(\phi\) or \(\psi\) in (5) and (7) may turn out to be polynomials of an arbitrarily high degree, even though the certainty and combination functions are not more than quadratic. The danger here is that variables \(r_i\) involved in these equations may be mutually dependent on each other. In general, such equations can be solved only approximately. One obvious method is finding the least fixpoint solution to (5) by iterating over the set of equations (5) [i.e., by iteratively applying \(\Phi\) to the vector \(<0,0,0>\) of initial values for \(r_i\)] each time getting better approximations to the actual solutions. For certain types of combination and certainty functions it is

\footnote{Here we assume that \(F_i\) is the M Rule combination function.}
possible to find reliable error estimates for the iterative method, but we are unaware of any such method for the general case.

On the bright side, note that if the rules are non-recursive then the iterative method always yields exact solutions. In this case, solving equations (5) or (7) can be done by simple substitution. This does not cause much overhead, since in this case the equations in (7) are non-recursive, and computing certainty factors can be done on-the-fly at the time of rule application. There is no need in this case in splitting the equalitationary system (7) in order to eliminate max(...), since the arguments to max(...), will be fully evaluated. This argument can be extended to recursive rules as follows.

Let \( E = P \cup D \) be an expert system. Consider the set \( H \) of all d-facts actually used in the query evaluation. Construct the fact dependence graph \( G(H, P) \) on \( H \) w.r.t. \( P \) as follows. As an arc from a fact \( q \) to \( p \) is drawn if and only if there is a ground instance of a rule in \( P \) a.t. its head is \( p \), one of its premises is \( q \), and all other premises are in \( H \).

Theorem 4. If the fact dependence graph, \( G(H, P) \), is acyclic then the iterative method of finding solutions to equalitational systems (5) and (7) always terminates. The number of iterations over the equations is the same as the number of rule applications in the semi-naive equalitationary algorithm.

Thus, if there are no cyclic facts (note: recursive predicates are allowed), then the qualitative semi-naive algorithm can be used without much overhead compared to the regular semi-naive algorithm. We conjecture that in real-life applications there are rarely a need in cyclic facts. Even when they are inherently necessary, we believe that they are small in number, which in contrast the iterative or other approximate techniques may be acceptable.

There are several possible optimizations to the algorithm presented in Figure 1 which take advantage of Theorem 4. The idea is to evaluate the equations on-the-fly at the rule application time. However, if certain fact is determined to be recursive, we switch to another mode (for that fact) and start generating recursive equations as in Figure 1. Technical details of this optimization are rather tedious and are omitted.

5. Semantics of Conflicting Evidences and Negation

In this section we extend our framework to allow negative literals to appear in rule premises as well as the consequent. The latter is particularly useful when dealing with incomplete knowledge, in which case different evidences may contradict each other. We use the term support logic as a generic name for a number of related approaches to coping with such situations. Although this logic suffers from some philosophical problems (as is the case with many other non-standard logics), it was proven practically useful, and we know of at least one commercial implementation [8]. However, we are unaware of any formal semantics for such logics. We attempt to rectify this drawback by extending the semantics described in Sections 2 and 3. Because of the space limitation the results of this section are rather sketchy.

In support logic, each fact, \( p \), has a measure of belief, \( \mu_B(p) \), and a measure of disbelief, \( \mu_D(p) \). The latter is the measure of belief in \( \neg p \). It is consistent to deal with 1 - MD instead of \( M \), viewing this new quantity as an upper limit on the belief in \( p \). Thus, each fact, \( p \), is assigned an interval \([\mu_L(p), \mu_H(p)]\) in which the strength of belief in \( p \) is somewhere in-between low and high. The difference \( \mu_H(p) - \mu_L(p) \) is the knowledge gap about \( p \). Under this convention it may seem unnecessary to consider negative literals, since the interval assigned to \( p \) is \([\mu_L(p), \mu_H(p)]\), which is uniquely determined by the interval for \( p \). However, body occurrences of \( \{l, A\} \) and \( \{L, A, L\} \) are treated differently, since certainty functions use interval \([l, A]\) as an argument in the former case, and \([L, A, L]\) in the latter. Besides, it is often convenient to use negative literals explicitly, even though they might be replaced by their positive counterparts.

