INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA

UMI®
800-521-0600

TOWARDS A GENERALIZED THEORY OF DEDUCTIVE DATABASES WITH UNCERTAINTY

NEMATOLLAAH SHIRI-VARNAAMKHAASTI

A THESIS
IN
THE DEPARTMENT
OF
COMPUTER SCIENCE

PRESENTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF DOCTOR OF PHILOSOPHY
CONCORDIA UNIVERSITY
MONTRÉAL, QUÉBEC, CANADA

AUGUST 1997

Abstract

Towards a Generalized Theory of Deductive Databases with Uncertainty

Nematollaah Shiri-Varnaamkhaasti, Ph.D.
Concordia University, 1997

Uncertainty management is identified as a challenging issue in databases [SSU91]. Logic database programming, with its declarative advantage and its powerful top-down and bottom-up query processing techniques has attracted the attention of researchers, and numerous logic frameworks with uncertainty have been proposed over the last decade. On the basis in which uncertainty is associated with the facts and rules in a program, we classify the approaches taken to uncertainty in these frameworks into the *annotation based* (AB) and *implication based* (IB). In the AB approach, certainties are associated with the atoms in the rules and facts, while in the IB approach, they are associated with the implications in the programs.

In this thesis, we attempt to address the aforementioned challenging issue. To this end, we take an axiomatic approach and study the relevant issues: (1) the language aspects, (2) query optimization, (3) the termination behaviors and data complexity of the bottom-up evaluations of logic programs with uncertainty, (4) the expressive power, and (5) the ease and efficiency of implementing such languages. Each of these issues is discussed in a chapter of this thesis, in that order. Although our focus in this study is on the IB approach, the insights provided and the lessons learned are useful in a more general context of deduction with uncertainty.

In order to study these issues in a framework independent manner, we first identify a number of "reasonable" properties which are normally satisfied by combination functions in programs with uncertainty. We will then propose a language, called *the parametric framework*, where the parameters are the combination functions defined on the underlying certainty lattice. The proposed framework uses *multisets* as the basis of the structure of the semantics, as opposed to using *sets*, which are often used.

This is the key point to the expressive power of our framework, which unifies and generalizes the IB approach to uncertainty. With this framework as a basis, we study the other relevant issues mentioned and establish various results. A main advantage of our axiomatic approach in studying the various issues in the thesis is that it makes the results applicable to a wide range of (IB) frameworks. Our top-down and bottom-up implementations of the parametric framework show that the ideas in the thesis are practical which lend themselves to an-easy-to-use and efficient "environment" for deduction with uncertainty at large.

Acknowledgments

In the name of Allah, *the Compassionate, the Merciful*

I am very grateful to my thesis supervisors, Professors Laks V.S. Lakshmanan and Fereidoon Sadri for their guidance, support, and encouragement throughout the development of this thesis. Dr. Lakshmanan introduced me to the world of mathematical logic, logic programming, and knowledge-bases and helped me expand my knowledge in those areas to the point at which this thesis has become possible. He has guided me trough the numerous stages in the development of this thesis, and I am indebted to him for that. I consider myself fortunate to work with him and always enjoyed the many discussions I had with him.

Dr. Sadri, my "cyber advisor", also was a great support during the development of this thesis. Despite being far away, he was always there to help me via the e-mail communications and during his visits to Concordia. I thank him for all his support.

I am grateful to Dr. David S. Warren, the chair of the department of Computer Science, State University of New York at Stony Brook, for helping me develop the inference engine of our top-down implementation in XSB of the language developed in the thesis. It was a great opportunity for me to learn a lot from him.

I would also like to thank Dr. Raghu Ramakrishnan, at the Computer Science department at the University of Wisconsin at Madison, for supporting the idea of extending the database language CORAL, developed in his research group, with user-defined aggregation needed in the bottom-up implementation of our language. Thanks also to Shaun Flisakowski, the CORAL database programmer, for providing the extended CORAL.

I would like to express my sincere gratitude to the analyst group of the Computer Science Department for their timely response and support whenever their help

was needed. In particular, I would like to thank Stan Swiercz, who can now install the CORAL system with his eyes closed! Thanks also to Michael Assels and William Wong for their many helps. Also, I would like to thank the department secretaries, especially Edwina Bowen, Monica Etwaroo, Halina Monkiewicz, and Stephanie Roberts for being very helpful and cooperative.

My study at Concordia has been financially supported by the following grants which I would like to gladly acknowledge. I was awarded an scholarship from the *Ministry of Culture and Higher Education* of the *Islamic Republic of Iran* for pursing my study, for which I feel in debt. I would also like to thank my supervisors for their financial support: Dr. Lakshmanan supported me from his NSERC and FCAR grants and Dr. Sadri from his NSERC grant.

I would like to thank my fellow graduate students Alanoly Andrews, Ramesh Achuthan, Gokul Chander, Fred Gingras, Hasan Jamil, Darmalingum Muthiayen, Iyer Subramanian, Raymund Teo, and Ioana Ursu with whom I shared much time in classrooms, offices, biking trips, conferences, etc. I had so many discussions on technical and non-technical topics with Andrews, Jamil, and Subbu. Andrews also helped me when I had questions on English idiom and grammar. Khaled Jababo was always there to help in any ways he could, for which I thank him.

Finally, I would like to thank my parents, *Mr. Hossein Shiri* and *Mrs. R. Farahmand*, my wife, *Farah*, and my children, *Raheleh*, *Vahid* and *Mona*, for their endless love and support during my studies. In particular, I would like to thank my mother without whose encouragement, I would not pursue my post graduate studies. In appreciation, I dedicate this thesis to my parents and my family.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

> Life is the art of drawing sufficient conclusions from insufficient premises.
> — Samuel Butler

Most real-life applications require an ability to represent, manage, and reason with uncertain knowledge. Examples includes weather forecasting, diagnostics, image processing applications, legal and military applications, data mining, risk analysis for banks, satellite data analysis, stock market predictions, and the like. For instance, in the FIST image database system [CRSS95], queries about the identity of persons in a photograph are answered by comparing it with a mugshot database, in the form of assertions like "*the person in the frame corresponding to bottom left corner (5.6) and top right corner (6.8) is john with probability 0.5 to 0.6, and peter with probability 0.4 to 0.55*". In the QBIC system [NBEY93], a query such as "*find all red objects in a certain image*" would return a set of objects together with a weight indicating their degree of "redness". Typical data mining algorithms generate rules which have associated metrics called *confidence* and *support*, essentially uncertainty measures [AIS93, HCC93]. Answering *complex queries* against such applications requires that certainties associated with answers to simple queries be *combined* using *well-grounded principles* and in a *meaningful way*. For instance, the Garlic multi-media information system developed at IBM Almaden [CHS+95] processes complex queries by dispatching subqueries to component subsystems and then by *combining* the answers coming from them.

