Semantics, Consistency and Query Processing of Empirical Deductive Databases

Raymond T. Ng[*]

Department of Computer Science
University of British Columbia
Vancouver, B.C., Canada V6T 1Z4

Abstract

In recent years, there has been growing interest in reasoning with uncertainty in logic programming and deductive databases. However, most frameworks proposed thus far are either non-probabilistic in nature or based on subjective probabilities. In this paper, we address the problem of incorporating empirical probabilities – that is, probabilities obtained from statistical findings – in deductive databases. To this end, we develop a formal model-theoretic basis for such databases. We also present a sound and complete algorithm for checking the consistency of such databases. Moreover, we develop consistency-preserving ways to optimize the algorithm for practical usage. Finally, we show how query answering for empirical deductive databases can be carried out.
Keywords: deductive databases, empirical probabilities, model semantics, constraint satisfaction, optimizations, query answering

1 Introduction

Uncertainty management plays a central role in everyday human decision making in general, and in many next-generation DBMSs in particular (e.g. one managing a scientific or image database). Of all scientific investigations into reasoning with uncertainty and chance, probability theory is one of the best understood paradigms. However, most of the probabilistic frameworks studied in deductive databases and artificial intelligence are based on subjective probabilities [18, 19, 21]. As argued by Bacchus [1], the subjective interpretation of probabilities "view probabilities as degrees of belief, held by a particular agent at a particular time." This leads to the following difficulties in terms of usability: i) this view does not suggest how an agent can acquire the probabilities, and ii) this view does not provide a mechanism for the agent to revise the probabilities. In contrast, empirical probabilities represent statistical truths about the world. They are objective in nature and are independent of the belief of an agent. They can be obtained and updated by statistical samples. Thus, the aim of this paper is to study how empirical probabilities can be incorporated in deductive databases.

[*]Research partially sponsored by NSERC Grants OGP0138055 and STR0134419.

To motivate our approach and framework, it is best to review how statistical inferencing is usually carried out. Suppose we wish to find out whether voters are satisfied with the performance of the head of a government. What we do then is to find a (sufficiently large) sample of voters, and to ask each of them whether he or she is satisfied. At the end of the sampling, we know exactly how each of the voters in the sample feels. However, a far more important purpose of the sample is to allow us to infer inductively how a *typical* voter in the population feels, who is more likely not having been included in the sample. More specifically, if there are n voters in the sample, k of whom are satisfied, we induce the probability that a typical voter is satisfied is (k/n), with certain margins of error.

To appropriately capture the kind of statistical inferencing outlined above, we study deductive databases which consist of the following two parts. The first component, called a *context*, stores two-valued true/false knowledge about the sample. The second component consists of conditional probability statements that are derived from the sample, and that can be used to induce information about individuals or objects not in the sample but in the population. As we shall see later, one of the advantages of this structure is that essentially the context is a normal deductive database. In other words, our empirical deductive databases are "downward" compatible with existing deductive databases without probabilities.

The framework presented here generalizes the preliminary framework reported in [20]. While the latter only supports unary predicates, the framework here provides for predicates of arbitrary arities. To deal with this generality and gain in expressive power, as we shall see later, we need to introduce a sophisticated notion of partitions and subpartitions, and to handle the possible existence of combinations of variable symbols. Moreover, we present here various ways to optimize our consistency checking algorithm for practical usage, and develop a query answering procedure that can handle non-ground queries and rank multiple answers to the queries. The following list summarizes the principal contributions of this paper.

1. The first contribution is the development of a formal model-theoretic basis for our empirical deductive databases. This model theory deals with the complication that Herbrand interpretations are not adequate to capture the essence of empirical reasoning.
2. The second contribution is the development of a sound and complete algorithm for checking the consistency (and $J-$consistency) of our databases. This algorithm is based on constraint satisfaction and can be readily implemented by mixed integer programming techniques.
3. The third contribution is the development of various consistency-preserving ways to optimize the above algorithm for practical usage.
4. The final contribution is the development of sound query processing procedures that can support inductive reasoning, and rank multiple answers to the queries.

The organization of the paper is as follows. Sections 2 and 3 introduce the syntax and develop a model semantics for empirical deductive databases. Sections 4, 5 and 6 study how the consistency of such databases can be verified, and how consistency checking can be optimized. Section 7 develops query answering procedures. Section 8 compares our proposed framework with related works.

2 Empirical Deductive Databases

Let $\mathcal{L}$ be a language generated by finitely many predicate symbols and constant symbols, but no function symbols. An *empirical deductive database*, or an empirical program, consists of two parts: a *context* and a set of *empirical clauses.*

Definition 1 A *context* C is a finite set of clauses of the form: $L_0 \leftarrow L_1 \wedge \ldots \wedge L_n$, where:
i) for all $0 \leq i \leq n$, L_i is a literal in $\mathcal{L}$; and
ii) any variable appearing in L_0 must appear in any one of $L_1, \ldots, L_n$. □

A non-ground clause in a context is implicitly universally quantified at the front of the clause. All negations in empirical programs are interpreted classically – not non-monotonically.

Definition 2 An *empirical clause* is of the form:

$$[c_1, c_2]\ L_0 \quad \leftarrow \quad L_1 \wedge \ldots \wedge L_n$$

where:
i) c_1, c_2 are real numbers in [0,1] such that $c_1 \leq c_2$, $[c_1, c_2] \neq [0, 0]$ and $[c_1, c_2] \neq [1, 1]$;
ii) for all $0 \leq i \leq n$, L_i is a literal in $\mathcal{L}$; and
iii) any variable appearing in L_0 must appear in any one of $L_1, \ldots, L_n$.
L_0 and $L_1 \wedge \ldots \wedge L_n$ are called the head and the body of the empirical clause respectively.
□

Example 1 The following empirical clauses represent two aspects of the voting behavior in an election:

$$\begin{aligned} [0.65, 0.7]\ male(X) &\quad \leftarrow \quad voted(X, Y) \\ [0.7, 0.8]\ voted(Z, Y) &\quad \leftarrow \quad voted(X, Y) \wedge spouse(X, Z) \end{aligned}$$

The intended meaning of the first clause is that: "given an arbitrary individual X who voted (for someone Y), the conditional probability that X is male is within the range [0.65, 0.7]." From an empirical point of view, this clause states that: "amongst all those who voted, between 65% and 70% of them are male." Similarly, the second clause indicates that among all those who voted and who have spouses, 70% to 80% of them voted for the same candidate as their spouses. □

Note that an empirical clause is *not* meant to include universal quantifiers on variables appearing in the clause. For instance, the first clause in the above example is not a statement about a universally quantified variable X; rather, it is a statement about a "generic" individual X in the domain of discourse.

Example 2 In recent years, there has been growing interest in data mining or knowledge discovery in databases. Many proposed frameworks attempt to learn quantitative rules from data [23]. For example, the attribute-oriented approach proposed in [10] learns probabilistic rules of the form: $learning\_class(X) \leftarrow condition(X)$. For instance, the empirical clause

$$[0.63, 0.63] graduate(X) \quad \leftarrow \quad canadian(X) \wedge GPA(X, excellent) \wedge major(X, arts)$$

represents the rule saying that amongst all Canadian students majoring in arts with excellent GPAs, 63% of them are graduate students. □

Definition 3 An *empirical program* (deductive database) $P = \langle C, E \rangle$ consists of a context C and a finite set E of empirical clauses. □

Example 3 Consider the empirical program:

$$\begin{array}{lrcl}
C: & male(duke) & \leftarrow & \\
 & voted(duke, bush) & \leftarrow & \\
 & male(Y) & \leftarrow & voted(X, Y) \\
E: & [0.65, 0.7]\ male(X) & \leftarrow & voted(X, Y) \\
 & [0.2, 0.3]\ \neg male(X) & \leftarrow & voted(X, bush) \\
 & [0.3, 0.4]\ voted(X, Y) & \leftarrow & young(X) \wedge \neg young(Y) \wedge candidate(Y)
\end{array}$$

The clauses in the context C indicate that *duke* is male and voted for *bush*, and that all those who were voted for are male. The second empirical clause indicates that amongst all those voted for *bush*, between 20% to 30% are not male. The third clause in E states that amongst all pairs $\langle X, Y \rangle$ where X is young and Y is a non-young candidate, between 30% and 40% of them satisfy the relationship that X voted for Y. □

3 Model Theoretic Semantics

In this section, we first discuss why Herbrand interpretations may not always be appropriate for empirical programs. We then present a model theory that suitably handles this complication, and that satisfies many requirements of probability theory. In later sections, we present algorithms that determine the consistency of empirical programs.

3.1 Insufficiency of Herbrand Interpretations

In conventional logic programming, it suffices to use the Herbrand universe as the domain of an interpretation. This is also the case in our previous works on supporting subjective probabilities [18, 19]. However, the following example shows that when dealing with empirical probabilities, using the Herbrand universe may not always be appropriate.

Example 4 Consider the following empirical program:

$$\begin{array}{llll} C: & voted(duke, duke) & \leftarrow & \\ E: & [0.65, 0.7]\; male(X) & \leftarrow & voted(X, Y) \end{array}$$

The empirical clause states that of all those who voted, between 65% to 70% of them are male. However, the Herbrand universe is the singleton $\{duke\}$. If we restrict ourselves to the Herbrand universe only, since there is only one member in the universe, what then is the meaning of "between 65% to 70% of all those who voted are male"? More specifically, there are two Herbrand interpretations that can possibly satisfy the empirical program, namely $\{voted(duke, duke), male(duke)\}$ and $\{voted(duke, duke)\}$. In both cases, $male(duke)$ is either true or false. With only these two interpretations, it is unclear how they can "satisfy" the empirical program. □

3.2 Interpretations and Models

As the above example shows, restricting our attention to Herbrand interpretations alone may fail to capture the nature of empirical reasoning. Thus, in our model theory, we also allow non-Herbrand interpretations.

Definition 4 An *interpretation* I for our language $\mathcal{L}$ consists of the following:
i) a non-empty, *finite* set D called the domain of I;
ii) for each constant symbol in $\mathcal{L}$, the assignment of an element in D; and
iii) for each predicate symbol q of arity n in $\mathcal{L}$, the assignment of a mapping ϕ_q that maps D^n to $\{true, false\}$.
Furthermore, we make the assumption that different constant symbols in $\mathcal{L}$ are mapped to distinct elements in the domain [1]. □

Note that for our language $\mathcal{L}$, the notion of an interpretation is almost identical to the usual one for first-order languages [15]. A key difference is that the domain D is assumed to be finite. There are two reasons for this assumption. The first one is that empirical programs are intended to reflect findings from statistical samples which are always finite. The other reason, which is more technical, will be apparent in a later definition on *satisfaction* (cf. Definition 9).

Suppose q is a predicate symbol of arity n. To suitably capture the essence of empirical probabilities, we need to count the number of n-tuples in D^n such that the tuples are assigned $true$ by ϕ_q. However, this counting process is complicated by the following issues:

[1]This is similar to enforcing an equality axiom that says that distinct constants in $\mathcal{L}$ are not equal [15].

- the variables $X_1, \ldots, X_n$ appearing in an atom $q(X_1, \ldots, X_n)$ may not be all distinct (e.g. $voted(X, X)$);
- an atom may be partially or fully ground (e.g. $voted(X, bush)$); and
- predicate symbols may be of varying arities.

To deal with these complications, we define the following notions.

Definition 5 Let $L_1 \wedge \ldots \wedge L_n$ ($n \geq 1$) be a conjunction of literals which may include constant symbols.
i) We use the notation $Cfree(L_1 \wedge \ldots \wedge L_n)$ to denote the "generalization" of $L_1 \wedge \ldots \wedge L_n$ where each constant occurring in the conjunction is replaced by a new variable.
ii) Correspondingly, we use the notation $CVar(L_1 \wedge \ldots \wedge L_n)$ to denote the set of all pairs $\langle m_w, i_w \rangle$ where the constant c_{m_w} is replaced by variable X_{i_w} in $Cfree(L_1 \wedge \ldots \wedge L_n)$. □

Definition 6 Let I be an interpretation with domain D, and $L_1 \wedge \ldots \wedge L_n$ ($n \geq 1$) be a conjunction of literals which may include constant symbols. Let the list of distinct variables in $Cfree(L_1 \wedge \ldots \wedge L_n)$ be $X_1, \ldots, X_k$. Let $\theta = \{\langle X_i, d_i \rangle |\ d_i \in D, 1 \leq i \leq k\}$.
i) (Base case 1: $n = 1$) If $L_1 \wedge \ldots \wedge L_n \equiv A$ and $Cfree(A) \equiv q(t_1, \ldots, t_j)$, define $Cond(A)$ to be $\phi_q(\langle t_1, \ldots, t_j \rangle \theta) = true$, where $\langle t_1, \ldots, t_j \rangle \theta$ denotes the tuple where all occurrences of X_i in $\langle t_1, \ldots, t_j \rangle$ are replaced by d_i for all $\langle X_i, d_i \rangle$ in θ.
ii) (Base case 2: $n = 1$) If $L_1 \wedge \ldots \wedge L_n \equiv \neg A$ and $Cfree(\neg A) \equiv \neg q(t_1, \ldots, t_j)$, define $Cond(\neg A)$ to be $\phi_q(\langle t_1, \ldots, t_j \rangle \theta) = false$.
iii) (Inductive case: $n > 1$) Define $Cond(L_1 \wedge \ldots \wedge L_n)$ to be $\bigwedge_{i=1,\ldots,n} Cond(L_i)$. □

For instance, $Cond(voted(X_1, X_2) \wedge young(X_1) \wedge \neg young(X_2))$ is the condition $\phi_{voted}(\langle d_1, d_2 \rangle) = true \wedge\ \phi_{young}(\langle d_1 \rangle) = true \wedge\ \phi_{young}(\langle d_2 \rangle) = false$. While the above notations deal with the complications that variables appearing in a conjunctions of literals may not be distinct and that constants may occur as well, the following notation handles the complication that different predicate symbols may have different arities.