Literals in support logic are of the form \( p \in \mu_B \) or \( p \in \mu_D \). We use measure of belief is denoted as \( \mu_B(p) \), and measure of disbelief is denoted as \( \mu_D(p) \). Certainty terms. A negative literal, \( \neg p \in \mu_D \), should be assigned as \( \neg p \in \mu_D \), instead of \( \neg p \in \mu_B \), which will be clear from their semantics. Rules are as in Section 2 (see [2]), except for the following two differences:

(i) Certainty terms/variables are replaced by intervals of certainty terms/variables.

(ii) Negative literals can appear in rule bodies as well as their heads.

It will be convenient to introduce partial order on the certainty intervals, which will also help to see the succession between the definitions in the current and the previous sections. Thus, we write \( \mu_B(p) \leq \mu_B(q) \) if \( \mu_B(p) \leq \mu_B(q) \). The motivation here is that a bigger (w.r.t. \( \leq \)) certainty interval means that stronger positive and negative evidences are available for the associated fact. We augment the domain of all certainty intervals by adding the maximal element, \( \top \), representing all inconsistent intervals, i.e., intervals \([a, \infty) \). Thus, \( \mu_B(p) \leq \mu_B(q) \). Interval \([a, b]\), \( \mu_B(p) \leq \mu_B(q) \). Interval \([a, b]\), is obviously, the smallest certainty interval. It is also easy to see that certainty intervals form a complete lattice \((\mathbb{R}, \leq)\). A set of intervals is the smallest \((\mathbb{R}, \leq)\) containing each of the intervals in the set. The least upper bound, lub, of that collection is the largest interval contained in each of the intervals in the set if, if exists; \( \top \), is otherwise. This partial order can be naturally extended to literals so that for the same d-literal \( p \), \( \mu_B(p) \leq \mu_B(q) \) if \( \mu_B(p) \leq \mu_B(q) \). We also write \( \mu_B(p) \leq \mu_B(q) \) if \( \mu_B(p) \leq \mu_B(q) \), but \( \mu_B(p) \leq \mu_B(q) \).

Herbrand base of a system \( E = P \cup D \) is now a collection of all positive ground facts \( \{p|\mu_B(p)\} \), where \( \mu_B(p) \) is a certainty interval or \( \top \), \( p \) is a predicate symbol from \( P \), and \( \mathbb{F} \) is a vector of values from the domain \( D \). A fact \( p \in \mathbb{F} \) is called an inconsistent fact. Interpretations are, as before, subsets of the Herbrand base. Without loss of generality we may assume every d-fact, \( p \), may appear in \( \mathbb{F} \) in conjunction with at most one certainty interval. For convenience, we also assume that if \( p \) does not appear in \( \mathbb{F} \) in conjunction with any interval, then \( p \in \{0, 1, p\} \), meaning that no information is available about \( p \) and \( \neg p \) (i.e., the truth of \( p \) and \( \neg p \) is undefined). Intervals \([0, 0]\) and \([1, 1]\) mean the usual values \( false \) and \( true \), respectively.

A ground positive fact, \( \mu_B(p) \in \mathbb{F} \), is true in an interpretation, \( I \), if there is \( \mu_B(p) \in \mathbb{F} \) such that \( \mu_B(p) \leq \mu_B(I) \). A negative fact, \( \mu_B(p) \in \mathbb{F} \), is true in \( I \) if \( \mu_B(p) \leq \mu_B(I) \). Certainty (resp. combination) functions now map sequences (resp. multisets) of intervals (including \( \top \)) into the set of all certainty intervals plus \( \top \). Satisfaction of rules by \( I \) is established in the standard way, as in Section 2. The definition of satisfaction of programs by interpretations carries over from Section 2 without change, except that each certainty variable/constant \( a_i \) should be replaced by an interval \([\mu_L(a_i), \mu_H(a_i)]\) of certainty variables/constants.