In a recent study [SSU91], Silberchatz, Stonebraker, and Ullman identify uncertainty management as one of the important future challenges in database research: "Further research [in *uncertainty*] is essential, as we must learn not only to cope with data of *limited reliability*, but do so *efficiently*, with *massive amounts of data*". Practical considerations dictate that a framework used for manipulating uncertain knowledge be easy to program in, and admit efficient implementation and efficient computations.

Intuitively, a piece of data is *uncertain* if its truth is not established definitely. Logic database programming, with its advantages of being declarative and modular, and with its powerful top-down and bottom-up query processing techniques has attracted the attention of researchers, and numerous frameworks for deductive databases with uncertainty have been proposed [DLP91, Fit88, Fit91, KL88, KS92, Lak94, LS94a, LS94b, NS91, NS92, NS93, Sha83, Sub87, van86]. Typically, these proposals offer a framework in which deduction can be combined with some form of uncertainty (including, e.g., certainty values, fuzzy, probabilities, possibilities, etc.). As in the classical logic programming, these frameworks offer a declarative semantics of programs. On the operational side, this is supported by a sound and complete (or weakly complete) proof procedure and a corresponding fixpoint semantics.

The underlying uncertainty formalisms in the proposed frameworks include probability theory [Lak94, LS94a, NS92, NS93], fuzzy set theory [Sha83, van86], multi-valued logics [Fit88, Fit91, KL88, KS92], possibilistic logic [DLP91], evidence theory [Bal87, NS91], and hybrid (that is, a combination of numerical and non-numerical) formalisms [Lak94, LS94b, Sad91].

Let us consider the following example, illustrating an application program defining uncertain facts and rules.

Example 1.0.1 Let P be a logic program containing information related to a health organization. The underlying certainty lattice in P is the set $\mathcal{T} = \mathcal{C}[0,1]$ of closed intervals in $[0,1]$, partially ordered by $\preceq$, where $[\alpha,\beta] \preceq [\gamma,\delta]$ iff $\alpha \preceq \gamma$ and $\beta \preceq \delta$. The "combination" functions or modes in this example are *positive correlation* (*pc*), *ignorance* (*ign*), and *independence* (*ind*), defined as follows. Suppose E_1 and E_2 are any ground atoms, which we view as events. Then, the positive correlation mode intuitively indicates that the occurrences of E_1 and E_2 overlap as much as possible.

The most general situation is the ignorance mode, which indicates that nothing is known or assumed about the interaction between E_1 and E_2, while it is the opposite in the independence mode, indicating that it is known or assumed that the (non-)occurrence of one event does not affect that of the other one. A formal definitions of these modes of conjunction and disjunction used in this example are as follows, adapted from [LS94a]. Let $\alpha = [a_1, a_2]$ and $\beta = [b_1, b_2]$ be any certainty elements in $\mathcal{T}$. Then,

$$\begin{aligned} \vee_{ign}(\alpha, \beta) &= [max(a_1, b_1),\ min(1, a_2 + b_2)] \\ \vee_{pc}(\alpha, \beta) &= [max(a_1, b_1),\ max(a_2, b_2)] \\ \wedge_{pc}(\alpha, \beta) &= [min(a_1, b_1),\ min(a_2, b_2)] \\ \wedge_{ind}(\alpha, \beta) &= [a_1 b_1,\ a_2 b_2]. \end{aligned}$$

When in this program a rule body includes just a single atom, we may use the *identity* function i as the conjunction function, where $i(\alpha) = \alpha$, for all $\alpha \in \mathcal{T}$.

The organization has information about the connectivity of the areas in a region along with the "degrees" of proximity, expressed as follows.

$$\begin{aligned} r_1 &: close(a, b) \xleftarrow{[0.9,1]}; && \langle \vee_{ign}, \wedge_{pc}, - \rangle. \\ r_2 &: close(a, c) \xleftarrow{[0.8,1]}; && \langle \vee_{ign}, \wedge_{pc}, - \rangle. \\ r_3 &: close(b, c) \xleftarrow{[0.7,1]}; && \langle \vee_{ign}, \wedge_{pc}, - \rangle. \\ r_4 &: close(Y, X) \xleftarrow{[1,1]} close(X, Y); && \langle \vee_{ign}, \wedge_{pc}, i \rangle. \end{aligned}$$

The triple $\langle f^d, f^p, f^c \rangle$ associated with a rule denotes, respectively, the disjunction, propagation, and conjunction functions. The first rule says that area a is "close" to area b with confidence 0.9, at least. Rule r_4 says closeness is definitely a symmetric property. The next rule asserts that there is an outbreak of a particular disease d in the area a.

$$r_5 : outbreak(d, a) \xleftarrow{[1,1]}; \qquad \langle \vee_{pc}, \wedge_{pc}, - \rangle.$$

An area X is connected to Y, provided X is close to Y, or there is some area Z close to X which is connected to Y. This can be expressed as follows.

$$r_7 : connected(X, Y) \xleftarrow{[0.8,0.9]} close(X, Z),\ connected(Z, Y); \quad \langle \vee_{pc}, \wedge_{pc}, \wedge_{pc} \rangle.$$

Finally, we use affected(D, A) to denote that area A is affected by disease D. The

rules defining this predicate are as follows.

$$r_8: \quad affected(D, A) \xleftarrow{[0.7,0.8]} outbreak(D, A); \qquad \langle \vee_{pc}, \wedge_{pc}, \wedge_{pc} \rangle.$$

$$r_9: \quad affected(D, A) \xleftarrow{[0.8,0.9]} connected(A, A_1),\ affected(D, A_1); \qquad \langle \vee_{pc}, \wedge_{pc}, \wedge_{pc} \rangle.$$

The reason why *closeness*, defined by r_1 to r_4, is an uncertain concept is that, in general, there could be several factors considered by an expert defining it. For instance, if a and b are two areas separated by some natural obstacles, such as a mountain, then the certainty assigned to the predicate $close(a, b)$ would be higher than when no such obstacles exist. This certainty might vary depending on the kind of obstacles there are, e.g. river, forest, and also depending on the weather condition, since a disease may spread easier to farther areas in a hot climate. Such details are abstracted away by the expert by assigning an "appropriate" certainty element to the rules and facts defining the closeness. Note that the distance between two areas could also be a factor considered in assigning certainties to rules and facts. For instance, the certainty of $connected(X, Y)$, defined by r_6, is less than the certainty assigned to this predicate by r_7, intuitively because "closer" areas are "more connected" than areas separated by other areas and/or by natural obstacles. In other words, the farther two areas, the lower the degree of their connectivity. ■

1.1 Approaches to Uncertainty

In this section, we study the approaches to uncertainty in logic programming and deductive databases. There are three ways, we note, in which the proposed frameworks of uncertainty differ: (i) in their underlying notion of uncertainty, (ii) the way in which uncertainties are manipulated, and (iii) the way in which uncertainty is associated with the facts and rules of a program. On the basis of (iii), we classify the approaches to uncertainty in these frameworks into what we call the *annotation based* (AB, for short) and the *implication based* (IB).