Definition 7 Let P be an empirical program and Cl be an empirical clause.
i) Given $Cl \equiv [c_1, c_2]L_0 \leftarrow L_1 \wedge \ldots \wedge L_n$, define $sa(Cl)$ to be the total number of constants and distinct variables appearing in the head and the body of Cl (i.e. the total number of distinct variables in $Cfree(L_0 \wedge \ldots \wedge L_n)$).
ii) Define $ma(P)$ to be the maximum (value) in $\{sa(Cl) \mid Cl$ an empirical clause in program $P\}$. □

We can now define how to count the number of tuples satisfying a conjunction of literals.

Definition 8 Let I be an interpretation with domain D, P be an empirical program, and $L_1 \wedge \ldots \wedge L_n$ be a conjunction of literals. Define $\|L_1 \wedge \ldots \wedge L_n\|_I$ to be the cardinality of the set $\{\langle d_1, \ldots, d_{ma(P)} \rangle \in D^{ma(P)} \mid Cond(L_1 \wedge \ldots \wedge L_n)$ and $d_{i_w} = c_{m_w}$ for all pairs $\langle m_w, i_w \rangle$ in $CVar(L_1 \wedge \ldots \wedge L_n)\}$. □

In the above definition, and hereafter whenever no confusion arises, we abuse notation by using $d_{i_w} = c_{m_w}$ to denote the fact that under interpretation I, constant symbol c_{m_w} is assigned to the element d_{i_w} in D.

Example 5 For the empirical program P discussed in Example 3, $ma(P)$ is 2. Let I be an interpretation with domain D. Then $\|voted(X_1, bush)\|_I$ is the cardinality of the set $\{\langle d_1, d_2\rangle \in D^2 | \ \phi_{voted}(\langle d_1, d_2\rangle) = true$ and $d_2 = bush\}$. Similarly, $\|voted(X_1, bush) \wedge \neg male(X_1)\|_I$ is the cardinality of $\{\langle d_1, bush\rangle \in D^2 | \ \phi_{voted}(\langle d_1, bush\rangle) = true$ and $\phi_{male}(\langle d_1\rangle) = false\}$. □

Before we proceed to define the notion of satisfaction for empirical programs, recall that such a program consists of a context C and a set E of empirical clauses. Given the nature of C, we say that an interpretation I satisfies C iff I satisfies every clause in C in the usual sense for first-order languages. Thus, we only need to define the condition for I to satisfy the set E of empirical clauses.

Definition 9 Let $\langle C, E\rangle$ be an empirical program and I be an interpretation.
1) We say that I satisfies the empirical clause $[c_1, c_2]\ L_0 \leftarrow L_1 \wedge \ldots \wedge L_n$ iff:
i) $\|L_1 \wedge \ldots \wedge L_n\|_I = 0$ or
ii) whenever $\|L_1 \wedge \ldots \wedge L_n\|_I > 0$,

$$c_1 \ \leq \ \frac{\|L_0 \wedge L_1 \wedge \ldots \wedge L_n\|_I}{\|L_1 \wedge \ldots \wedge L_n\|_I} \ \leq \ c_2.$$

2) We say that I satisfies E iff I satisfies every empirical clause in E. Finally, I satisfies the empirical program $\langle C, E\rangle$ iff I satisfies both C and E. □

Example 6 For the empirical program discussed in Example 4, consider the following interpretation:

- The domain of I is the set $D = \{d_1, \ldots, d_{20}\}$.
- I assigns $duke$ to d_1.
- I assigns to $voted$ the mapping f_1 such that $f_1(\langle d_i, d_j\rangle) = true$ iff $1 \leq i \leq 10$ and $d_j = d_1$.
- I assigns to $male$ the mapping f_2 such that $f_2(\langle d_i\rangle) = true$ iff $1 \leq i \leq 7$.

Obviously, I satisfies the context C of the program. By definition, $\|voted(X_1, X_2)\|_I$ is the cardinality of the set $\{\langle e_1, e_2\rangle \in D^2 | \ f_1(\langle e_1, e_2\rangle) = true\}$. This value is 10. Similarly, $\|voted(X_1, X_2) \wedge male(X_1)\|_I$ is the cardinality of the set $\{\langle e_1, e_2\rangle \in D^2 | \ f_1(\langle e_1, e_2\rangle) = true$ and $f_2(\langle e_1\rangle) = true\}$. This value is 7. Thus, I satisfies the empirical clause and the program. □

Recall that an empirical clause intuitively represents a conditional probability statement. More specifically, if S_1 denotes the set (event) that satisfies $L_1 \wedge \ldots \wedge L_n$, S_2 the set that satisfies L_0, and S_3 the set that satisfies $L_0 \wedge L_1 \wedge \ldots \wedge L_n$ [2], then the empirical clause $[c_1, c_2]L_0 \leftarrow L_1 \wedge \ldots \wedge L_n$ corresponds to the statement: $c_1 \leq Prob(S_2|S_1) = \frac{Prob(S_3)}{Prob(S_1)} \leq c_2$, assuming that $Prob(S_1) > 0$. Furthermore, when probabilities are interpreted empirically over statistical samples, we have: $\frac{Prob(S_3)}{Prob(S_1)} = \frac{Num(S_3)}{Num(S_1)}$, where $Num(S)$ denotes the cardinality of set S. This is the intuition behind Definition 9 for an interpretation to satisfy an empirical clause. Moreover, in the same definition, it is crucial that whenever $\|L_1 \wedge \ldots \wedge L_n\|_I > 0$, the ratio $\frac{\|L_0 \wedge L_1 \wedge \ldots \wedge L_n\|_I}{\|L_1 \wedge \ldots \wedge L_n\|_I}$ must be well-defined and between 0 and 1. This is guaranteed by the restriction that all interpretations of $\mathcal{L}$ have finite domains (cf. Definition 4) and that variables occurring in the head of an empirical clause must appear in the body (cf. Definition 2).

The lemma below, which is very easy to be proved, shows a few elementary properties of an interpretation I. The first three properties correspond to the requirements proposed by Fenstad for defining a probability function on a first-order language [6].

Lemma 1 (Properties related to Probability Theory) Let I be an interpretation and P be an empirical program. Then the following conditions hold:
i) $\|L_1\|_I = \|L_2\|_I$, if L_1 and L_2 are logically equivalent;
ii) $\|L_1\|_I = n^{ma(P)} - \|\neg L_1\|_I$, where n is the cardinality of the domain of I;
iii) $\|L_1 \vee L_2\|_I + \|L_1 \wedge L_2\|_I = \|L_1\|_I + \|L_2\|_I$;
iv) $\|L_1 \wedge \ldots \wedge L_n\|_I \leq \|L_0\|_I$, if $L_0 \leftarrow L_1 \wedge \ldots \wedge L_n$ is true in I; and
v) $\|L_0 \wedge L_1 \wedge \ldots \wedge L_n\|_I = \|L_1 \wedge \ldots \wedge L_n\|_I$, iff $L_0 \leftarrow L_1 \wedge \ldots \wedge L_n$ is true in I. □

4 Consistency of Sets of Empirical Clauses

In the previous section, we have presented a model theory for empirical programs. We say that an empirical program is consistent iff a model exists for that program. In the remainder of this paper, we discuss how to determine the consistency of empirical programs. Our approach is based on constraint satisfaction, which may be implemented by (mixed integer) linear programming techniques. For the ease of understanding, in this section we first present a method for determining the consistency of sets of empirical clauses (i.e. empirical programs with empty contexts), and show its soundness and completeness. In the next section, we will discuss how this method can be optimized in practice, and in Section 6, we will show how to extend this method to general empirical programs that may have non-empty contexts.

[2] S_2 may or may not be equal to $S_1 \cap S_3$, depending on the variables occurring in L_0.

4.1 Enumeration of Partitions

Recall that the satisfaction of an empirical clause involves counting tuples of the right kinds. This counting process can be facilitated by dividing the set of all tuples into partitions, as illustrated below.

Example 7 Consider the following set E of empirical clauses:

$$\begin{array}{rcl} [0.5, 0.6]voted(X, bush) & \leftarrow & male(X) \\ [0.7, 0.8]voted(X, perot) & \leftarrow & male(X) \end{array}$$

To determine whether E is consistent or not, we begin by partitioning all pairs $\langle d_1, d_2\rangle \in D^2$ where D is the domain of an interpretation to be tested. First of all, there is the set S_1 of pairs $\langle d_1, d_2\rangle$ such that $voted(X, bush)$ is "satisfied" (i.e. $bush$ is assigned to d_2 and $\phi_{voted}(\langle d_1, d_2\rangle) = true$). S_1 can be further decomposed into two partitions: one including all those $\langle d_1, d_2\rangle$ such that d_1 satisfies $male(X)$, and the other one including all the remaining pairs in S_1. Thus, the cardinality of the former partition, say v_1, gives the number of pairs that satisfy $voted(X, bush) \wedge male(X)$. Similarly, the set $\overline{S_1}$ can be divided into two partitions: one including all the pairs that satisfy $male(X)$, and the other one including all the rest in $\overline{S_1}$. If v_2 is the cardinality of the former partition, then the total number of pairs satisfying $male(X)$ is given by $v_1 + v_2$. Hence, to check whether the interpretation satisfies the first empirical clause amounts to checking whether $\frac{v_1}{v_1+v_2}$ is within the range [0.5,0.6] (cf. Definition 9). □

This example outlines how partitioning can help to translate the consistency checking problem into a constraint satisfaction problem. However, this example is simplistic in that the partitions are set up to accommodate one clause only. In general, the partitions must be set up in such a way that all the empirical clauses in a set can be accommodated. Moreover, partitioning is further complicated by the possible appearance of constants and common variables in literals. The partitioning scheme below deals with these complications.

Given a literal L, we use $pos(L)$ to denote $\neg L$ if L is negative, and to denote L if L is positive. Let $\mathcal{Q}$ be the set $\{pos(L) \,|\, L \text{ appears in program } P\}$ [3]. Furthermore, let all possible subsets of $\mathcal{Q}$ be enumerated in an arbitrary but fixed way: $\mathcal{P}_1, \ldots, \mathcal{P}_t$. For all $1 \leq i \leq t$, let v_i denote the number of tuples that satisfy $\bigwedge_{A \in P_i} A \ \ \bigwedge_{A \notin P_i} \neg A$. It is easy to see that the $\mathcal{P}_i$'s divide the set of all tuples into t partitions. Thus, if $L_1 \wedge \ldots \wedge L_n$ does not contain any constant symbols, then the number of tuples that satisfy the conjunction of literals can be determined using the v_i's. This amounts to checking for all partitions P_i, whether the conjunction is true in $\mathcal{P}_i$ in the classical sense or not. Hereafter, we abuse notation by writing $\mathcal{P}_i \models L_1 \wedge \ldots \wedge L_n$, whenever $L_1 \wedge \ldots \wedge L_n$ is true in $\mathcal{P}_i$. Thus, the number of tuples that satisfy $L_1 \wedge \ldots \wedge L_n$ is given by the summation of the tuples in each $\mathcal{P}_i$ satisfying the literals,

[3] Here we assume that whenever necessary, variables in an empirical clause in program P are renamed to $X_1, \ldots X_{ma(P)}, \ldots$.

i.e. $\sum_{\mathcal{P}_i \models L_1 \wedge \ldots \wedge L_n ; i=1,\ldots,t} v_i$. Throughout this paper, we often write the above summation as simply $\sum_{\mathcal{P}_i \models L_1 \wedge \ldots \wedge L_n} v_i$.

Example 8 Consider the following empirical clauses:

$$\begin{array}{rcl} [0.65, 0.7] male(X_1) & \leftarrow & voted(X_1, X_2) \\ [0, 0.1] \neg male(X_2) & \leftarrow & voted(X_1, X_2) \end{array}$$

$\mathcal{Q}$ is the set $\{voted(X_1, X_2), male(X_1), male(X_2)\}$. This gives rise to the following eight partitions:

$voted(X_1, X_2)$	$male(X_1)$	$male(X_2)$	partition	number
1	1	1	$\mathcal{P}_1$	v_1
1	1	0	$\mathcal{P}_2$	v_2
1	0	1	$\mathcal{P}_3$	v_3
1	0	0	$\mathcal{P}_4$	v_4
0	1	1	$\mathcal{P}_5$	v_5
0	1	0	$\mathcal{P}_6$	v_6
0	0	1	$\mathcal{P}_7$	v_7
0	0	0	$\mathcal{P}_8$	v_8

The first partition consists of pairs $\langle d_1, d_2 \rangle$ such that $\phi_{voted}(\langle d_1, d_2 \rangle) = \phi_{male}(\langle d_1 \rangle) = \phi_{male}(\langle d_2 \rangle) = true$; the second partition is identical to the first one except that $\phi_{male}(\langle d_2 \rangle) = false$, and so on. Since $voted(X_1, X_2)$ is true in $\mathcal{P}_1, \ldots, \mathcal{P}_4$, the number of pairs that satisfy $voted(X_1, X_2)$ is given by $\sum_{\mathcal{P}_i \models voted(X_1, X_2)} v_i = v_1 + \ldots + v_4$. Similarly, the number of pairs satisfying $voted(X_1, X_2) \wedge male(X_1)$ is given by $v_1 + v_2$. Thus, according to Definition 9, for the interpretation to satisfy the first empirical clause, it is necessary that the constraint $0.65 \leq \frac{v_1 + v_2}{v_1 + \ldots + v_4} \leq 0.7$ is satisfied. Similarly, for the second clause, the constraint $0 \leq \frac{v_2 + v_4}{v_1 + \ldots + v_4} \leq 0.1$ cannot be violated. □