Since certainty intervals can be viewed as a lattice of truth values, our semantics can be regarded as an extension of [31], although the latter is described in quite different terms. On the other hand, our notions of implication, model, and entailment are different from another related proposal [17] in which truth value is based on a lattice. Notice that the above definition of an interpretation containing inconsistent is that an interpretation containing inconsistent is that a model of an expert system. However, in our logic, inconsistent facts cannot cause much damage, since they may only affect the facts which are directly dependent on them. Thus, it is possible in our semantics to detect an inconsistency, and proceed with other inferences without getting into deep troubles. This is particularly useful in analyzing, say, intelligence information.

The partial order on interpretations and other definitions carry over from Section 2 without much change. Let \( I \) and \( J \) be a pair of interpretations of \( E \). We write \( I \leq J \) for every positive fact \( \mu_B(p) \in \mathbb{F} \) there is a fact \( \mu_B(p) \in \mathbb{F} \) such that \( \mu_B(p) \leq \mu_B(I) \). The notions of intersection and union of interpretations carry over directly with the exception that \( \leq \) replaces the usual ordering on real numbers in the interval \([0, 1]\) which was used in Section 2.

It is now easy to see that the framework of Section 2 is a special case of support logic once we replace each literal \( p \in \mathbb{F} \) of Section 2 by a support logic literal of the form \( p \in \mathbb{F} \).

Because of the negative information (either in the form of negative literals, or as disbelief measures), defining model-theoretic semantics is much more involved than it was in Section 2. As in Logic Program- ming, non-Horn programs may have no unique least model. Instead, they usually have several minimal models, and it is not always clear which one should be preferred. We will handle this situation along the lines suggested in [1, 28], developing a theory of stratified programs in the framework of support logic.

In pursuing this line we have an additional difficulty, though. The semantics of \([1]\) is a manifestation of the so-called closed world assumption (CWA) [29], in which a fact is assumed false unless there is an evidence to the contrary. However, negation is already present in our framework in the form of disbelief factors, and we do not always want to jump to a negative conclusion whenever there is a knowledge gap about some fact. Namely, if, say, \( p \in [A, B] \) is known then in response to the query \( p \in [A, B] \) we would still expect the answer \( p \in [A, B] \), not \( p \in [A, B] \) as CWA would suggest. As in Logic Programming, we take the position that the intent of jumping to a negative conclusion should be explicitly
To make things work, we assume that the certainty and combination functions satisfy all the requirements for these functions listed in Section 2, where, as before, scalar certainties are replaced by intervals of certainties, and the usual order $\leq$ on $[0,1]$ is replaced by $\leq$ on the intervals. Again, it is easy to see that the requirements in Section 2 represent a special case of the new requirements, once each scalar entity, $a$, is replaced by the interval $[a,b]$.

Although simple and intuitively appealing, these assumptions are no longer that natural as they were in Section 2. For instance, it may be desirable to assume that, as the belief in $\neg \gamma$ increases, the belief in $\gamma$ should decrease. The monotonicity assumption about the certainty and combination functions rules this possibility out. Particularly, one of its deplorable casualties is the famous Dempster-Shafer combination rule [32]. On the other hand, our theory is still useful in many problem domains. For instance, the above requirements are satisfied by the functions $\triangleright$ and $\triangleright\triangleright$ available in MYCIN. Extension of our semantics to include other important combination rules is a topic for future research.

We now focus on the new requirements that are satisfied.

Example 2. Suppose the EDB D is $[p, [3,5]]$, and consider the following pair of logically equivalent sets of rules: $P_1 = \{ \{p, 1, \neg \gamma, \gamma\}$ and $P_2 = \{ \{p, 1, \neg \gamma, \gamma, \gamma\}$. Recall that $\neg \gamma$ is designated for treatment under CWA. In $P_1$, none of the rules can be applied, and the intended model would be identical to $D$. In contrast, both rules are applicable (since $\neg \gamma, 3, 5$) and $\neg \gamma, \gamma$ are derived under CWA and we obtain $p, 3, 5$, and $p, 1$. Because of the monotonicity of combination functions, the combined evidence would be $p, 5, \gamma$, an inconsistent fact.

Notice that although the above example programs $P_1$ and $P_2$ are logically equivalent, they are interpreted differently by the CWA. However, this is similar to the corresponding situation in Deductive Databases and Logic Programming where, say $p \triangleright \neg q$ is handled differently than $p \triangleright q$.