In the AB approach, a rule r is an assertion of the form:

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

which says "the certainty of A is at least (or is in) $f(\beta_1, \ldots, \beta_n)$, whenever the certainty of B_i is at least (or is in) β_i, $1 \leq i \leq n$." Here f is a computable n-ary function, and β_i is an annotation constant or an annotation variable ranging over an appropriate certainty domain. Examples of the AB frameworks include the annotated logic programming of Subrahmanian [Sub87], Kifer and Li [KL88], the probabilistic logic programming of Ng and Subrahmanian [NS91, NS92, NS93], and the generalized theory of annotated logic programming (GAP) proposed by Kifer and Subrahmanian [KS92]. In [Sub94], Subrahmanian shows that annotated logics may be used for amalgamating multiple knowledge bases when these knowledge bases (possibly) contain inconsistencies, uncertainties, and non-monotonic mode of negation, and develops a unified theoretical framework based on this idea.

In the IB approach, a rule r is an assertion of the form:

$$r: \quad A \xleftarrow{\alpha} B_1, \ldots, B_n$$

which says "the certainty of the implication $B_1, \ldots, B_n \rightarrow A$ is α", or "the certainty that the conjunction $B_1, \ldots, B_n$ implies A is α". The certainty α associated with the implication in the rule in a sense controls or filters the "propagation" of truth from the rule body to the head. Such a rule is evaluated as follows, given an assignment of certainties to B_i's. First, the certainty of the rule body is computed using the "conjunction function" associated with the rule. The value obtained is then combined with the rule certainty α, using the propagation function associated with the rule. This yields a certainty for the atom in the rule head. Alternate derivations of the same atom, obtained possibly from different rules, are then "combined" into a single certainty for that atom, using the disjunction function associated with the predicate symbol of the atom. This is done for every atom and at every step in a bottom-up evaluation of a rule set. The process goes on to some iteration at which no "new" atom is derived. Examples of the IB frameworks include van Emden [van86], Fitting [Fit88, Fit91], Dubois et al. [DLP91], Escalada-Imaz and Manyà [EM95], Lakshmanan and Sadri [LS94a, LS94b], and Lakshmanan [Lak94]. Examples 1.2.2 to 1.2.7 illustrate some of these frameworks informally.

Let us briefly compare the AB and IB approaches to uncertainty. While the way implication is treated in the AB approach is closer to the classical framework of logic programming, the way rules are fired in the IB approach is similar to the classical

case and hence has a definite intuitive appeal [KS92]. In terms of expressive power, the AB approach is strictly more expressive than the IB, in general. For instance, it is shown in [KS92] that the GAP framework, through annotation variables, can simulate the IB framework of van Emden. In fact, we note that *any* IB framework can be simulated within an "appropriately" defined AB framework which uses multisets as the basis of its semantics. In Chapter 5, we introduce such an AB framework and compare its expressive power with our IB approach. Hähnle [H h96] also discusses the expressive power of IB vs. AB in the context of many-valued logics. Comparing the two approaches from the point of view of query processing, the AB approach is more involved, since unification requires taking into account the annotations constants and variables in a program, and also resolution requires constraint solving, in general. Leach and Lu [LL96] discuss query processing in the AB approach.

Another difference between the two approaches is the fixpoint operator T_P, which is continuous in the IB frameworks but not so in some AB frameworks. An important consequence of this is that the bottom-up fixpoint evaluation of programs in IB frameworks would always terminate in at most ω steps. With respect to fixpoint evaluation, our experience with a number of IB frameworks suggests that evaluating large classes of programs in these frameworks could be accomplished in time polynomial in the size of the input database (e.g. see [Lak94, LS94a, LS94b, Shi93]. In Chapter 4, we provide an analysis of termination and complexity of the fixpoint evaluations for programs with the IB approach. Our analysis is useful also for programs with the AB approach.

We do not enter into a debate of which form of uncertainty is the best. Rather, our contention is that different forms may be appropriate for different applications. Furthermore, *different* ways of *manipulating* uncertainty may be required in *one* application (See Example 1.0.1, for an illustration). Considering the differences mentioned above between the AB and IB approaches, we believe that the IB approach is easier to program in and is more amenable to efficient implementation, since it can benefit more readily from the existing query processing and query optimization techniques, such as unification and resolution procedures, developed for standard logic programming and deductive databases.

While there have been numerous proposals for deductive databases with uncertainty, unfortunately there has been very little progress on (1) the relationship between these frameworks mainly from the point of view of semantics, (2) query optimization techniques/tools, (3) the termination and data complexity analysis of fixpoint evaluations of programs with uncertainty, (4) the expressive power of these frameworks, and (5) their ease and efficient implementations. These challenging issues are particularly relevant, given the importance of them in typical data intensive applications of deductive databases technology. The question is: *"how can we study these problems so that the results obtained and the tools developed will be applicable and useful over a spectrum of frameworks for deductive databases with uncertainty?"*

Rather than study these issues for individual frameworks, we believe an "axiomatic" approach to address them would prove useful and provide insights. Given the usefulness of manipulating uncertainty in more than one manner for a given application (e.g. see Example 1.0.1), a "framework independent" and unified study such as ours is likely to be far more beneficial.

In this thesis, we attempt to address the aforementioned challenging issue of uncertainty management in deductive databases. To this end, we proposed an extension of the standard deductive database framework, *datalog*, and developed a language for deduction with uncertainty, which we called the *parametric framework* [LS96b]. The proposed framework unifies and generalizes the IB approach to uncertainty. With this as a basis, we studied the problem of containment of parametric conjunctive queries in this framework [LS96c, LS97]. Following our approach, we study the other related issues enumerated (3) to (5) in the above, each discussed in a chapter in this thesis.

The rest of this chapter is organized as follows. We illustrate the ideas behind the parametric framework and motivate our approach in Section 1.2. We explain the parameters in Section 1.3. Section 1.4, collects together the relevant preliminary notions. The contributions of the thesis are enumerated in Section 1.5, and the outline is provided in Section 1.6.

We assume the reader is familiar with the elements of logic programming as discussed in Lloyd [Llo87] or Apt [Apt86]. Enderton [End72] is an excellent introduction to mathematical logic. For the principles of database and knowledge-base systems see Ullman [Ull89]. Ceri et al. [CGT89] contains what one may want to know about

datalog, the standard framework of deductive databases.