Example 9 We now rewrite the second clause in the previous example to $[0, 0.1] \neg male(X_1) \leftarrow voted(X_2, X_1)$, by interchanging the roles of X_1 and X_2. Then $\mathcal{Q}$ becomes the set $\{voted(X_1, X_2), male(X_1), voted(X_2, X_1)\}$. This gives rise to 8 partitions different from the ones listed above. Suppose that the third column of the partition table in the previous example is now used for $voted(X_2, X_1)$, instead of for $male(X_2)$. Then the constraint for the rewritten clause becomes $0 \leq \frac{v_3 + v_7}{v_1 + v_3 + v_5 + v_7} \leq 0.1$. On the surface, this constraint is very different from the corresponding one listed in the previous example. But a careful analysis reveals that based on the new partitions, $v_1 + v_3 + v_5 + v_7$ represents the cardinality of the set $\{\langle d_1, d_2 \rangle \in D^2 | \ \phi_{voted}(\langle d_2, d_1 \rangle) = true\}$. Similarly, $v_3 + v_7$ represents the cardinality of the set $\{\langle d_1, d_2 \rangle \in D^2 | \ \phi_{voted}(\langle d_2, d_1 \rangle) = true \wedge \ \phi_{male}(\langle d_1 \rangle) = false\}$. It is obvious that the cardinality of this set is identical to the cardinality of the set $\{\langle d_1, d_2 \rangle \in D^2 | \ \phi_{voted}(\langle d_1, d_2 \rangle) = true \wedge \ \phi_{male}(\langle d_2 \rangle) = false\}$. Thus, the constraint here is equivalent to the corresponding constraint in the previous example. In other words, renaming variables in a clause will cause partitions and constraints to change – but only to something equivalent. □

4.2 Subpartitioning for Constant Symbols

In finding out the number of tuples satisfying the conjunction $L_1 \wedge \ldots \wedge L_n$, so far we have only considered the case when the conjunction does not include any constant symbols. For instance, the partitioning scheme above does not directly give the number of tuples satisfying the atom $voted(X_1, bush)$ in Example 7. To deal with the occurrences of constant symbols, we can subdivide a partition according to the constant symbols. In particular, for partition $\mathcal{P}_i$, we use $v_i^{\langle -,bush \rangle}$ to denote the number of pairs $\langle d_1, d_2 \rangle$ in $\mathcal{P}_i$, where *bush* is assigned to d_2, and the hyphen $-$ indicates that d_1 can be anything in the domain. Similarly, we use $v_i^{\langle -,perot \rangle}$ to denote the number of pairs $\langle d_1, d_2 \rangle$ where *perot* is assigned to d_2. Note that these two numbers are related by the constraint $v_i^{\langle -,bush \rangle} + v_i^{\langle -,perot \rangle} \leq v_i$.

While the description above outlines the idea of subpartitioning (e.g. $v_i^{\langle -,bush \rangle}$), it does not indicate which partition (i.e. v_i) needs to be used. This can be achieved by the process described in the previous subsection. More specifically, to determine the (sub)partitions for $L_1 \wedge \ldots \wedge L_n$ (which may include constants), we first determine the partitions $\mathcal{P}_i$ based on the generalization $Cfree(L_1 \wedge \ldots \wedge L_n)$ introduced in Definition 5. Then we use the subpartitions based on the constants occurring in the conjunction in the way described here. For instance, for the atom $voted(X_1, bush)$, we first determine the appropriate partitions $\mathcal{P}_i$'s in which $voted(X_1, X_2)$ is true. Then for all such partitions, the summation of $v_i^{\langle -,bush \rangle}$ gives the total number of tuples satisfying the atom. The following definition helps to formalize this discussion.

Definition 10 Let P be an empirical program, and $L_1 \wedge \ldots \wedge L_n$ be a conjunction of literals which may include constant symbols. Define $Label(L_1 \wedge \ldots \wedge L_n)$ to be the tuple $\langle u_1, \ldots, u_{ma(P)} \rangle$ where for all $1 \leq j \leq ma(P)$:
i) $u_j = c_{m_w}$ iff there exists a pair $\langle m_w, i_w \rangle$ in $CVar(L_1 \wedge \ldots \wedge L_n)$ such that $j = i_w$;
ii) otherwise, $u_j = -$. □

4.3 Constraint Version of Empirical Clauses

We are now in a position to define the constraint version of an empirical clause.

Definition 11 Let $Cl \equiv [c_1, c_2]L_0 \leftarrow L_1 \wedge \ldots \wedge L_n$ be an empirical clause. The *constraint* version of Cl, denoted by $con(Cl)$, is the constraint:

$$\begin{aligned} c_1 * \Big(\sum_{\mathcal{P}_i \models Cfree(L_1 \wedge \ldots \wedge L_n)} v_i^{Label(L_1 \wedge \ldots \wedge L_n)} \Big) &\leq \Big(\sum_{\mathcal{P}_i \models Cfree(L_0 \wedge L_1 \wedge \ldots \wedge L_n)} v_i^{Label(L_0 \wedge L_1 \wedge \ldots \wedge L_n)} \Big) \\ &\leq c_2 * \Big(\sum_{\mathcal{P}_i \models Cfree(L_1 \wedge \ldots \wedge L_n)} v_i^{Label(L_1 \wedge \ldots \wedge L_n)} \Big). \end{aligned}$$

□

Example 10 Consider setting up the constraint versions of the clauses listed in Example 7. Using the partitions listed in Example 8, the number of tuples satisfying $male(X_1)$ is given by $v_1 + v_2 + v_5 + v_6$. As for $voted(X_1, bush) \wedge male(X_1)$, $Cfree(voted(X_1, bush) \wedge male(X_1))$ is given by $voted(X_1, X_2) \wedge male(X_1)$, which is true in $\mathcal{P}_1$ and $\mathcal{P}_2$. Furthermore, $Label(voted(X_1, bush) \wedge male(X_1))$ is given by $\langle -, bush\rangle$. Thus, the number of tuples satisfying $(voted(X_1, bush) \wedge male(X_1))$ is $v_1^{\langle -,bush\rangle} + v_2^{\langle -,bush\rangle}$. Hence, the constraint version of the clause $[0.5, 0.6]voted(X_1, bush) \leftarrow male(X_1)$ is $0.5*(v_1+v_2+v_5+v_6) \leq (v_1^{\langle -,bush\rangle}+v_2^{\langle -,bush\rangle}) \leq 0.6 * (v_1 + v_2 + v_5 + v_6)$. Similarly, for the second clause in Example 7, the constraint is $0.7 * (v_1 + v_2 + v_5 + v_6) \leq (v_1^{\langle -,perot\rangle} + v_2^{\langle -,perot\rangle}) \leq 0.8 * (v_1 + v_2 + v_5 + v_6)$. □

Strictly speaking, in the above example, $(v_1 + v_2 + v_5 + v_6)$ should be written as $(v_1^{\langle -,-\rangle} + v_2^{\langle -,-\rangle} + v_3^{\langle -,-\rangle} + v_4^{\langle -,-\rangle})$. However, for notational simplicity, whenever $Label(L_1 \wedge \ldots \wedge L_n)$ contains all $-$'s and no constant symbols, we simply regard it as null.

To generalize the notion of the constraint version of an empirical clause to the constraint version of a set of such clauses, we need to deal with two issues. First, as required in Definition 4, the domain of an interpretation cannot be empty. This property can be enforced by the constraint $\sum_{i=1}^{t} v_i \geq 1$, where t denotes the number of partitions as before. Second, constraints must be set up to deal with the subpartitions created for constant symbols as described in the previous subsection.

Definition 12 Let P be an empirical program, and v_i denote the number of tuples in $\mathcal{P}_i$. Define $subpar(v_i)$ to be the following set of constraints:
$subpar(v_i) = \{v_i^{\langle u_1,\ldots,u_{j-1},-,u_{j+1},\ldots,u_{ma(P)}\rangle} \geq \sum_c v_i^{\langle u_1,\ldots,u_{j-1},c,u_{j+1},\ldots,u_{ma(P)}\rangle} |$ for all $1 \leq w \leq ma(P)$, u_w can either be $-$ or any constant symbol in our language $\mathcal{L}\}$,
where the symbol $\sum_c$ denotes the sum over all constant symbols c in our language. □

For instance, consider the set of empirical clauses in Example 7. Among others not shown, the following three constraints are in $subpar(v_i)$: i) $v_i \equiv v_i^{\langle -,-\rangle} \geq v_i^{\langle -,bush\rangle} + v_i^{\langle -,perot\rangle}$; ii) $v_i \geq v_i^{\langle bush,-\rangle} + v_i^{\langle perot,-\rangle}$; and iii) $v_i^{\langle -,bush\rangle} \geq v_i^{\langle bush,bush\rangle} + v_i^{\langle perot,bush\rangle}$.

Definition 13 Let E be a set of empirical clauses. The constraint version of E, denoted by $con(E)$, is the set: $\{con(Cl) | Cl \in E\} \cup \{\sum_{i=1}^{t} v_i \geq 1\} \cup \bigcup_{i=1}^{t} subpar(v_i)$. □

As stated above, $con(E)$ contains a huge number of variables and constraints. In the next section, we discuss how in practice the numbers of variables and constraints can be drastically reduced.

4.4 Soundness and Completeness

The reason why we set up $con(E)$ is that we intend to check the consistency of E based on the constraints in $con(E)$. The major result of this section is to prove that E is consistent iff there is a solution to the constraints in $con(E)$ (cf. Theorem 1 below). To obtain this result, we need the following definitions and lemmas. To be compatible with the notation described before, let subpartitions of $\mathcal{P}_i$ be denoted by $\mathcal{P}_i^{\langle u_1,\ldots,u_{ma(P)}\rangle}$ where for all $1 \leq w \leq ma(P)$, u_w can either be $-$ or any constant symbol in $\mathcal{L}$. Recall that $\mathcal{P}_i$ can be represented by the conjunction $\bigwedge_{A\in\mathcal{P}_i} A \bigwedge_{A\notin\mathcal{P}_i} \neg A$.

Definition 14 Let $\mathcal{P}_i^{\langle u_1,\ldots,u_{ma(P)}\rangle}$ be a subpartition. Define $Cbind(\mathcal{P}_i^{\langle u_1,\ldots,u_{ma(P)}\rangle})$ to be the version ("restriction") of $\bigwedge_{A\in\mathcal{P}_i} A \bigwedge_{A\notin\mathcal{P}_i} \neg A$ such that for all $1 \leq w \leq ma(P)$, whenever $u_w = c_w$ for some constant symbol c_w, all occurrences of X_w in the conjunction are replaced by c_w. □

$Cbind()$ can be regarded as the inverse of $Cfree()$ introduced earlier. As an example, for the partition $\mathcal{P}_1$ listed in Example 8, $Cbind(\mathcal{P}_1^{\langle -,bush\rangle})$ is the conjunction $voted(X_1, bush) \wedge male(X_1) \wedge male(bush)$.

Definition 15 Let I be an interpretation. Let S_I be defined as follows: for all partitions $\mathcal{P}_i$,
i) $S_I(\mathcal{P}_i) = \| \bigwedge_{A\in\mathcal{P}_i} A \bigwedge_{A\notin\mathcal{P}_i} \neg A\|_I$; and
ii) for all subpartitions $\mathcal{P}_i^{\langle u_1,\ldots,u_{ma(P)}\rangle}$ of $\mathcal{P}_i$, $S_I(\mathcal{P}_i^{\langle u_1,\ldots,u_{ma(P)}\rangle}) = \|Cbind(\mathcal{P}_i^{\langle u_1,\ldots,u_{ma(P)}\rangle})\|_I$. □

Intuitively, S_I specifies the number of tuples within each partition $\mathcal{P}_i$ and each subpartition of $\mathcal{P}_i$. The following lemma shows that the number of tuples satisfying $L_1 \wedge \ldots \wedge L_n$ is the summation of the number of tuples in each (sub)partition in which the conjunction is true.

Lemma 2 Let I be an interpretation, and $L_1 \wedge \ldots \wedge L_n$ be any conjunction of literals. Then it is the case that $\|L_1 \wedge \ldots \wedge L_n\|_I = \sum_{\mathcal{P}_i \models Cfree(L_1\wedge\ldots\wedge L_n)} S_I(\mathcal{P}_i^{Label(L_1\wedge\ldots\wedge L_n)})$.