Theorem 5. The fixedpoint computation yields the intended model of $P$.

Quantitative semi-naive algorithms of Section 4 can also be extended to handle support logic queries. This issue will be dealt with in the full version of the paper. We conclude this section with a classic example of "Flying Tweety", slightly modified to illustrate some of the issues discussed in this section, including conflicting evidences.

Example 5. Suppose the EDB contains a single fact $bird(tweety) \in \{7,9\}$, and consider the following rule:

\[ flies(tweety, x) \rightarrow \neg \text{animal}(x) \rightarrow \neg \text{abnormal}(x) \]

Applying the rule yields $\text{fies(tweety, 7, 9)}$. If, in addition, there would be an evidence that Tweety is abnormal, e.g., $\neg \text{abnormal} \text{tweedy} \in \{4,7\}$, then we can only conclude $\text{fies(tweety, 3, 8)}$, thereby decreasing our belief in Tweety’s ability to fly. On the other hand, if we were told that $\text{abnormal(tweety, 7, 1)}$ (say, because Tweety looks like a penguin), then the conclusion would be $\text{fies(tweety, 0, 9)}$, eliminating our belief in Tweety being a flying creature, while leaving some small evidence to the contrary.

Suppose now that we had another rule, $\neg \text{fies(tweety, x)} \rightarrow \text{abnormal}(x) \in \{4,9\}$, and the combination function for $\text{fies(tweety)}$ were $F_\gamma([p, [3,5], x]) = [p \neg \gamma \neg \neg \gamma, \gamma]$. Here abnormal is treated under OWA. Applying abnormal(tweety, 7, 1), we get $\text{fies(tweety)} \in \{0,3\}$ from the last rule. Combining this with $\text{fies(tweety, 0, 9)}$, which was obtained earlier, we conclude $\text{fies(tweety, 0, 27)}$, strongly suggesting that Tweety cannot fly.

Of course, these conclusions are not too surprising, since our semantics is an extension of that given in [1, 25, 28].
6. Conclusions and Future Work

We presented a model-theoretic and fixpoint semantics for rule-based expert systems with uncertainty. Our approach to the problem is much more general than the earlier works on that issue [33, 34, 37]. We have also considered some new aspects of the problem such as evidential independence, conflicting evidence, etc. Our treatment of negation accommodates both the closed and the open world assumptions.

As a first step towards query optimization in expert systems, we presented a generalization of the well-known semi-naive evaluation algorithm widely used in deductive databases [2, 6, 18]. Although some optimization methods from deductive databases (e.g., 7, 23) can be directly applied to the quasitautological case discussed in this paper, more work needs to be done in order to be able to recognize and eliminate certain-query facts from the query evaluation process.

Our treatment of conflicting evidences is not as general as one would like it to be. For instance, as noted in Section 5, Dempster-Shafer combination rule is precluded by our assumptions about monotonicity of certainty and combination functions. Extending this framework is also a topic for the future. Nevertheless, our logic is still useful for a wide range of applications. Another intriguing issue is the ability of that logic to tolerate inconsistent information, yet be able to make sensible inferences. This feature is very important in certain problem domains (e.g., military, medicine).

Additional cases of evidential independence need to be studied. The two independence criteria presented in Section 2 do not always produce the desired effect. For instance, under both criteria, combination of the rule $p \rightarrow q$ and a fact $p$ leads us to conclude $p \rightarrow q$. This conclusion may not be the desired one in many situations. Likewise, given the following three rules: $p \rightarrow q$, $q \rightarrow r$, and $r \rightarrow p$, we may not want to view the first pair of rules as being independent (because $p \rightarrow q$). Independence criteria appropriate to that situation will be presented elsewhere.

Finally, the very concept of independence may be too restrictive. The problem is that we view rules as being either totally independent, or totally dependent on each other. In practice, however, one may want to think of a pair of rules $p \rightarrow q$ and $q \rightarrow p$ as being only partially dependent (because of the common premise $q$). This means that different evidences supplied by the two rules should not be combined to a full extent, as it would be in the case of totally independent rules. We are currently pursuing several possibilities for accommodating partial independence in the proposed framework.

7. References