1.2 Motivating the Parametric Framework

In this section, we illustrate the ideas behind the parametric framework, and motivate the axiomatic approach in our study which resulted in a unification and generalization of the IB frameworks to deductive databases with uncertainty. The task in the unification is (i) to permit various forms of uncertainty to be manipulated in different ways, and (ii) to allow for the fact that certain manipulations amount to treating different derivations of an atom as a *set* (e.g. [DLP91, Fit88, Fit91, LS94b, Lak94, van86]) while others amount to treating it as a *multiset* (e.g. [BS75, LS94a]).

Example 1.2.1 Let P be a "template" IB program with the following rules/facts:

$$r_1: \; A \xleftarrow{\alpha_1} B.$$
$$r_2: \; A \xleftarrow{\alpha_2} C.$$
$$r_3: \; B \xleftarrow{\alpha_3} .$$
$$r_4: \; C \xleftarrow{\alpha_4} .$$

where A, B, C are ground atoms[1]*, and α_i is the certainty associated with rule r_i, for $1 \leq i \leq 4$. Let $\mathcal{T}$ denote the underlying certainty lattice of P. Examples 1.2.2 - 1.2.7 illustrate instances of this program each of which in an IB framework.* ■

Example 1.2.2 (Datalog) Let $\mathcal{T} = \{0, 1\}$ be the set of truth values, and $\alpha_i = 1$, for $1 \leq i \leq 4$. Suppose both the propagation and conjunction functions associated with r_i are *min*, and the disjunction function associated with every predicate symbol is *max*. Then P is a program in the standard framework of logic programming and deductive databases. ■

Example 1.2.3 (Dubois et al. [DLP91]) Let the certainty domain be the unit interval, that is, $\mathcal{T} = [0, 1]$. Suppose $\alpha_1 = 0.8$, $\alpha_2 = \alpha_3 = 0.7$, and $\alpha_4 = 0.8$ are possibility/necessity degrees associated with the rules. Also suppose the conjunction, propagation, and disjunction functions are as in the previous example. Then, P is

[1] That is, P is in propositional logic, which is simple but good enough to illustrate the idea.

a program in the framework proposed by Dubois et al. [DLP91], founded on the possibility theory, proposed by Zadeh [Zad78]. In the fixpoint semantics of P, the possibility degrees obtained for A, B, and C are $0.7, 0.7$, and 0.8, respectively. ■

Example 1.2.4 (van Emden [van86]) Let $\mathcal{T} = [0,1]$, and suppose the rule certainly α_i is as defined in the previous example, for $1 \leq i \leq 4$. Also suppose the conjunction and disjunction functions are as before, but the propagation function is $*$, the product. Then P is a program in van Emden's framework[2] [van86], which is mathematically founded on the fuzzy set theory, proposed by Zadeh [Zad65]. In the least fixpoint of P, the certainties associated with atoms A, B, and C are respectively 0.56, 0.7, and 0.8. ■

Example 1.2.5 (MYCIN [BS75]) Let $\mathcal{T} = [0,1]$, and suppose α_i's are probability values defined as in the previous example. Suppose the propagation and conjunction functions associated with every rule in P is $*$, and the disjunction function associated with every predicate symbol in P is f, where $f(\alpha, \beta) = \alpha + \beta - \alpha\beta$. Viewing each ground atom as an event, f returns the probability of the occurrence of any event, where different occurrences of the same event are assumed to be independent, in the probabilistic sense. The MYCIN system [BS75] uses f as the disjunction function.

Let us now consider a fixpoint evaluation of P. In the first iteration, we derive B and C with probabilities 0.7 and 0.8, respectively. In the second iteration, there are two derivations of A, one by r_1 with probability 0.56 and the other by r_2 also with probability 0.56. Therefore, the overall probability of A at iteration 2 is $f(0.56, 0.56) = 0.8064$. Note that in this example, collecting derivations as a multiset is crucial; if the derivations were collected as a set, then the probability of A obtained would be 0.56 — an incorrect result. ■

Example 1.2.6 (Lakshmanan and Sadri [LS94b]) This is a deductive database obtained by extending the *Information Source Tracking method* (IST, for short) proposed by Sadri [Sad91]. In an IST database, there is a fixed number, say $k \geq 1$, of sources/agents contributing to the information in the database. The certainty domain $\mathcal{T}$ is thus defined as a set of source vectors of the form $(a_1 \ \ldots \ a_k)$, where $a_i \in L =$

[2]Function symbols are allowed in [van86], but in the context of databases, we only consider the function-free fragment of that framework, whenever we refer to it.

$\{\perp, -1, 1, \top\}$. The partial ordering $\prec$ on L is defined as follows: $\perp \prec \{-1, 1\} \prec \top$. When $a_i = 1$ in a source vector associated with a tuple in a relation, it indicates that source i has confirmed the tuple. Similarly, when a_i is -1, $\perp$, or $\top$, it indicates that source i's contribution to that tuple is negative, none, or contradictory, respectively. For a concrete example, suppose there are three sources, i.e., $k = 3$. Also suppose the certainties associated with the rules in P are as follows: $\alpha_1 = (\perp\ 1\ \perp)$, $\alpha_2 = (1\ \perp\ \perp)$, $\alpha_3 = (\perp\ \perp\ 1)$, and $\alpha_4 = (1\ 1\ \perp)$. The conjunction function used in this framework is $\overset{s}{\wedge}$, and is defined as follows. Given the source vectors $u = (a_1\ a_2\ a_3)$ and $v = (b_1\ b_2\ b_3)$, the conjunction of u and v is $u \overset{s}{\wedge} v = (\oplus(a_1, b_1)\ \oplus(a_2, b_2)\ \oplus(a_3, b_3))$, and for any sets S_1 and S_2 of source vectors, $S_1 \overset{s}{\wedge} S_2 = \{u \overset{s}{\wedge} v \mid u \in S_1 \text{ and } v \in S_2\}$, where $\oplus$ is the join operator on the certainty lattice $\mathcal{T}$. We define the propagation function associated with every rule in P to be the same as the conjunction function, $\overset{s}{\wedge}$. The disjunction function defined in this framework is $\overset{s}{\vee}$, which is essentially the set union. A fixpoint evaluation of P yields A, B, and C with certainties $\{(\perp\ 1\ 1),\ (1\ 1\ \perp)\}$, $\{(\perp\ \perp\ 1)\}$, and $\{(1\ 1\ \perp)\}$, respectively. ■

Example 1.2.7 (Lakshmanan and Sadri [LS94a]) Let $\mathcal{T} = \mathcal{C}[0,1] \times \mathcal{C}[0,1]$. Each element in $\mathcal{T}$, called a confidence (level), is a pair of closed interval in $[0,1]$. If $\alpha = \langle [a_1, a_2],\ [b_1, b_2] \rangle$ is the confidence associated with an atom, say A, it means that it is "believed" the probability that "A is true" lies in the interval $[a_1, a_2]$, and the probability that "A is false" lies in $[b_1, b_2]$.