Proof Outline <u>Case 1</u> $L_1 \wedge \ldots \wedge L_n$ does not contain any constant symbols.
For simplicity, let $Conj$ denote the conjunction $L_1 \wedge \ldots \wedge L_n$. Then $Cfree(Conj)$ is identical to $Conj$, and $Label(Conj)$ is null. Given the way that the partitions $\mathcal{P}_1, \ldots, \mathcal{P}_t$ are defined, it is easy to see that $\bigvee_{i=1}^{t}(\bigwedge_{A\in\mathcal{P}_i} A \bigwedge_{A\notin\mathcal{P}_i} \neg A)$ is equivalent to true. Thus, $\bigvee_{i=1}^{t}(Conj \bigwedge_{A\in\mathcal{P}_i} A \bigwedge_{A\notin\mathcal{P}_i} \neg A)$ is equivalent to $Conj$. Since for all $i \neq j$, $(Conj \bigwedge_{A\in\mathcal{P}_i} A \bigwedge_{A\notin\mathcal{P}_i} \neg A) \wedge (Conj \bigwedge_{A\in\mathcal{P}_j} A \bigwedge_{A\notin\mathcal{P}_j} \neg A)$ is false in all interpretations, then by Parts (ii) and (iii) of Lemma 1, it is necessary that $\|Conj\|_I = \sum_{i=1}^{t} \|Conj \bigwedge_{A\in\mathcal{P}_i} A \bigwedge_{A\notin\mathcal{P}_i} \neg A\|_I$. Now for all partitions there are two cases.
<u>Case 1.1</u>: $Conj \leftarrow \bigwedge_{A\in\mathcal{P}_i} A \bigwedge_{A\notin\mathcal{P}_i} \neg A$.
This can be rewritten as $\mathcal{P}_i \models Conj$. Now by Part (v) of Lemma 1, it is the case that $\|Conj \bigwedge_{A\in\mathcal{P}_i} A \bigwedge_{A\notin\mathcal{P}_i} \neg A\|_I = \| \bigwedge_{A\in\mathcal{P}_i} A \bigwedge_{A\notin\mathcal{P}_i} \neg A\|_I = S_I(\mathcal{P}_i)$.
<u>Case 1.2</u>: $Conj \not\leftarrow \bigwedge_{A\in\mathcal{P}_i} A \bigwedge_{A\notin\mathcal{P}_i} \neg A$.
Then whenever $\bigwedge_{A\in\mathcal{P}_i} A \bigwedge_{A\notin\mathcal{P}_i} \neg A$ is true in an interpretation, $Conj$ is false in that interpretation. In other words, $Conj \bigwedge_{A\in\mathcal{P}_i} A \bigwedge_{A\notin\mathcal{P}_i} \neg A$ is always false. Hence, it is necessary that $\|Conj \bigwedge_{A\in\mathcal{P}_i} A \bigwedge_{A\notin\mathcal{P}_i} \neg A\|_I = 0$.
By combining the two cases above, $\|Conj\|_I = \sum_{\mathcal{P}_i \models Conj} S_I(\mathcal{P}_i)$.

Case 2 $L_1 \wedge \ldots \wedge L_n$ contain constant symbols.
Let $Conj$ denote the conjunction $L_1 \wedge \ldots \wedge L_n$. To find the number tuples satisfying $Conj$, based on the same argument in Case 1, it suffices to add up tuples in each partition in which $Conj$ is true, i.e. $\mathcal{P}_i \models Cfree(Conj)$. Within each such partition, it amounts to finding the number of tuples in the right subpartition $\mathcal{P}_i^{Label(Conj)}$. This number is given by $\|Cbind(\mathcal{P}_i^{Label(Conj)})\|_I$. Thus, based on the argument in Case 1, and by Definition 15, $\|Conj\|_I = \sum_{\mathcal{P}_i \models Cfree(Conj)} S_I(\mathcal{P}_i^{Label(Conj)})$. □

Lemma 3 I is a model of an empirical clause Cl iff S_I is a solution of $con(Cl)$ (i.e. by assigning all v_i to $S_I(\mathcal{P}_i)$ and all $v_i^{\langle u_1,\ldots,u_{ma(P)}\rangle}$ to $S_I(\mathcal{P}_i^{\langle u_1,\ldots,u_{ma(P)}\rangle})$, the constraint in $con(Cl)$ is satisfied).

Proof Outline Let Cl be $[c_1, c_2]L_0 \leftarrow L_1 \wedge \ldots \wedge L_n$.
Case 1 I is a model of Cl.
There are two cases.
Case 1.1 $\|L_1 \wedge \ldots \wedge L_n\|_I = 0$.
By Lemma 2, it is necessary that $\sum_{\mathcal{P}_i \models Cfree(L_1\wedge\ldots\wedge L_n)} S_I(\mathcal{P}_i^{Label(L_1\wedge\ldots\wedge L_n)}) = 0$. Furthermore, since $L_0 \wedge L_1 \wedge \ldots \wedge L_n \rightarrow L_1 \wedge \ldots \wedge L_n$ is true in I, then by Part (iv) of Lemma 1, $\|L_0 \wedge L_1 \wedge \ldots \wedge L_n\|_I \leq \|L_1 \wedge \ldots \wedge L_n\|_I = 0$. Thus, by Lemma 2 again, it is necessary that $\sum_{\mathcal{P}_i \models Cfree(L_0\wedge L_1\wedge\ldots\wedge L_n)} S_I(\mathcal{P}_i^{Label(L_0\wedge L_1\wedge\ldots\wedge L_n)}) = 0$. Hence, the constraint $con(Cl)$ becomes $c_1 * 0 \leq 0 \leq c_2 * 0$ which is trivially satisfied.
Case 1.2 $\|L_1 \wedge \ldots \wedge L_n\|_I > 0$.
Since I is a model of Cl, by Definition 9, it is true that $c_1 * \|L_1 \wedge \ldots \wedge L_n\|_I \leq \|L_0 \wedge L_1 \wedge \ldots \wedge L_n\|_I \leq c_2 * \|L_1 \wedge \ldots \wedge L_n\|_I$. By Lemma 2, it follows immediately that S_I is a solution of $con(Cl)$.
Case 2 S_I is a solution of $con(Cl)$.
Similar to the argument used in Case 1 above, I satisfies Cl. □

Theorem 1 Let I be an interpretation and E be a set of empirical clauses. I is a model of E iff S_I is a solution of $con(E)$.

Proof Outline By the previous lemma, for every clause Cl in E, I satisfies Cl iff S_I is a solution of $con(Cl)$. Now by Definition 4, I has a non-empty domain. Thus, the constraint $\sum_{i=i}^{t} v_i \geq 1$ must be satisfied. Finally, since I satisfies the properties listed in Lemma 1, for all constraints in $subpar(v_i)$, it is easy to see that it is the case that $S_I(\mathcal{P}_i^{\langle u_1,\ldots,u_{j-1},-,u_{j+1},\ldots,u_{ma(P)}\rangle}) \geq \sum_c S_I(\mathcal{P}_i^{\langle u_1,\ldots,u_{j-1},c,u_{j+1},\ldots,u_{ma(P)}\rangle})$. Thus, S_I satisfies the constraints in $subpar(v_i)$. □

Example 11 Consider the clauses in Example 7. As shown in Example 10, the constraints of the clauses are:

$$0.5 * (v_1 + v_2 + v_5 + v_6) \leq (v_1^{\langle -,bush\rangle} + v_2^{\langle -,bush\rangle}) \leq 0.6 * (v_1 + v_2 + v_5 + v_6) \quad (1)$$
$$0.7 * (v_1 + v_2 + v_5 + v_6) \leq (v_1^{\langle -,perot\rangle} + v_2^{\langle -,perot\rangle}) \leq 0.8 * (v_1 + v_2 + v_5 + v_6) \quad (2)$$

Furthermore, relevant to these two clauses are the following constraints from $subpar(v_1)$ and $subpar(v_2)$:

$$v_1 \geq v_1^{\langle -,bush\rangle} + v_1^{\langle -,perot\rangle} \quad (3)$$

$$v_2 \geq v_2^{\langle -,bush\rangle} + v_2^{\langle -,perot\rangle} \quad (4)$$

Now adding Constraints (3) and (4) gives:

$$v_1 + v_2 \geq v_1^{\langle -,bush\rangle} + v_2^{\langle -,bush\rangle} + v_1^{\langle -,perot\rangle} + v_2^{\langle -,perot\rangle} \quad (5)$$

However, by Constraints (1) and (2), it is the case that $v_1^{\langle -,bush\rangle} + v_2^{\langle -,bush\rangle} + v_1^{\langle -,perot\rangle} + v_2^{\langle -,perot\rangle} \geq (0.5+0.7)*(v_1+v_2+v_5+v_6)$. Combining this with Constraint (5), it is necessary that $(v_1 + v_2) \geq 1.2*(v_1 + v_2 + v_5 + v_6)$. This is only possible if $v_1 = v_2 = v_5 = v_6 = 0$. Intuitively, this implies that the two clauses in Example 7 are only consistent iff there is no tuple satisfying $male(X_1)$. If these clauses are combined with any context that can deduce $male(c)$ for some constant c, inconsistency results. □

Corollary 1 A set E of empirical clauses is consistent iff $con(E)$ has a solution. □

The corollary above follows directly from Theorem 1. Another corollary from the theorem is the one below which states that the approach of constraint satisfaction can also apply to determine $J-consistency$ of an empirical program. That is, given a pre-interpretation J (cf. [15]) and thus a *fixed* domain D, the question is whether there exists an interpretation I based on J such that I is a model of the program.

Corollary 2 Let J be a pre-interpretation with domain D. A set E of empirical clauses is $J-$consistent iff $(con(E) \cup \{\sum_{i=1}^{t} v_i = \|D\|^{ma(E)}\})$ has a solution. □

Example 12 Consider the empirical clause in Example 4: $[0.65, 0.7]male(X_1) \leftarrow voted(X_1, X_2)$. Let J be an Herbrand pre-interpretation with a domain of size 1 (e.g. $D = \{duke\}$ in Example 4). Then following the partitions listed in Example 8, the constraints below need to be satisfied:

$$0.65*(v_1 + \ldots + v_4) \leq (v_1 + v_2) \leq 0.7*(v_1 + \ldots + v_4)$$

$$v_1 + \ldots + v_8 = 1$$

Since all v_i's must be non-negative integers, it is easy to check that the only solution to the constraints is to have $v_1 = \ldots = v_4 = 0$. In other words, for E to be $J-$consistent, $voted(d_1, d_1)$ cannot be true, where d_1 is the sole element in the domain. This explains why the program listed in Example 4 has no Herbrand model. □

5 Optimization for Practical Usage

In the previous section, we present a way of translating a set E of empirical clauses to a set $con(E)$ of constraints. The major result there is that E is consistent iff $con(E)$ has a solution. This suggests that checking the consistency (and similarly the $J-$consistency) of E reduces to checking whether $con(E)$ has an integer solution. Since all the constraints are linear, the latter process can be carried out by mixed integer programming algorithms such as the cutting plane method [16]. Thus, one advantage of the framework set up in the previous section is that implementations of such algorithms are widely available in many systems including for example IBM/PC (i.e. the LINDO package). However, as stated in the previous section, $con(E)$ may contain too huge a number of variables v_i's and constraints for integer linear programming algorithms to tackle. The major results of this section are consistency-preserving ways:

- to eliminate (irrelevant) variables [4] and constraints, and
- to reduce the integer programming problem to a (real-valued) linear programming problem which is computationally much cheaper to solve.

With these optimizations, we believe that it is now feasible to check program consistency by constraint satisfaction.

5.1 Eliminating Irrelevant Constraints and Variables

Recall from Definition 13 that $con(E)$ is the set: $\{con(Cl) | Cl \in E\} \cup \{\sum_{i=1}^{t} v_i \geq 1\} \cup \bigcup_{i=1}^{t} subpar(v_i)$. It is easy to see that the cardinality of the set $SP \equiv \bigcup_{i=1}^{t} subpar(v_i)$ is exponential and huge in value. Fortunately, as shown below, most of the variables and constraints in SP can be eliminated. We call a variable $v_i^{\langle u_1,\ldots,u_{ma(P)}\rangle}$ *irrelevant* if it does not appear in the set $con(E) - SP$.

Definition 16 Let $con(E)$ be a set of constraints as defined in Definition 13. Let the set $con^{elim}(E)$ be constructed as follows:
i) for all constraints in $con(E)$ of the form: $\ldots \geq \ldots$, set all ocurrences of irrelevant variables at the right-hand-side of the constraint to 0;
ii) then discard all constraints of the form $v_i^{\langle u_1,\ldots,u_{ma(P)}\rangle} \geq 0$. □

For instance, if E is the set of clauses considered in Example 11, then Constraints (3) and (4) shown in that example are the only two constraints from $\bigcup_{i=1}^{8} subpar(v_i)$ that remain in $con^{elim}(E)$.

Lemma 4 Let E be a set of empirical clauses. $con(E)$ has a solution iff $con^{elim}(E)$ has a solution.

[4] Throughout this section, the word "variables" refers to variables appearing in constraints, not variables in $\mathcal{L}$.

Proof Outline If $con^{elim}(E)$ has a solution S, then S can be augmented to be a solution of $con(E)$ by setting all irrelevant variables to 0. Thus, it suffices to prove that if $con(E)$ has a solution S, then S is also a solution of $con^{elim}(E)$.

Case 1 all irrelevant variables are 0 in S.

Then trivially S is a solution of $con^{elim}(E)$.

Case 2 there exists some irrelevant variable $v_i^{\langle u_1,\ldots,u_{ma(P)}\rangle} = c > 0$ in S.

Case 2.1 $v_i^{\langle u_1,\ldots,u_{ma(P)}\rangle}$ appears in a constraint in $con(E)$ involving some non-irrelevant variable.

Let the non-irrelevant variable be $v_i^{\langle w_1,\ldots,w_{ma(P)}\rangle}$.

Case 2.1.1 Suppose the constraint is of the form $v_i^{\langle w_1,\ldots,w_{ma(P)}\rangle} \geq V + v_i^{\langle u_1,\ldots,u_{ma(P)}\rangle}$, where V denotes a summation of some variables.

Then since S is a solution to this constraint, and since $c > 0$, S must also satisfy the constraint $v_i^{\langle w_1,\ldots,w_{ma(P)}\rangle} \geq V$. So as this latter constraint is in $con^{elim}(E)$, S satisfies this constraint in $con^{elim}(E)$.

Case 2.1.2 Suppose the constraint is of the form $v_i^{\langle u_1,\ldots,u_{ma(P)}\rangle} \geq V + v_i^{\langle w_1,\ldots,w_{ma(P)}\rangle}$, where V denotes a summation of some variables.

Then the constraint is in $con^{elim}(E)$. Since solution S satisfies this constraint in $con(E)$, it still satisfies this constraint in $con^{elim}(E)$.

Case 2.2 $v_i^{\langle u_1,\ldots,u_{ma(P)}\rangle}$ does not appear in any constraint in $con(E)$ involving non-irrelevant variables.

Then all constraints in which $v_i^{\langle u_1,\ldots,u_{ma(P)}\rangle}$ occur are not included in $con^{elim}(E)$. Thus, since S is a solution of $con(E)$, S is a solution of $con^{elim}(E)$. □

The number of variables appearing in $con^{elim}(E)$ can be further reduced by merging variables that always appear together. For instance, consider the set E of clauses discussed in Example 11. Then $con^{elim}(E)$ consists of Constraints (1) to (4) and $v_1 + \ldots + v_8 \geq 1$. There are a total of 12 variables. Then, it is easy to see that the consistency of $con^{elim}(E)$ is not affected by merging v_5 and v_6 together, and merging v_3, v_4, v_7 and v_8, reducing the total number of variables to 8.

5.2 Dropping Integrality Constraints

Apart from the number of constraints and variables in $con(E)$ (and similarly in $con^{elim}(E)$), whether the variables are integer-valued or real-valued also affects the performance of checking whether $con(E)$ has a solution. If the variables are all integer-valued, the checking process will take a long time. Indeed as defined in the previous section, our variables in $con(E)$ are all integer variables as they serve to count tuples. Fortunately, for all empirical programs we have in mind, the integrality constraints on variables can be dropped, i.e. all variables can be real-valued. The following lemma shows that for empirical programs that only use rational numbers in their empirical clauses, the simplex method [5] for linear programming can be used

[5] While numerous algorithms can be used in the place of the simplex method, the reason why we only show the proof of the lemma for the simplex method is that the method, given its availability, is the algorithm we

directly to check whether $con(E)$ has a solution.

Lemma 5 Let E be a set of empirical clauses using rational numbers in their ranges. Let $con^{real}(E)$ be the variant of $con(E)$ such that all integrality constraints on the variables of $con(E)$ are dropped, i.e. all variables in $con^{real}(E)$ are real-valued. Then: $con(E)$ has a solution iff the simplex method finds a solution for $con^{real}(E)$.

Proof

i) (the "only-if" part) If $con(E)$ has a solution S, then S is integer-valued. Clearly, S is a solution of $con^{real}(E)$. Thus, the simplex method will find a solution for $con^{real}(E)$.

ii) Claim:- (the "if" part) If the simplex method finds a solution for $con^{real}(E)$, then $con(E)$ has a solution.

Consider the real-valued solution S obtained by the simplex method for $con^{real}(E)$. This solution is computed based on a sequence of simplex steps. Within each step, a pivot element y_{pq} in the simplex tableau is chosen. (For our purpose here, a simplex tableau can simply be regarded as a two dimensional array.) Then every element y_{ij} in the tableau is updated to y'_{ij} according to the following formula [16]:

$$y'_{ij} = \begin{cases} y_{ij} - \frac{y_{pj}}{y_{pq}} y_{iq} & i \neq p \\ \frac{y_{pj}}{y_{pq}} & \text{otherwise} \end{cases}$$

If y_{ij}, y_{pj}, y_{pq} and y_{iq} are all rational numbers, so is y'_{ij}. Given the fact that clauses in E only use rational numbers, and the form of the constraints in $con(E)$ and $con^{real}(E)$, all elements in the initial simplex tableau are rational. Thus, it is an easy induction to show that all elements in the final simplex tableau are rational. In other words, the real-valued solution S obtained by the simplex method for $con^{real}(E)$ is rational. By multiplying S with a sufficiently large integer, an integer-valued solution S' can be obtained for $con^{real}(E)$ and hence for $con(E)$. $\Box$

Though the above lemma applies only to empirical clauses using rational numbers, and though as defined in Definition 2, irrational numbers can be used in empirical clauses, all practical empirical programs we have in mind really fall into the category of using rational numbers only. This is because we intend to obtain all the probability ranges $[c_1, c_2]$ from statistical samples. Furthermore, the lemma applies equally to constraint sets whose irrelevant variables have been eliminated, i.e. $con^{elim}(E)$.

6 Consistency of Empirical Programs

In Section 4, we have presented a method, based on constraint satisfaction, that can determine the consistency of sets of empirical clauses. In the previous section, we have developed

will use in our prototype system.

consistency-preserving ways to optimize this method for practical usage. In this section, we show how to extend this method to determine the consistency of empirical programs which may have non-empty contexts.

6.1 An Algorithm for Consistency Checking

Given an empirical program $\langle C, E\rangle$, recall from Definition 4 that an interpretation I is a model of the program iff I satisfies both C and E. In Section 4, we have developed a method for checking the E part of the program. Moreover, there are certainly many ways to check the consistency of the C part, like using the systems described in [3, 17]. Obviously, as shown in Example 11, the problem is that the consistency of C and E, when considered separately, does not guarantee the joint consistency of $\langle C, E\rangle$. One straightforward solution to this problem is to find a model for the C part and then test for satisfaction of the E part using Theorem 1. However, if $\langle C, E\rangle$ is jointly inconsistent, this strategy may not terminate, as C may have infinitely many models. In the following, we present a consistency checking algorithm that always terminates.

Algorithm 1 Let the input be an empirical program $P = \langle C, E\rangle$.

1. Partition the clauses in C into two sets C_1, C_2 such that C_1 consists of all non-ground clauses in C, and $C_2 = C - C_1$.
2. Initialize S to $\emptyset$. For each clause $Cl \equiv L_0 \leftarrow L_1 \wedge \ldots \wedge L_n$ in C_1, add the constraint: $(\sum_{\mathcal{P}_i \models Cfree(L_0 \wedge L_1 \wedge \ldots \wedge L_n)} v_i^{Label(L_0 \wedge L_1 \wedge \ldots \wedge L_n)} = \sum_{\mathcal{P}_i \models Cfree(L_1 \wedge \ldots \wedge L_n)} v_i^{Label(L_1 \wedge \ldots \wedge L_n)})$ to S.
3. Find an Herbrand model of C_2, using techniques such as the one described in [3].
4. If no such Herbrand model can be found, declare that the program is inconsistent and halt.
5. Otherwise, initialize the set T to $con(E) \cup S$.
6. Let M be the model computed. Then for all tuples of the form $\langle c_{i_1}, \ldots, c_{i_{ma(P)}}\rangle$ where for all $1 \leq j \leq ma(P)$, c_{i_j} is a constant symbol in $\mathcal{L}$, do the following:
   (a) Find the partition $\mathcal{P}_i$ such that $Cbind(\mathcal{P}_i^{\langle c_{i_1}, \ldots, c_{i_{ma(P)}}\rangle})$ is true in M.
   (b) Add the constraint $v_i^{\langle c_{i_1}, \ldots, c_{i_{ma(P)}}\rangle} \geq 1$ to T.
7. Determine whether the constraints in T have a solution. If they do, declare that the program is consistent and halt.
8. Otherwise, find a new Herbrand model of C_2, and go back to Step (4). $\square$

Example 13 Consider the following empirical program:

$$\begin{array}{lrcl} C: & male(bush) & \leftarrow & \\ E: & [0.5, 0.6] voted(X, bush) & \leftarrow & male(X) \\ & [0.7, 0.8] voted(X, perot) & \leftarrow & male(X) \end{array}$$

Consider the Herbrand model $M = \{male(bush)\}$ of $C_2 \equiv C$. Now there are 4 pairs to be considered in Step (6): $\langle bush, bush\rangle$, $\langle bush, perot\rangle$, $\langle perot, bush\rangle$, $\langle perot, perot\rangle$. For the first pair, since $\neg voted(bush, bush) \wedge male(bush) \wedge male(bush)$ is true in M, $\mathcal{P}_5$ is the right partition (cf. Example 8). Thus, the following constraint is added to T:

$$v_5^{\langle bush,bush\rangle} \geq 1 \qquad (1)$$

Similarly, for the other 3 pairs, the following constraints are added:

$$v_6^{\langle bush,perot\rangle} \geq 1 \qquad (2)$$
$$v_7^{\langle perot,bush\rangle} \geq 1 \qquad (3)$$
$$v_8^{\langle perot,perot\rangle} \geq 1 \qquad (4)$$

From Definition 13 and Step (5) of the algorithm above, the constraints: $v_5 \geq v_5^{\langle bush,bush\rangle}$ and $v_6 \geq v_6^{\langle bush,perot\rangle}$ are satisfied. Now coupled with Constraints (1) and (2), it is necessary that $v_5 \geq 1$ and $v_6 \geq 1$. However, based on the analysis shown in Example 11, the constraints in T corresponding to the E part can only be satisfied iff $v_1 = v_2 = v_5 = v_6 = 0$. Hence, with respect to M, no solution can be found. Note that M is the least Herbrand model of C_2, and thus $male(bush)$ must be true in every other Herbrand model M' of C_2. By an analysis similar to the one above, it is easy to see that the constraints added in Step (6b) with respect to M' require that v_1, v_2, v_5 and v_6 cannot be equal to 0 simultaneously, contradicting the constraints for E. Hence, this program is inconsistent. □

Example 14 Consider the following empirical program:

$$\begin{array}{lrcl} C: & voted(duke, duke) & \leftarrow & \\ & \neg male(duke) & \leftarrow & \\ E: & [0.65, 0.7]\ male(X) & \leftarrow & voted(X, Y) \end{array}$$

Consider the Herbrand model $M = \{voted(duke, duke)\}$ of $C_2 \equiv C$. For the pair $\langle duke, duke\rangle$ considered in Step (6), the constraint:

$$v_4^{\langle duke,duke\rangle} \geq 1$$

is added to T, which also contains the following (relevant) constraints from $con(E)$ (cf. Example 12):

$$\begin{array}{rcl} 0.65 * (v_1 + \ldots + v_4) & \leq & (v_1 + v_2) \leq 0.7 * (v_1 + \ldots + v_4) \\ v_1 + \ldots + v_8 & \geq & 1 \\ v_4 & \geq & v_4^{\langle duke,duke\rangle} \end{array}$$

All these constraints are satisfiable. For instance, a solution is: $v_4 = v_4^{\langle duke,duke\rangle} = 1$, $v_1 + v_2 = 7$ and $v_3 = 2$. Thus, the program is consistent.

This example highlights one of the major differences between the work presented here and our earlier framework based on subjective probabilities [18, 19]. There, a clause corresponding in appearance to the empirical clause in E applies to *every* element in the Herbrand domain. Thus, the subjective probability of $male(duke)$ is simultaneously 0 due to $\neg male(duke)$, and between 0.65 and 0.7 due to the clause in E. Within our subjective framework, this discrepancy would render the program inconsistent. □

In Step (1) of the above algorithm, the context C is partitioned into two sets: C_1 consisting of all non-ground clauses in C, and C_2 consisting of all ground clauses. An Herbrand model M of C_2 is computed. Then for each tuple $\langle c_{i_1}, \ldots, c_{i_{ma(P)}}\rangle$ of constant symbols, the partition such that $Cbind(\mathcal{P}_i^{\langle c_{i_1},\ldots,c_{i_{ma(P)}}\rangle})$ is true in M is found [6] (cf. Definition 14). An appropriate constraint is then added in Step (6b) to guarantee that subsequently an interpretation that corresponds to a solution of the constraints in T will be able to make ground literals L true iff L is true in M. This property is crucial in proving the soundness and completeness of Algorithm 1 (cf. Theorem 2 below).

After the appropriate constraints have been added to T, any standard (mixed integer) linear programming algorithm can be used in Step (7) to determine whether the constraints in T have a solution. This decision process is guaranteed to halt. If there is a solution, then Algorithm 1 also halts. Otherwise, a new Herbrand model of C_2 is considered. Since the number of Herbrand models of C_2 is finite, Algorithm 1 always terminates. Note that this termination property does not depend on the order the Herbrand models are considered. However, we believe that it is a good heuristic to begin with the minimal models. Techniques described in [3, 17] use this heuristic.

6.2 Soundness and Completeness of Algorithm 1

The following theorem shows that Algorithm 1 is a sound and complete procedure for determining consistency of empirical programs. The proof of the theorem makes use of the following definition and the assumption that distinct constant symbols in $\mathcal{L}$ are mapped to distinct elements in the domain of an interpretation (cf. Definition 4).

Definition 17 Let M be an Herbrand interpretation and I be any interpretation. We say that I *extends* M if the following conditions hold:
i) the domain of M is a subset of the domain of I, i.e. every constant in $\mathcal{L}$ is also in the domain of I and

[6] Given the way the partitions $\mathcal{P}_i$ are set up, and the fact that M is a (2-valued) Herbrand model, there can only be one partition that satisfies the condition that $Cbind(\mathcal{P}_i^{\langle c_{i_1},\ldots,c_{i_{ma(P)}}\rangle})$ is true in M.

ii) for each constant symbol c in $\mathcal{L}$, I assigns c to c in the domain of I and
iii) for all literals L, I makes L true iff M makes L true. $\square$

Theorem 2 Algorithm 1 is sound and complete in determining consistency of empirical programs, i.e. $\langle C, E\rangle$ is consistent iff Algorithm 1 declares that it is consistent.