Suppose we have $\alpha_1 = \alpha_4 = \langle [0.7, 0.8],\ [0.1, 0.2] \rangle$, $\alpha_2 = \langle [0.8, 0.95],\ [0.05, 0.15] \rangle$, and $\alpha_3 = \langle [0.9, 0.95],\ [0, 0.15] \rangle$. Also suppose the conjunction function associated with r_1 and r_2 is $\wedge_{pc}$, and the disjunction function associated with A is $\vee_{ind}$, where pc stands for the "positive correlation" mode, and ind for the "independence", in the probabilistic sense. These modes are defined as follows. (See [LS94a] for a full explanation of these modes.) Let $\alpha = \langle [a_1, a_2],\ [a_3, a_4] \rangle$ and $\beta = \langle [b_1, b_2],\ [b_3, b_4] \rangle$ be any intervals in $\mathcal{T}$. Then,

$$\wedge_{pc}(\alpha, \beta) = \langle [min(a_1, b_1),\ min(a_2, b_2)],\ [max(a_3, b_3),\ max(a_4, b_4)] \rangle$$

and

$$\vee_{ind}(\alpha, \beta) = \langle [a_1 + b_1 - a_1 b_1,\ a_2 + b_2 - a_2 b_2],\ [a_3 b_3,\ a_4 b_4] \rangle.$$

Let us consider a fixpoint evaluation of P. Initially, each atom is assigned the least confidence, $\langle[0,0],\ [1,1]\rangle$ in $\mathcal{T}$, which corresponds to the truth value *false* in the classical logic. Then, in step 1, we derive B and C with confidences $\langle[0.9,0.95],\ [0,0.15]\rangle$ and $\langle[0.7,0.8],\ [0.1,0.2]\rangle$, respectively. In step 2, we obtain two derivations of A with the same confidence, $\langle[0.7,0.8],\ [0.1,0.2]\rangle$, which when combined, using $\vee_{ind}$, we obtain $\langle[0.91,0.96],\ [0.01,0.04]\rangle$ as A's confidence in this step. In step 3, no new/better fact is derived, and hence P's evaluation terminates. As in Example 1.2.5, the role of multisets here is crucial; if the derivations were collected as sets, we would have obtained only one copy of A at iteration 2, resulting in an incorrect confidence level for A.

Note that this is not to suggest that a user in such a framework is forced to conceive of certainty as multisets. However, because of the way the fixpoint evaluation proceeds, using different derivations, we may deduce a fact with the same certainty more than once, which suggests that, in general, we need to collect the derived facts as multisets, as opposed to sets, conventionally done. ■

Example 1.0.1 shown earlier in this chapter illustrates the expressive power of the parametric framework developed in this thesis. This program could not be expressed in any of the existing IB frameworks. A point in this example is that *different* ways of manipulating uncertainty are combined within *one* program. That is, fuzzy and probabilistic concepts for instance, could be manipulated in one framework uniformly: the semantics is built into the combination functions used in the program.

1.3 Parameters

The idea of the parametric framework inspired from our observation that a user in a program in an IB framework specifies, implicitly or explicitly, the following notions or *parameters*, as we call them.

1. The certainty domain, which we denote by $\mathcal{T}$. As usual, we assume that $<\mathcal{T}, \preceq, \otimes, \oplus>$ is a complete lattice, where $\preceq$ is the partial order on $\mathcal{T}$, $\otimes$ is the meet operator in the lattice and $\oplus$ is the join. (For concepts in lattice theory Birkhof [Bir67] is an excellent source.) The elements of $\mathcal{T}$ could be certainty factors (e.g. [van86]), probability points, vectors of elements or sets of

such vectors (e.g. [Lak94, LS94b]), (pairs of) probability ranges (e.g. [LS94a]), etc. The certainty domain $\mathcal{T}$ could also be a bilattice, as in [Fit88, Fit91]. Independent of the structure and the semantics of the certainty elements in $\mathcal{T}$, we refer to them as *certainty values* or *truth* values, interchangeably. Once the parameters in our framework are defined and fixed, the structure and the semantics of the certainty values would be defined and fixed as well.

2. The family $\mathcal{F}_c = \{f_i^c\}_{i \in \mathcal{I}_c}$ of "conjunction" functions allowed, where $\mathcal{I}_c$ is an index set. Associated with each rule in the program is a function, f_i^c, in $\mathcal{F}_c$ which "combines" the certainties of the atoms in the rule body and returns the certainty of the rule body as a whole. Since rule bodies contain a number of atoms, we model conjunction functions as mappings from finite multisets over $\mathcal{T}$ to $\mathcal{T}$.

3. The family $\mathcal{F}_p = \{f_i^p\}_{i \in \mathcal{I}_p}$ of "propagation" functions allowed, where $\mathcal{I}_p$ is an index set. Associated with each rule in the program is a function, f_i^p, in $\mathcal{F}_p$ which combines the certainty of the rule body with the certainty of the rule itself to compute the certainty of the rule head. Thus, f_i^p controls the propagation of truth from the rule body to the head.[3] Propagation functions are binary functions on $\mathcal{T}$.

4. The family $\mathcal{F}_d = \{f_i^d\}_{i \in \mathcal{I}_d}$ of "disjunction" functions allowed, where $\mathcal{I}_d$ is an index set. Associated with each predicate symbol p in the program is a function, f_i^d, in $\mathcal{F}_d$ which combines a number of certainties associated with (alternate derivations of) a ground p-atom[4] and returns a single certainty for that p-atom. That is, disjunction functions are mappings from finite multisets over $\mathcal{T}$ to $\mathcal{T}$.

1.4 Preliminaries

In this section, we quickly review some basic terms and concept from *datalog*, the standard language for deductive databases. We will then introduce some terms and concepts we need in this work and fix our notations.

[3] In an AB framework, the role of conjunction and propagation functions is played by the so called *certainty* functions.

[4] An atom whose predicate symbol is p.

Datalog is the the standard framework for deductive databases, which has been designed and intensively studied over a decade. Ceri et al. [CGT89] is an excellent survey of research on datalog. Another source is Ullman [Ull89]. Syntactically, datalog is a subset of Prolog, and thus each program in datalog can be evaluated by a Prolog interpreter. A *datalog* program is a collection of function-free Horn clauses, each of which is an statement of the form:

$$r: \quad A \leftarrow B_1, \ldots, B_n$$

where A and B's are atomic formulas and $n \geq 0$. We call such statements as rules. In rule r above, A is called the rule *head* and $B_1, \ldots, B_n$ the *body*. A *fact* is a special case of a rule where $n = 0$. We assume that rules are range restricted. That is, each variable in the head of a rule must appear in the rule body.