Proof
i) Claim:- $\langle C, E\rangle$ is consistent if Algorithm 1 declares that it is consistent.
Since Algorithm 1 can only declare the consistency of the program in Step (7), the constraints in T, corresponding to some Herbrand model M of C_2, must have a solution S. In particular, since the constraints added in Step (6b) are also satisfied, there exists an interpretation I such that I extends M, and $S_I = S$ is a solution to the constraints in T. Now consider all the clauses in $\langle C, E\rangle$.
Case 1: Clauses in C_2 obtained in Step (1)
Let $Cl \equiv L_0 \leftarrow L_1 \wedge \ldots \wedge L_n$ be any *ground* clause in C_2. Suppose I satisfies $L_1 \wedge \ldots \wedge L_n$. Since for all literals L, I makes L true iff M makes L true, M satisfies $L_1 \wedge \ldots \wedge L_n$. Since M is a model of C_2, M satisfies L_0. Again by virtue of the fact that I extends M, I satisfies L_0. Thus, I is a model of Cl.
Case 2: Clauses in C_1 obtained in Step (1)
Let $Cl \equiv L_0 \leftarrow L_1 \wedge \ldots \wedge L_n$ be any non-ground clause in C_1. Since S_I is a solution of the constraints in T, it satisfies the constraints added in Step (2). Thus, it follows that $\sum_{\mathcal{P}_i \models Cfree(L_0 \wedge L_1 \wedge \ldots \wedge L_n)} S_I(\mathcal{P}_i^{Label(L_0 \wedge L_1 \wedge \ldots \wedge L_n)}) = \sum_{\mathcal{P}_i \models Cfree(L_1 \wedge \ldots \wedge L_n)} S_I(\mathcal{P}_i^{Label(L_1 \wedge \ldots \wedge L_n)})$. Now by Lemma 2, it follows that $\|L_0 \wedge L_1 \wedge \ldots \wedge L_n\|_I = \|L_1 \wedge \ldots \wedge L_n\|_I$. Hence, by Part (v) of Lemma 1, I satisfies the clause Cl.
Case 3: Empirical clauses in E
The constraints in $con(E)$ are added to T in Step (5). Since S_I is a solution of the constraints in T, it is a solution of $con(E)$. Hence, by Theorem 1, I satisfies all the clauses in E.
By combining all three cases above, I is a model of $\langle C, E\rangle$.

ii) Claim:- If $\langle C, E\rangle$ is consistent, Algorithm 1 declares that it is consistent.
Let I be a model of $\langle C, E\rangle$. Now consider all the constraints in T.
Case 1: Constraints in $con(E)$
Since I is a model of E, then by Theorem 1, S_I is a solution of $con(E)$.
Case 2: Constraints added in Step (2)
Corresponding to each constraint is the clause $L_0 \leftarrow L_1 \wedge \ldots \wedge L_n$ in C_1. Since I satisfies this clause, then by Part (v) of Lemma 1, it follows that $\|L_0 \wedge L_1 \wedge \ldots \wedge L_n\|_I = \|L_1 \wedge \ldots \wedge L_n\|_I$. Now by Lemma 2, it follows that $\sum_{\mathcal{P}_i \models Cfree(L_0 \wedge L_1 \wedge \ldots \wedge L_n)} S_I(\mathcal{P}_i^{Label(L_0 \wedge L_1 \wedge \ldots \wedge L_n)}) = \sum_{\mathcal{P}_i \models Cfree(L_1 \wedge \ldots \wedge L_n)} S_I(\mathcal{P}_i^{Label(L_1 \wedge \ldots \wedge L_n)})$. Hence, S_I satisfies these constraints.
Case 3: Constraints of the form $v_i^{\langle c_1, \ldots, c_{ma(P)}\rangle} \geq 1$ added in Step (6b)
Let M be an Herbrand interpretation such that for all literals L, M makes L true iff I makes L true. Since I is a model of $\langle C, E\rangle$, and hence a model of C_2, M is an Herbrand model of C_2 as well, and is considered in an iteration of Algorithm 1. Now for $\langle c_1, \ldots, c_{ma(P)}\rangle$, let

$Conj$ denote the ground literal $Cbind(\mathcal{P}_i^{\langle c_1,\ldots,c_{ma(P)}\rangle})$. Since $Conj$ is true in M, I also makes $Conj$ true. Therefore, it is the case that $\langle d_1,\ldots,d_{ma(P)}\rangle$ is in the subpartition $\mathcal{P}_i^{\langle c_1,\ldots,c_{ma(P)}\rangle}$, where for all $1 \leq j \leq ma(P)$, c_j is mapped to d_j in I. Under the assumption that distinct constant symbols are mapped to distinct elements in the domain of I, it is necessary that $S_I(\mathcal{P}_i^{\langle c_1,\ldots,c_{ma(P)}\rangle}) \geq 1$. Hence, I satisfies these constraints.
By combining all the cases above, S_I is a solution of T. Hence, in Step (7), Algorithm 1 declares that $\langle C, E\rangle$ is consistent. This completes the proof of the theorem. $\Box$

The corollary below states that, given a pre-interpretation J, Algorithm 1 can be extended to check for the $J-$consistency of empirical programs.

Corollary 3 Let J be a pre-interpretation with domain D. When applied with the additional constraint $\sum_{i=1}^{t} v_i = \|D\|^{ma(P)}$ in T, Algorithm 1 is sound and complete in determining the $J-$consistency of empirical program P. $\Box$

Note that Algorithm 1 can be optimized for practical usage using the same techniques described in Section 5. With those optimizations, Algorithm 1 provides a feasible, as well as sound and complete, way of verifying consistency of empirical programs.

7 Query Processing for Consistent Empirical Programs

7.1 Outline for Query Answering

In most logic programming frameworks, including the ones we proposed in [18, 19], queries are existential in nature. Here, queries to an empirical program $\langle C, E\rangle$ are *different*. A query is of the form: $Q \equiv L$ which intuitively asks for the *conditional probability* of L, given that the program is true. As a preview, we first outline a two-step procedure that can be used to answer this query; the procedure will be formalized in Section 7.3.

In the first step, the query answering procedure poses the query against the context C. If the context can deduce the truth or falsity[7] of the query, then the procedure returns the answer 1 or 0 respectively, and the processing for the query is completed. Otherwise, when no definite answer to the query can be *deduced*, the procedure then tries to *induce* the conditional probability by consulting the empirical clauses in E as illustrated below.

Example 15 Consider writing an empirical program for the University of British Columbia Department of Computer Science Hyperbrochure, which is a system for the automatic delivery of video information. More specifically, the system manages a one-hour video disk containing overview material intended for faculty members, students, staff and visitors. When a user first logs onto the system, the system asks the user for his/her profile, based on which the

[7] As we are only interested in answering queries to consistent empirical programs, an atom cannot be both true and false in the context.

system determines and delivers the parts of the video most interesting to the user. [8] To do so, the following empirical program P may be used by the system:

$$\begin{array}{lrcl}
C: & student(X) & \leftarrow & gradStudent(X) \\
 & adult(X) & \leftarrow & gradStudent(X) \\
E: & [0.6, 0.7]\ interest(X,Y) & \leftarrow & student(X) \wedge aboutFacilities(Y) \\
 & [0.8, 1]\ interest(X,Y) & \leftarrow & gradStudent(X) \wedge aboutFunding(Y) \\
 & [0.5, 0.7]\ interest(X,Y) & \leftarrow & adult(X) \wedge aboutHistory(Y) \\
 & [0.9, 1]\ interest(X,Y) & \leftarrow & \neg adult(X) \wedge presentation(Y, colorAnimation)
\end{array}$$

Consider posing the query $Q_1 \equiv interest(paul, Y)$ to $P \cup \{student(paul)\}$. Then by the first empirical clause, the system determines it is quite likely that all items Y about facilities may be of interest to *paul*. On the other hand, suppose that *paul* has used the system before, and has explicitly stated that he is interested in all color animation items. Then in the context of P, the system may contain the (non-empirical) clause: $interest(paul, Y) \leftarrow presentation(Y, colorAnimation)$. In this case, query Q_1 can be answered directly from the context, without induction from the empirical part, and the system shows *paul* all color animation items.

Now consider posing the query $Q_2 \equiv interest(mary, Y)$ to $P \cup \{gradStudent(mary)\}$. Since the context cannot be used to answer the query, the system uses the empirical clauses. By the second empirical clause, the system determines it is very likely that *mary* would like to view all items Y about funding. Moreover, because all graduate students are students, as specified by the first clause in the context, the first empirical clause is also applicable to *mary*. Thus, the system may also show *mary* the items about facilities. Similarly, since all graduate students are adults, the system may choose to show *mary* items about the history of the department and the university. □

Query Q_2 above highlights a major issue involved in the kind of inductive answering we wish to support – the choice of answers when more than one empirical clause (i.e. inductive answer) is applicable. In the above example, the system needs to choose from items about facilities, funding or history. The approach we take to resolve such conflicts is the one customarily used in statistical inferences – choose the one with the most specific *reference class*. In our example, since $gradStudent(X)$ implies $student(X)$ and $adult(X)$, $gradStudent$ is the most specific reference class. Thus, the system will show items about funding. For more discussion on reference classes, see [14]. As many researchers have observed [1, 22], changing reference classes can lead to non-monotonic modes of reasoning. For instance, if $student(mary)$ is the only fact about *mary*, then the system will show items about facilities. However, if the additional fact $gradStudent(mary)$ is included, the items of interest may change immediately.

[8] The full system allows the user to give feedback to it which can then adjust, if necessary, what to show the user next. Such feedback provides a basis for updating the probabilities used by the system.

7.2 Compilation of Empirical Programs

The following algorithm uses empirical clauses and clauses in the context to generate other empirical clauses so that query processing can be simplified. As the generation process is query-independent, this algorithm should be carried out at compile-time.

Algorithm 2 Let $P = \langle C, E \rangle$ be an empirical program.

1. Set T_0 to E and i to 1.
2. Construct the set $S_1 = \{ [1-c_2, 1-c_1] \neg L_0 \leftarrow L_1 \wedge \ldots \wedge L_n \mid [c_1, c_2]\ L_0 \leftarrow L_1 \wedge \ldots \wedge L_n$ is a clause in $T_{i-1}\}$.
3. Construct the set $S_2 = \{[0,0] L_1 \leftarrow L_0 \mid [0,0] L_0 \leftarrow L_1$ is a clause in $T_{i-1}\}$.
4. Construct the set $S_3 = \{ [c_1, 1]\ L' \leftarrow L_1 \wedge \ldots \wedge L_n \mid [c_1, c_2]\ L_0 \leftarrow L_1 \wedge \ldots \wedge L_n$ is a clause in T_{i-1} and $L' \leftarrow L_0$ is a logical consequence of $C\}$.
5. Set $T_i = T_{i-1} \cup S_1 \cup S_2 \cup S_3$. If T_i is the same as T_{i-1}, then set $comp(P) = \langle C, T_i \rangle$ and halt. Otherwise, increment i and go to Step 2. □

Example 16 Apply Algorithm 2 to the program listed in Example 15. It is not difficult to see that in the first iteration, S_1 consists of 4 clauses, such as $[0.3, 0.4] \neg interest(X, Y) \leftarrow student(X) \wedge aboutFacilities(Y)$. S_2 and S_3 are empty. In the second iteration, nothing else is generated, and the compilation ends. Note that given $student(X) \leftarrow gradStudent(X)$ and $[0.6, 0.7] interest(X, Y) \leftarrow student(X) \wedge aboutFacilities(Y)$, one may wonder whether there is a non-trivial empirical clause of the form: $[c_1, c_2] interest(X, Y) \leftarrow gradStudent(X) \wedge aboutFacilities(Y)$. The answer is no, because in general it is possible to have $c_1 = 0$ and $c_2 = 1$ simultaneously. □

Checking whether T_i is the same as T_{i-1} in Step 5 of Algorithm 2 ensures that the algorithm starts the next iteration only if at least one new empirical clause is added in the current iteration. Since $\mathcal{L}$ is a finite language, there can only be a finite number of empirical clauses of the forms generated above. Thus, Algorithm 2 terminates after a finite number of iterations, producing a finite compiled program $comp(P)$. The lemma below shows that the generated clauses do not change the original meaning of P.

Lemma 6 Let $P = \langle C, E \rangle$ be an empirical program. Then P and $comp(P)$ are logically equivalent.

Proof Outline Since all clauses in P are in $comp(P)$, it is sufficient to show that all models of P are models of $comp(P)$. Let I be a model of P. Now proceed by induction on i to show that I is a model of clauses in T_i. The base case is trivial (cf. Step 1 of Algorithm 2). Now for $i > 0$, by the induction hypothesis, I is a model of T_{i-1}. It suffices to consider clauses contained in S_1, S_2 and S_3. As the proofs for clauses in S_1 and S_2 are quite trivial, below we only show the proof for clauses in S_3.

Consider a clause $Cl \equiv [c_1, 1]L' \leftarrow L_1 \wedge \ldots \wedge L_n$ in S_3. It is generated from the clauses $Cl_1 \equiv [c_1, c_2]L_0 \leftarrow L_1 \wedge \ldots \wedge L_n$ in T_{i-1} and $Cl_2 \equiv L' \leftarrow L_0$ which is a logical consequence of context C. I is a model of both Cl_1 and Cl_2. There are two cases. First, if $\|L_1 \wedge \ldots \wedge L_n\|_I = 0$, then by Definition 9, I is trivially a model of Cl. On the other hand, if $\|L_1 \wedge \ldots \wedge L_n\|_I > 0$, it follows from Definition 9 that $c_1 * \|L_1 \wedge \ldots \wedge L_n\|_I \leq \|L_0 \wedge L_1 \wedge \ldots \wedge L_n\|_I \leq c_2 * \|L_1 \wedge \ldots \wedge L_n\|_I$. Given Cl_2, by Lemma 1, it is the case that $\|L_0 \wedge L_1 \wedge \ldots \wedge L_n\|_I \leq \|L' \wedge L_1 \wedge \ldots \wedge L_n\|_I$. Thus, it is necessary that $c_1 * \|L_1 \wedge \ldots \wedge L_n\|_I \leq \|L' \wedge L_1 \wedge \ldots \wedge L_n\|_I$. Hence, I is a model of Cl. This completes the proof of the induction and the lemma. □

7.3 A Basic Algorithm for Query Answering

Thus far, we have discussed how an empirical program can be compiled to facilitate query answering. Next we will give a declarative specification of correct answers to queries, and will then present a procedure that formalizes the approach for query answering outlined in Section 7.1. Recall that inductive answering involves choosing the most specific reference class. However, in general, as reference class may only follow a partial order in specificity, there may not be a unique most specific reference class. The notion of *maximally preferred class* introduced below deals with the situation when there are several maximally specific reference classes.

Definition 18 Let C be the context of an empirical program. Let S be $\{Body_1, \ldots, Body_n\}$, where $Body_i \equiv L_1^i \wedge \ldots \wedge L_{m_i}^i$.
i) $Body_i$ is *more preferred* than $Body_j$ iff either $Body_i$ is an instance of $Body_j$ or $Body_j \leftarrow Body_i$ is a logical consequence of C.
ii) $Body_i$ is a *maximally preferred class* in S if there is no element $Body_j$ in S such that $Body_j$ is more preferred than $Body_i$. □

We are now in a position to specify declaratively the set of correct answers for a given query. In the following, we will first focus on ground queries $Q \equiv L$. In Section 7.6, we will discuss how to handle non-ground queries.

Definition 19 Let $P = \langle C, E\rangle$ be an empirical program, and L be a ground query. Let $consq(P, L)$ be defined as follows:
1) If either L or $\neg L$ is a logical consequence of C, then $consq(P, L) = \{[1, 1]\}$ or $consq(P, L) = \{[0, 0]\}$ respectively.
2) Otherwise, $consq(P, L)$ is the set of ranges $[c_1, c_2]$ such that:

i) there exists a maximally preferred class $Body_i$ in the set $\{Body|\ L' \leftarrow Body$ is an empirical clause in P, and there exists a unifier θ that unifies L and L', and that $Body\theta$ is a logical consequence of $C\}$; and

ii) $[c_1, c_2]$ is the tightest range so that $c_1 * \|Body_i\|_I \leq \|Body_i \wedge L'\|_I \leq c_2 * \|Body_i\|_I$ is satisfied for all models I of P. □

The following algorithm computes the set of correct answers for queries. Lemma 7 will show that the algorithm is sound with respect to $consq(P, L)$ defined above.

Algorithm 3 Let L be a ground query and $comp(P) = \langle C, E\rangle$ be the compiled version of an empirical program.

1. If L is a logical consequence of C, return [1,1] as the probability range, and halt.
2. If $\neg L$ is a logical consequence of C, return [0,0] as the probability range, and halt.
3. (Inductive answering begins) Construct the set $S = \{Cl|\ Cl \equiv [c_1, c_2]L' \leftarrow Body$ is an empirical clause in E, and there exists a unifier θ that unifies L and L' and that $Body\theta$ is a logical consequence of $C\}$. Intuitively S consists of all the empirical clauses applicable to query L. If S is empty, halt [9].
4. Otherwise, based on C, compute all the maximally preferred classes in $\{Body|\ [c_1, c_2]L' \leftarrow Body$ is in $S\}$. (For empirical clauses of the form $[c_1, c_2]L' \leftarrow$, $Body \equiv true$.)
5. For each maximally preferred class $Body_i$, return $[c_{1,i}, c_{2,i}]$ as the probability range based on $Body_i$, where the clause $[c_{1,i}, c_{2,i}]L' \leftarrow Body_i$ is in S. □

Example 17 Consider posing the query $Q \equiv interest(mary, clip_1)$ to the following empirical program:

$$\begin{array}{lrcl} C: & gradStudent(mary) & \leftarrow & \\ & student(X) & \leftarrow & gradStudent(X) \\ E: & [0.6, 0.7]\ interest(X, clip_1) & \leftarrow & student(X) \\ & [0.8, 1]\ interest(X, clip_1) & \leftarrow & gradStudent(X) \end{array}$$

Obviously, Steps 1 and 2 of Algorithm 3 do not give any definite answer to the query. In Step 3, S consists of both empirical clauses. In Step 4, the computed maximally preferred class in $\{student(X), gradStudent(X)\}$ is $gradStudent(X)$. Thus, Step 5 returns the range [0.8,1] (not the range [0.6,0.7]) as the answer. Further suppose that having seen $clip_1$, $mary$'s feedback is that she is not interested in $clip_1$. Thus, the fact $\neg interest(mary, clip_1)$ may be added to the context. Then next time when the query $interest(mary, clip_1)$ is posed, Step 2 of Algorithm 3 returns the range [0,0] as the answer. □

Note that Algorithm 3 requires a procedure for checking logical consequences. Such checking can be implemented by a standard unsatisfiability checker, such as one based on resolution, or a mixed integer programming algorithm such as the one described in [3].

[9] This is equivalent to returning [0,1] (i.e. "unknown") as the answer. However, for the ease of presentation of the algorithm and the soundness proof later on, we would rather the algorithm halts without returning any range.

7.4 Soundness of Algorithm 3

Given a compiled empirical program $comp(P)$ and a ground query L, we use the notation $proof(P, L)$ to denote the set of ranges obtained by applying Algorithm 3 to $comp(P)$ and L.

Lemma 7 Let $comp(P) = \langle C, E \rangle$ be an empirical program and L be a ground query. Then for any range $[c_1, c_2] \in proof(P, L)$, there exists a range $[d_1, d_2] \in consq(P, L)$ such that $[d_1, d_2] \subseteq [c_1, c_2]$.

Proof Outline Case 1: L is a logical consequence of C
By Step 1 of Algorithm 3, it is necessary that $proof(P, L) = \{[1, 1]\}$. Furthermore, by Definition 19, it is the case that $consq(P, L) = \{[1, 1]\} = proof(P, L)$.

Case 2: $\neg L$ is a logical consequence of C
Similar to Case 1 above, it is obvious that $consq(P, L) = \{[0, 0]\} = proof(P, L)$.

Case 3: Otherwise
Consider any range $[c_1, c_2] \in proof(P, L)$. [10] According to Step 5 of Algorithm 3, there exists a maximally preferred class $Body_i$ such that the clause $[c_1, c_2]L' \leftarrow Body_i$ is in the set S constructed in Step 3. Now for all models I of P, by Definition 9, it must be the case that $c_1 * \|Body_i\|_I \leq \|L' \wedge Body_i\|_I \leq c_2 * \|Body_i\|_I$. Thus, by Definition 19, there exists a range $[d_1, d_2] \in consq(P, L)$ such that $[d_1, d_2] \subseteq [c_1, c_2]$. □

The lemma above shows that Algorithm 3 is sound. However, the example below shows that Algorithm 3 may not be complete. By Definition 19, we say that Algorithm 3 is complete iff the algorithms finds all the *tightest* ranges contained in $consq(P, L)$, i.e. $consq(P, L) \subseteq proof(P, L)$.

Example 18 Consider the following empirical program P:

$$
\begin{array}{llcl}
C: & D(X) & \leftarrow & A(X) \wedge B(X) \\
 & A(X) & \leftarrow & D(X) \\
 & B(X) & \leftarrow & D(X) \\
 & F(X) & \leftarrow & \neg A(X) \wedge B(X) \\
 & \neg A(X) & \leftarrow & F(X) \\
 & B(X) & \leftarrow & F(X) \\
E: & [c_1, c_2]D(X) & \leftarrow & C(X) \\
 & [d_1, d_2]F(X) & \leftarrow & C(X)
\end{array}
$$

For any model I of the program P, it is necessary that $c_1 * \|C(X)\|_I \leq \|D(X) \wedge C(X)\|_I \leq c_2 * \|C(X)\|_I$. But according to the first three clauses in the context, it is the case that $D(X) \leftrightarrow A(X) \wedge B(X)$. In other words, it must be the case that $c_1 * \|C(X)\|_I \leq \|A(X) \wedge$

[10] If $proof(P, L)$ is empty, this corresponds to the situation when the "returned answer" is [0,1]. In this case, the theorem is trivially true.

$B(X) \wedge C(X)\|_I \leq c_2 * \|C(X)\|_I$. Similarly, based on the second empirical clause and the last three clauses in the context, it is necessary that $d_1 * \|C(X)\|_I \leq \|\neg A(X) \wedge B(X) \wedge C(X)\|_I \leq d_2 * \|C(X)\|_I$. However, since $\|B(X) \wedge C(X)\|_I = \|A(X) \wedge B(X) \wedge C(X)\|_I + \|\neg A(X) \wedge B(X) \wedge C(X)\|_I$, it must be the case that $(c_1+d_1) * \|C(X)\|_I \leq \|B(X) \wedge C(X)\|_I \leq (c_2+d_2) * \|C(X)\|_I$. In other words, I satisfies the clause $[(c_1+d_1), (c_2+d_2)] B(X) \leftarrow C(X)$. However, it is obvious that this clause cannot be generated by the compilation carried out by Algorithm 2. Hence, if the query is $L \equiv B(d)$ for some constant symbol d, then $proof(P \cup \{C(d)\}, L)$ as computed by Algorithm 3 does not contain the range $[(c_1 + d_1), (c_2 + d_2)]$ which is, however, contained in $consq(P \cup \{C(d)\}, L)$. □

The above example highlights one of the major reasons why Algorithm 3 is not complete. That is, in order to get the tightest ranges, query answering may require very intricate reasoning involving both empirical clauses and clauses in the context. Moreover, this process may in general involve multiple clauses. The compilation carried out by Algorithm 2 is certainly not powerful enough. However, for a procedure to get all the tightest ranges, it may need to reason with *all* possible combinations of clauses. This is a process we believe is too expensive even to be conducted at compile-time. Hence, the current compilation procedure, as specified in Algorithm 2, tries to strike a balance between efficiency and completeness by generating clauses that do not appear in the original program and that are easy to be produced. In fact, our query answering algorithm is sound but incomplete for both P and $comp(P)$. The only difference is that the "degree" of incompleteness for the latter is less than that of the former, as $P \subseteq comp(P)$.

Guntzer et al [9] develops a probabilistic calculus that can handle non-monotonic uncertainty reasoning. While they do not provide a model semantics for their framework, they prove the soundness of their calculus. However, for reasons similar to the ones cited above, their calculus may not be complete.

7.5 An Enhanced Algorithm for Query Answering: Ranking Maximally Preferred Classes

Example 19 Consider the following empirical program which generalizes the program discussed in Example 17.

$$\begin{array}{rrcl}
C: & gradStudent(mary) & \leftarrow & \\
 & student(X) & \leftarrow & gradStudent(X) \\
 & aboutFacilities(clip_1) & \leftarrow & \\
 & aboutFunding(clip_1) & \leftarrow & \\
E: & [0.6, 0.7]\ interest(X, Y) & \leftarrow & student(X) \wedge aboutFacilities(Y) \\
 & [0.8, 1]\ interest(X, Y) & \leftarrow & gradStudent(X) \wedge aboutFunding(Y)
\end{array}$$

When Algorithm 3 is applied to answer the query $interest(mary, clip_1)$, the set S constructed in Step 3 now becomes $\{Body_1 \equiv student(X) \wedge aboutFacilities(Y),\ Body_2 \equiv gradStudent(X) \wedge aboutFunding(Y)\}$. Even though it is still true that $gradStudent(X)$

implies $student(X)$, the implication $Body_1 \leftarrow Body_2$ is no longer a logical consequence of the context. Thus, neither $Body_1$ nor $Body_2$ is more preferred than the other, and there are now two maximally preferred classes. □

The above example demonstrates that the notion of $Body_i$ being more preferred than $Body_j$ may be too strong for many situations, thus permitting too many maximally preferred classes. In the above example, even though $Body_1$ and $Body_2$ are two maximally preferred classes, it is reasonable to argue that $Body_2$ is more applicable than $Body_1$. This is due to the fact that $gradStudent(X)$ implies $student(X)$ and that $aboutFacilities(Y)$ and $aboutFunding(Y)$ do not entail one another. In the following, we define a way to rank maximally preferred classes. We begin with the definition below.

Definition 20 Let C be the context of an empirical program. Given $Body_i \equiv \{L_1^i, \ldots, L_{m_i}^i\}$ and $Body_j \equiv \{L_1^j, \ldots, L_{m_j}^j\}$, $Body_i$ is *more applicable* than $Body_j$ if there exist a subset $B \subseteq Body_i$ and a literal $L \in Body_j$ such that $L \leftarrow \bigwedge_{L_u \in B} L_u$ is a logical consequence of C. □

Example 20 Suppose the context C_1 consists of $\{L_1 \leftarrow L_2\}$. If $Body_1 \equiv L_1 \wedge L_3$ and $Body_2 \equiv L_2 \wedge L_4$, then $Body_2$ is more applicable than $Body_1$. On the other hand, suppose the context C_2 is $\{L_1 \leftarrow L_2, L_3 \leftarrow L_4\}$, and $Body_1 \equiv L_1 \wedge L_4$ and $Body_2 \equiv L_2 \wedge L_3$. Then because of $L_3 \leftarrow L_4$, $Body_1$ is more applicable than $Body_2$. However, because of $L_1 \leftarrow L_2$, $Body_1$ is also more applicable than $Body_2$. In other words, it is possible to have $Body_i$ and $Body_j$ more applicable than one another simultaneously. Furthermore, given the context C_2, consider $Body_1 \equiv L_4$, $Body_2 \equiv L_2 \wedge L_3$, and $Body_3 \equiv L_1$. It is the case that $Body_1$ is more applicable than $Body_2$, which in turn is more applicable than $Body_3$. However, it is not true that $Body_1$ is more applicable than $Body_3$. In other words, the relationship of being more applicable is not transitive. □

The above example shows that the notion of $Body_i$ being more applicable than $Body_j$ is weaker than the notion of $Body_i$ being more preferred than $Body_j$. In particular, for a set $\{Body_1, \ldots, Body_n\}$, the notion of applicability does not constitute a partial ordering on the elements of the set. Thus, this notion alone is not powerful enough to rank maximally preferred classes. However, the following definition proposes a heuristic way to rank maximally preferred classes.