Suppose P is a datalog program. An IDB predicate is a predicate appearing in some rule head in P. If a predicate appears in some rule body but in no rule head, then it is an EDB predicate. The *extensional database* (EDB, for short) is a set of relations for the EDB predicates. The *intensional database* (IDB) is a set of relations for the IDB predicates. Each relation, whether IDB or EDB, is a set of ground atoms. The IDB relations are defined by applying the rules in P to the EDB relations. Therefore, every datalog program P has two components, an EDB and an IDB. The EDB is the input of P and the IDB is the output. The collection of the EDB and IDB relations is called a *database*. In our context of deductive databases with uncertainty, a database is a set of *annotated* tuples, each of which is of the form $t : \alpha$, where t is the data or tuple, and α is the certainty associated with t. We use $\mathcal{T}$ to denote the set of certainty elements, to which we may also refer as the certainty domain.

1.4.1 Notations and Conventions

Sometimes the number of times an element occurs in an unordered collection matters. A *multiset*, also called a *bag*, is an unordered collection of elements where an element can occur as a member more than once. Formally, let B be any set. Then a *multiset* X over B is a mapping from B to $\mathcal{N} = \{0, 1, 2, 3, \ldots\}$. We call B as the *base* set of X. One can think of a multiset X over B as a set of annotated elements of the

form $a : m$, where $m \in \mathcal{N}$ denotes the number of times the element a of B occurs in X. We use $\dot{\in}$ to denote the membership relation for multisets; for any $a \in B$ and any $m \in \mathcal{N}$, if the multiset X contains m occurrences of a, we write $(a : m) \dot{\in} X$ or $a \dot{\in} X$. If $x = (a : m) \in X$, we call a as the *basic part* of x, and m as the *multiplicity* of a in X. By default, the multiplicity of every element in B not present in X is 0. A set is a special multiset in which every element is annotated with 1 (or 0), as its multiplicity. To distinguish between multisets and sets, we will use $\{\!\{ \ldots \}\!\}$ for multisets. The empty multiset, denoted $\dot{\emptyset}$, is the multiset in which the multiplicity of every element in B is 0. Let X, Y be any finite multisets over a base set B. We say X is "multiset-contained" in Y, denoted $X \dot{\subseteq} Y$, provided $\forall a \in B$, the number of copies of a in X is no more than that in Y.

In the context of logic programs and deductive databases with uncertainty, we are interested in multisets over $B = B_P \times \mathcal{T}$, where B_P is the Herbrand base of the given logic program P and $\mathcal{T}$ is the set of certainty values used. In this case, if X is a multiset over B, then every element in X is of the form $(A, \alpha) : m$, where $A \in B_P$ is a ground atom, $\alpha \in \mathcal{T}$ is a certainty associated with A, and $m \in \mathcal{N}$ is the multiplicity of the basic part (A, α) in X. The multiset union, denoted $\dot{\cup}$, is an operator which retains the duplicates. That is, given multisets X, Y, their multiset union is a multiset defined as

$$X \dot{\cup} Y = \{\!\{ a : k \mid (a : m) \dot{\in} X,\ (a : n) \dot{\in} Y,\ k = m + n \}\!\}.$$

A disjunction function associated with a predicate p "translates" a multiset of uncertainties associated with a ground p-atom into a single certainty. This translation, which may elsewhere be called *normalization* (e.g. in [KKTG93]), is defined as follows. Given a multiset X over $B_P \times \mathcal{T}$, the *normalization* of X amounts to defining a multiset $X^{\#}$ whose content is "equivalent" to X and in which the multiplicity of every element is 1 (that is, $X^{\#}$ is essentially a set). Formally stated, suppose $p_1, \ldots, p_k$ are all the predicate symbols mentioned in a logic program with uncertainty. Also suppose f_i is the disjunction function associated with p_i, $1 \leq i \leq k$. Note that p_i's are distinct while f_i's need not be so. For an atom A, we denote by $\pi(A)$ the predicate symbol of A. Then,

$$X^{\#} = \{\!\{ (A : \beta) : 1 \mid (A : \alpha_j) : m_j \dot{\in} X,\ 1 \leq j \leq \ell_A,\ \pi(A) = p_i,\ \beta = f_i(Y) \}\!\}$$

where Y is a multiset of certainties associated with A containing m_j copies of α_j, for $1 \leq j \leq \ell_A$.

Our conventions for using the various symbols are as follows. The lower case letters represent predicate symbols and constants terms. We use the upper case letters $A, B, \ldots$ from the beginning of the alphabet to represent ground atoms, and $X, Y, \ldots$ to represent multiset (of certainties) as well as the variables in the predicate arguments. The particular usage will be clear from the context. We use the lower case Greek letters $\alpha, \beta, \ldots$ to represent elements in the certainty domain $\mathcal{T}$. For an atom A, we denote by $\pi(A)$ the predicate symbol of A. We will use $\preceq$ to denote the partial order on the certainty lattice $\mathcal{T}$. We will also use the orderings $\prec$, $\succ$, and $\succeq$ whose meanings are obvious. Since we will study the containment of conjunctive queries in our framework (in Chapter 3), we will use bold upper case letters $\mathbf{Q}$'s and $\mathbf{R}$'s to denote conjunctive queries in the parametric framework. If Q_1 and Q_2 are conventional conjunctive queries, we write $Q_1 \subseteq Q_2$, as usual, to mean that Q_1 is *contained* in Q_2. For $\mathbf{Q}_1$,$\mathbf{Q}_2$ in our framework, we write $\mathbf{Q}_1 \leq \mathbf{Q}_2$ to mean that the conjunctive query $\mathbf{Q}_1$ is *contained* in $\mathbf{Q}_2$.

1.5 Contributions of the Thesis

In what follows, we list the main contributions of this thesis. For each item in the list, we provide pointer to the chapter or section in which the item is discussed.

1. We develop a generic framework for deduction with uncertainty, called the *parametric framework* (Chapter 2). We show that by "tuning" the parameters of our framework appropriately, any of the known IB frameworks can be simulated and new ones can be realized (Section 1.2).

2. We identify a collection of reasonable properties of combination functions assumed in uncertainty frameworks (Section 2.1).

3. We develop a declarative, fixpoint, and proof-theoretic semantics of (positive) programs in the parametric framework (Section 2.3) and establish their equivalence (Section 2.4).

4. As the heart of query optimization techniques, we study the problem of containment of conjunctive queries in the parametric framework and establish necessary and sufficient conditions for the containment for classes of queries (Chapter 3). The study is organized according to how the combination functions involved in the queries compare with the meet and join operators in the underlying certainty lattice. Our results yield tools for query optimization for large classes of conjunctive queries in the known IB frameworks.

5. Regarding the complexity of the bottom-up naive evaluation of programs in the parametric framework, we show that whenever the underlying lattice is finite, the evaluation can be done in PTIME, in the size of the input database (EDB) and the lattice (Section 4.4). When the certainty lattice is fixed, this evaluation can be done in PTIME in the EDB size only.

6. To study the termination and data complexity of programs in the parametric framework whose underlying certainty lattice is infinite, we classify the collection of disjunction functions allowed into three types — 1 to 3. A type 1 disjunction function coincides with the join operator in the lattice. A disjunction functions of type 2 and 3 return a "better" certainty on inputs different from the bottom and the top values, with the difference that a type 3 function will eventually "saturate" and return the top certainty value, while a type 2 function will never reach the top (unless the top is supplied as an argument value). Accordingly, we say a predicate symbol is of type i, if its associated disjunction function is of type i, for $1 \leq i \leq 3$. We show that in a program in our language, whenever every recursive predicate is of type 1, the evaluation of the program can be done in PTIME in the size of the EDB (Section 4.4).

7. In our study of the termination behavior of programs in the parametric framework in general, we argue that it is a hard problem to study. We illustrate that when the disjunction function associated with a recursive predicate in a program is of type 2 or 3, evaluating such a program on some EDBs may terminate in arbitrarily large number of iterations, independent of the EDB size, or it may not terminate at all. The behavior is dictated by the combination functions and the specificity of the certainties used in the program (Section 4.5). We develop a methodology for studying the termination property which intuitively amounts

to ignoring the specific rule certainties mentioned in the program, and instead study with symbolic certainty values. We perform detailed analysis for special such cases, the results of which are generalized as conjectures characterizing the termination property for arbitrary programs in our framework.

8. We study the expressive power of the AB and IB approaches to uncertainty (Chapter 5). We identify two useful operations which are not captured by our framework, namely that *selection by certainty* and *join by certainty*. We show that these operations can be neatly incorporated within our framework with the use of certainty constraints (CCs, for short). We also show that the incorporation of CCs in AB frameworks increases the expressive power of those frameworks. Since there are various AB frameworks, we consider their commonalities and introduce a "generic" AB framework, in which the annotation functions satisfy similar postulates imposed on the combination functions in the parametric framework. We show that the parametric framework extended with CCs has the same expressive power as the generic AB framework extended with CCs. This is an important result connecting the two approaches to uncertainty in terms of expressiveness.

9. We develop a semi-naive method for evaluating programs in the parametric framework which also takes into account the certainties as well as the combination functions specified in user programs (Section 6.1). The proposed method extends the corresponding method in the standard framework.

10. To show that the ideas in this thesis lend themselves to an easy-to-use and efficient environment for deduction with uncertain knowledge, we have implemented the semi-naive method we proposed and developed two implementations of the parametric framework — a top-down and a bottom-up (Section 6.2). Our top-down implementation is on top of the XSB system [SSW94], and the bottom-up implementation is on top of the CORAL database programming language [RSS92].

1.6 Thesis Outline

The rest of the chapters of this thesis are organized as follows. Chapter 2 presents the parametric framework developed in this thesis. This includes a collection of reasonable properties that should be satisfied by the combination functions allowed in the framework, followed by the presentation of the syntax and the semantics of the language. We will establish the equivalence of the declarative, fixpoint, and proof-theoretic semantics developed.

In Chapter 3, we study the problem of conjunctive query containment in the presence of uncertainty, which is a central issue in query optimization with uncertainty. Our various results in this chapter provide necessary and sufficient conditions for containment of parametric conjunctive queries.

The complexity of the bottom-up naive evaluations of query programs in the parametric framework and termination behaviors of these programs are discussed in Chapter 4. We show that the study is hard, in general, by showing the difficulties faced when we attempt the issue. Instead of providing an "exact" characterization of termination, we motivate our approach which essentially amounts to studying a stronger termination condition. We identify a large collection of programs in the parametric framework which enjoy the PTIME data complexity, in the size of the EDB. Also, we identify classes of programs whose evaluations terminate in arbitrarily large but finite number of iterations. We conclude the chapter with a brief discussion on the so-called *finite precision assumption* and its affect on our special-case results obtained in this chapter.

In Chapter 5, we study the expressive power of the parametric framework. More specifically, we study the relationships between the AB and IB approaches to uncertainty. It was a desire to know if there is a theory which unifies the two. Fortunately, we find one such theory for which the notion of *certainty constraints (CCs)* is a key. We define a *generic* AB framework, which includes all the essential features of the AB frameworks. This framework is strictly more expressive than any existing AB framework. We show that the parametric framework extended with CCs is as expressive as this generic AB framework also extended with CCs. The proof technique to show this is based on transformation of programs in one formalism into the other and simulating its computation.

Chapter 6 is devoted to implementation details of the parametric framework. In Section 6.1, we study efficient evaluation of query programs in our framework and propose a semi-naive method by extending the idea of the corresponding method in the standard framework. The proposed method is used in our top-down implementation of the parametric framework as well as the bottom-up implementation. The various modules which constitute the former implementation are introduced in Section 6.2, and the inference engine of the letter is presented in 6.2.5. Our top-down system runs on top of the XSB system and the bottom-up runs on top of the CORAL database programming language. We will demonstrate through some example how programs would be evaluated using these implementations.

In addition to concluding remarks provided in various chapters, the last chapter includes some general remarks and future research directions.

Chapter 2

The Parametric Framework

In this chapter, we present the parametric framework [LS97, LS96c, LS96b]. We first characterize the families of propagation, conjunction, and disjunction functions allowed in this framework, and then introduce the syntax of this framework. We develop the declarative, fixpoint, and proof-theoretic semantics of programs in this framework and establish their equivalence. The semantics of programs in the parametric framework uses *multiset* as the basis of the structure of the semantics, as opposed to using *sets* which is often done. Finally, in Section 2.5, we will compare our work with previous work on similar unifying frameworks developed in different contexts, and close the chapter with a summary and concluding remarks.

Let $\mathcal{L}$ be an arbitrary, but fixed, first order language that contains infinitely many variable symbols, finitely many constants and predicate symbols, but no function symbols. While $\mathcal{L}$ does not contain function symbols, it contains symbols for the families of propagation ($\mathcal{F}_p$), conjunction ($\mathcal{F}_c$), and disjunction ($\mathcal{F}_d$) functions. We let $\mathcal{F} = \mathcal{F}_c \cup \mathcal{F}_p \cup \mathcal{F}_d$, and refer to each element in $\mathcal{F}$ as a *combination function*. We let $\mathcal{T}$ denote the underlying certainty domain in a framework considered. As is customary, we assume that $\mathcal{T}$ is a complete lattice with the partial order $\preceq$, and with $\bot$ and $\top$ as the bottom and top elements, respectively. The properties of these functions are defined next.