Definition 21 Let $\{Body_1, \ldots, Body_n\}$ be a collection of maximally preferred classes.
i) For all $1 \leq i \leq n$, define $more(Body_i)$ to be the cardinality of the set $\{Body_j \mid Body_i$ is more applicable than $Body_j\}$.
ii) For all $1 \leq i \leq n$, define $less(Body_i)$ to be the cardinality of the set $\{Body_j \mid Body_j$ is more applicable than $Body_i\}$.
iii) For all $1 \leq i \leq n$, define $rank(Body_i) = more(Body_i) - less(Body_i)$. □

If there are n maximally preferred classes, $rank(Body_i)$ ranges from $n-1$ to $-(n-1)$. A more careful analysis indicates that $rank(Body_i) = n-1$ and $rank(Body_i) = -(n-1)$ are two special cases. In the former case, this is only possible if for all $j \neq i$, $Body_i$ is more applicable to $Body_j$, but not vice versa. Thus, it is arguable that $Body_i$ is the "most applicable." Conversely, $rank(Body_i) = -(n-1)$ is only possible if for all $j \neq i$, $Body_j$ is more applicable than $Body_i$, but not vice versa. In this case, $Body_i$ is the "least applicable," and can therefore be ignored. These observations lead to the following algorithm which enhances Algorithm 3 by providing a heuristic ranking on the maximally preferred classes.

Algorithm 4 Same as Algorithm 3, except that Step 5 of Algorithm 3 is modified to:

5. Suppose $Body_1, \ldots, Body_n$ are the maximally preferred classes.
   (a) Compute $rank(B_i)$ for all $1 \leq i \leq n$.
   (b) If there exists $Body_i$ such that $rank(Body_i) = n-1$, return $[c_{1,i}, c_{2,i}]$ as the probability range based on $Body_i$, where the clause $[c_{1,i}, c_{2,i}]L' \leftarrow Body_i$ is in S (constructed in Step 3). Halt.
   (c) Otherwise, for all $1 \leq i \leq n$ such that $rank(Body_i) > -(n-1)$, return in descending order of ranks, the range $[c_{1,i}, c_{2,i}]$ as the probability range based on $Body_i$, where the clause $[c_{1,i}, c_{2,i}]L' \leftarrow Body_i$ is in S. Halt. □

Example 21 Consider the following empirical program which is very similar to the programs discussed in Examples 15 and 19.

$$
\begin{array}{lrcl}
C: & gradStudent(mary) & \leftarrow & \\
 & student(X) & \leftarrow & gradStudent(X) \\
 & adult(X) & \leftarrow & gradStudent(X) \\
 & aboutFacilities(clip_1) & \leftarrow & \\
 & aboutFunding(clip_1) & \leftarrow & \\
 & aboutHistory(clip_1) & \leftarrow & \\
E: & [0.6, 0.7]\ interest(X,Y) & \leftarrow & student(X) \wedge aboutFacilities(Y) \\
 & [0.8, 1]\ interest(X,Y) & \leftarrow & gradStudent(X) \wedge aboutFunding(Y) \\
 & [0.5, 0.7]\ interest(X,Y) & \leftarrow & adult(X) \wedge aboutHistory(Y)
\end{array}
$$

Following the discussion in Example 19, it is not difficult to see that there are three maximally preferred classes: $Body_1 \equiv student(X) \wedge aboutFacilities(Y)$, $Body_2 \equiv gradStudent(X) \wedge aboutFunding(Y)$, and $Body_3 \equiv adult(X) \wedge aboutHistory(Y)$. But now according to Step 5a of Algorithm 4, it is the case that $rank(Body_2) = 2$ because of the second and third clauses in the context. Thus in Step 5b, the algorithm only returns the range [0.8,1] and halts. □

As a final note, computing the ranks of maximally preferred classes may be a time-consuming task to be carried out at run-time. One optimization would be to compute at compile-time a table A whose entry $A_{i,j}$ indicates whether $Body_i$ is more applicable than $Body_j$, for all bodies $Body_i, Body_j$ of clauses in $comp(P)$. Then at run-time, the computation of $rank(Body_i)$ would amount to simple table look-ups.

7.6 Query Answering for Non-ground Queries

Thus far, we have only considered answering ground queries. The following algorithm modifies Algorithm 4 to handle non-ground queries.

Algorithm 5 Let L be a query, not necessarily ground, and $comp(P) = \langle C, E \rangle$ be the compiled version of an empirical program.

1. For all most general unifiers θ, if $L\theta$ (or $\neg L\theta$) is a logical consequence of C, return θ and [1,1] (or [0,0] respectively), and halt.
2. Same as Steps 3 and 4 of Algorithm 4 (and Algorithm 3).
3. Same as Step 5 of Algorithm 4 except that the unifiers θ are returned with the corresponding probability ranges. $\square$

Example 22 Suppose the context of the program discussed in Example 21 is modified to:

$$
\begin{array}{lrcl}
C: & gradStudent(mary) & \leftarrow & \\
 & student(X) & \leftarrow & gradStudent(X) \\
 & adult(X) & \leftarrow & gradStudent(X) \\
 & aboutFacilities(clip_1) & \leftarrow & \\
 & aboutFunding(clip_2) & \leftarrow & \\
 & aboutHistory(clip_3) & \leftarrow &
\end{array}
$$

Suppose the query is $Q \equiv interest(mary, Y)$. Following the discussion in Example 21, there are the same three maximally preferred classes: $Body_1 \equiv student(X) \wedge aboutFacilities(Y)$, $Body_2 \equiv gradStudent(X) \wedge aboutFunding(Y)$, and $Body_3 \equiv adult(X) \wedge aboutHistory(Y)$. And again, because $rank(Body_2) = 2$, Algorithm 5 returns the range [0.8,1] with $Y = clip_2$, and halts. $\square$

Note that in Algorithm 5, ranking maximally preferred classes is not conducted on a per unifier basis. If that was the case, the algorithm would return the three answers for the above example: range = [0.6,0.7], $Y = clip_1$; range = [0.8,1], $Y = clip_2$; and range = [0.5,0.7], $Y = clip_3$. However, we believe that conducting the ranking across the maximally preferred classes of *all* unifiers corresponds more closely to the kind of query answering based on the most specific reference classes outlined in Section 7.1. This is because the notion of a class being more specific than another is a concept that is based on the classes themselves, but not on individual elements in the classes.

8 Related Work

There have been many proposals on multivalued logic programming. These include the works by Baldwin [2], Blair and Subrahmanian [4], Dubois, Prade and Lang [5], Fitting [7, 8], Kifer et al [11, 12, 13], Shapiro [25] and van Emden [26]. However, all of these proposals are non-probabilistic, as they are based either on fuzzy set theory, possibilistic logic or Dempster-Shafer theory. As we believe that a probabilistic approach to quantitative deduction in logic programming is important, we have proposed a framework for probabilistic deductive databases [18, 19]. This framework is based on a subjectivistic view of probabilities, that is viewing probabilities as degrees of belief held by a particular agent. More specifically, in technical terms, the framework is based on regarding Herbrand interpretations as possible worlds and attaching probabilities to closed formulas in the language. However, it is incapable of expressing statistical generalizations.

In contrast, the framework presented here is intended to express empirical probabilities that represent statistical truths about the world, or at least about statistical samples drawn from the world. In other words, these probabilities are objective in nature, independent of the beliefs of an agent. More technically speaking, while in the framework studied in [18, 19], probability distributions are defined over the sentences of logical languages, here probability distributions are defined over the domain of discourse. Example 14 highlights the major difference between these two approaches. Within our subjective framework, a clause corresponding in appearance to the empirical clause in E applies to *every* element in the Herbrand domain. Thus, the subjective probability of $male(duke)$ is simultaneously 0 due to $\neg male(duke)$, and between 0.65 and 0.7 due to the clause in E. Within our subjective framework, this discrepancy would render the program inconsistent.

The framework presented here generalizes our framework reported in [20] which only supports *unary* predicate symbols. Thus, our language here allows us to express and reason with relationships among groups of elements in the domain of discourse. To deal with this generality and gain in expressive power, we need to adopt a more sophisticated notion of partitions, introduce the notion of subpartitions, and handle the existence of combinations of variable symbols (cf. the material covered in Section 4 here). Furthermore, we present in this paper various ways to optimize our consistency checking framework for practical usage, and develop a query answering procedure that can handle non-ground queries and rank maximally preferred classes.

The integration of logic and probability theory has been the subject of numerous studies [1, 21, 22]. More relevant to our work here is Bacchus' framework that extends full first order logic with empirical probability statements [1]. He develops a sound and complete proof procedure for consistent theories. We provide explicit mechanisms for determining the consistency of empirical programs. These mechanisms are based upon (mixed integer) linear programming techniques, and may be implemented on top of standard (integer) linear programming packages.

9 Conclusions

In this paper, we investigate how to incorporate empirical probabilities in deductive databases. We propose a framework whereby an empirical deductive database consists of a context and a collection of empirical clauses. For such databases, we develop a model-theoretic semantics, and a sound and complete algorithm for checking consistency. Combined with the optimization techniques proposed in this paper, this algorithm can be readily implemented by linear programming methods. Last but not least, we develop query processing procedures which can support inductive answering, and can rank multiple answers to queries heuristically. In ongoing work, we are developing a prototype implementation of the query answering procedures, and will integrate the prototype with the Hyperbrochure discussed in Example 15.

References

[1] F. Bacchus. (1988) *Representing and Reasoning with Probabilistic Knowledge*, Research Report CS-88-31, University of Waterloo.

[2] J.F. Baldwin. (1987) *Evidential Support Logic Programming*, Journal of Fuzzy Sets and Systems, 24, pps 1-26.

[3] C. Bell, A. Nerode, R.T. Ng and V.S. Subrahmanian. (1992) *Implementing Deductive Databases by Mixed Integer Programming*, to appear in: ACM Transactions of Database Systems. Preliminary version appeared in: Proc. 11th Symposium on Principles of Database Systems, pp 283–292.

[4] H. A. Blair and V.S. Subrahmanian. (1987) *Paraconsistent Logic Programming*, Theoretical Computer Science, 68, pp 35-54.

[5] D. Dubois, H. Prade and J. Lang. (1991) *Towards Possibilistic Logic Programming*, Proc. 1991 Intl. Conf. on Logic Programming, ed. K. Furukawa, pps 581–595, MIT Press.

[6] J. E. Fenstad (1980) *The Structure of Probabilities Defined on First-Order Languages*, in: Studies in Inductive Logic and Probabilities Volume 2, ed. R. C. Jeffrey, pp 251–262, University of California Press.

[7] M. C. Fitting. (1988) *Logic Programming on a Topological Bilattice*, Fundamenta Informaticae, 11, pps 209–218.

[8] M. C. Fitting. (1991) *Bilattices and the Semantics of Logic Programming*, Journal of Logic Programming, 11, 2, pp 91–116.

[9] U. Guntzer, W. Kiesling and H. Thone. (1991) *New Directions For Uncertainty Reasoning in Deductive Databases*, Proc. ACM SIGMOD, pp. 178–187.

[10] J. Han, Y. Cai and N. Cercone. (1992) *Knowledge Discovery in Databases: An Attribute-Oriented Approach*, Proc. 18th VLDB Conference, pp 547–559.

[11] M. Kifer and A. Li. (1988) *On the Semantics of Rule-Based Expert Systems with Uncertainty*, 2-nd Intl. Conf. on Database Theory, Springer Verlag LNCS 326, pp 102–117.

[12] M. Kifer and E. Lozinskii. (1989) *RI: A Logic for Reasoning with Inconsistency*, 4-th Symposium on Logic in Computer Science, Asilomar, CA, pp. 253-262.

[13] M. Kifer and V. S. Subrahmanian. (1992) *Theory of Generalized Annotated Logic Programming and its Applications*, Journal of Logic Programming, 12, 4, pp 335–367.

[14] H. E. Kyburg, Jr. (1983) *The Reference Class*, Philosophy of Science, 50, 3, pp 374–397.

[15] J.W. Lloyd. (1987) *Foundations of Logic Programming*, Springer.

[16] D. Luenberger. (1984) *Linear and Nonlinear Programming*, Addison-Wesley.

[17] A. Nerode, R.T. Ng and V.S. Subrahmanian. (1992) *Computing Circumscriptive Databases, Part I: Theory and Algorithms*, to appear in: Information and Computation.

[18] R.T. Ng and V.S. Subrahmanian. (1989) *Probabilistic Logic Programming*, Information and Computation, 101, 2, pp 150–201. Preliminary version in: Proc. 5th International Symposium on Methodologies for Intelligent Systems, pp 9-16.

[19] R.T. Ng and V.S. Subrahmanian. (1990) *A Semantical Framework for Supporting Subjective and Conditional Probabilities in Deductive Databases*, to appear in: Journal of Automated Reasoning. Preliminary version in: Proc. 1991 Intl. Conf. on Logic Programming, ed. K. Furukawa, pps 565–580, MIT Press.

[20] R.T. Ng and V.S. Subrahmanian. (1992) *Empirical Probabilities in Monadic Deductive Databases*, Proc. 8th Conf. on Uncertainty in Artificial Intelligence, pp 215–222.

[21] N. Nilsson. (1986) *Probabilistic Logic*, AI Journal 28, pp 71–87.

[22] J. Pearl. (1988) *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*, Morgan Kaufmann.

[23] G. Piatetsky-Shapiro and W. Frawley. (1991) *Knowledge Discovery in Databases*, MIT Press.

[24] Hans Reichenbach. *Theory of Probability*, University of California Press.

[25] E. Shapiro. (1983) *Logic Programs with Uncertainties: A Tool for Implementing Expert Systems*, Proc. IJCAI '83, pps 529–532, William Kauffman.

[26] M.H. van Emden. (1986) *Quantitative Deduction and its Fixpoint Theory*, Journal of Logic Programming, 4, 1, pp 37-53.