2.1 Combination Functions

Let $\mathcal{T}$ be a certainty lattice and $\mathcal{B}(\mathcal{T})$ be the set of finite multisets over $\mathcal{T}$. Then, a propagation function is a mapping from $\mathcal{T}^2$ to $\mathcal{T}$, and a conjunction or disjunction function is a mapping from $\mathcal{B}(\mathcal{T})$ to $\mathcal{T}$. For practical reasons, we assume that every combination function in $\mathcal{F}$ can be computed "efficiently". Theoretically, we also assume that every such function can be computed with arbitrary precision.

In order that derivations in our parametric framework are meaningful, we impose some natural properties on the combination functions, as enumerated below. For simplicity, we formulate these properties, treating every combination function f as a binary function on $\mathcal{T}$. The properties have their obvious formulation when f is a function from $\mathcal{B}(\mathcal{T})$ to $\mathcal{T}$. Since, as we will specify shortly, our conjunction and disjunction functions are required to be associative and commutative, the binary formulation is quite meaningful in all cases. Properties 7 to 9 are peculiar to unary functions of the form $f : \mathcal{B}(\mathcal{T}) \rightarrow \mathcal{T}$. After the following list of properties, we will identify, for each family of functions I_d, I_p and I_c, a subset of these properties that should be satisfied by every member in the family considered.

1. *Monotonicity*: $f(\alpha_1, \alpha_2) \preceq f(\beta_1, \beta_2)$, whenever $\alpha_i \preceq \beta_i$, for $i = 1, 2$.
2. *Continuity*: f is continuous w.r.t. each one of its arguments (w.r.t. Scott's topology).
3. *Bounded-Above*: $f(\alpha_1, \alpha_2) \preceq \alpha_i$, for $i = 1, 2$. That is, the result of f cannot be "more" than any one of its arguments.
4. *Bounded-Below*: $f(\alpha_1, \alpha_2) \succeq \alpha_i$, for $i = 1, 2$. That is, the result of f cannot be "less" than any one of its arguments.
5. *Commutativity*: $f(\alpha, \beta) = f(\beta, \alpha)$, $\forall \alpha, \beta \in \mathcal{T}$.
6. *Associativity*: $f(\alpha, f(\beta, \gamma)) = f(f(\alpha, \beta), \gamma)$, $\forall \alpha, \beta, \gamma \in \mathcal{T}$.
7. $f(\{\!|\alpha|\!\}) = \alpha$, $\forall \alpha \in \mathcal{T}$.
8. $f(\dot{\emptyset}) = \perp$, where $\perp$ is the least element in $\mathcal{T}$.

9. $f(\dot{\emptyset}) = \top$, where $\top$ is the greatest element in $\mathcal{T}$.

10. $f(\alpha, \top) = \alpha, \forall \alpha \in \mathcal{T}$.

11. $f(\alpha, \beta) \succ \bot, \forall \alpha, \beta \succ \bot$.

Postulate 2.1.1 *We require that the combination functions in the parametric framework should satisfy certain properties, postulated as follows.*

- *Every conjunction function in $\mathcal{F}_c$ should satisfy properties 1, 2, 3, 5, 6, 7, 9, 10, and 11;*
- *Every propagation function in $\mathcal{F}_p$ should satisfy properties 1, 2, 3, 10, and 11;*
- *Every disjunction function in $\mathcal{F}_d$ should satisfy properties 1, 2, 4, 5, 6, 7, and 8.*

We naturally require that every combination function in $\mathcal{F}$ be computed efficiently. Similar assumptions were made by Kifer and Li [KL88] in a different context. A detailed comparison with their work appears in Section 2.5.

Remarks. The continuity of combination functions is needed for proving continuity of the immediate consequence operator (Lemma 2.3.6). This requirement is not crucial for our results on containment of parametric conjunctive queries in Chapter 3. The commutativity and associativity of the conjunction functions are required for allowing query optimization, e.g. for performing subgoal reordering, if desired. The commutativity and associativity of disjunction functions are needed so that in evaluating a program, the certainty obtained for each ground atom is unique and is independent of the order in which the certainties of the subgoals in a rule are combined, and independent of the order in which the various certainties obtained for an atom are combined. Boundedness requirements are imposed in order that derivations make intuitive sense. Postulate 9 for a conjunction function together with Postulate 10 for a propagation function allow derivation of a ground atom A with certainty α from a fact of the form $A \xleftarrow{\alpha}$ (the rule syntax is presented below). Postulate 8 has a similar rationale for disjunction functions. Postulate 11, which is called *strictness* in [Fag96], is introduced to disallow useless rules which derive atoms with the least certainty $\bot$. Although without this postulate, such rules could be safely removed from a program without changing the semantics, it is crucial for some of our results in Chapter 4 where we discuss the termination property of programs with uncertainty.

2.2 Syntax

In this section, we introduce the syntax of programs in the parametric framework.

Definition 2.2.1 *A* parametric program *(p-program) P is a 5-tuple $\langle \mathcal{T}, \mathcal{R}, \mathcal{C}, \mathcal{P}, \mathcal{D} \rangle$, whose components are defined as follows.*

- *$\mathcal{T}$ is a set of truth values partially ordered by $\preceq$. We assume $<\mathcal{T}, \preceq, \otimes, \oplus>$ is a complete lattice, where $\otimes$ is the meet operator and $\oplus$ is the join. We denote the least element of the lattice by $\perp$ and the greatest element by $\top$, which correspond, respectively, to the truth values* false *and* true *in standard logic.*

- *$\mathcal{R}$ is a finite set of* parametric rules *(p-rules), each of which is a statement of the form:*

$$r : A \xleftarrow{\alpha_r} B_1, \ldots, B_n$$

  where $A, B_1, \ldots, B_n$ are atoms, and $\alpha_r \in \mathcal{T} - \{\perp\}$ is the certainty of r.

- *$\mathcal{C}$ is a mapping which associates with each p-rule in P a conjunction function in $\mathcal{F}_c$.*

- *$\mathcal{P}$ is a mapping which associates with each p-rule in P a propagation function in $\mathcal{F}_p$.*

- *$\mathcal{D}$ is a mapping which associates with each predicate symbol in P a disjunction function in $\mathcal{F}_d$.*

We will use the terms p-program and program interchangeably. The same remark holds for p-rule and rule. Also for convenience, we represent a p-rule r as

$$r : A \xleftarrow{\alpha_r} B_1, \ldots, B_n;\ \langle f^d, f^p, f^c \rangle$$

where $f^d \in \mathcal{F}_d$ is the disjunction function associated with $\pi(A)$, and $f^p \in \mathcal{F}_p$ and $f^c \in \mathcal{F}_c$ are the propagation and conjunction functions associated with r. Atom A in the p-rule r above is called the rule *head*, and the conjunction $B_1, \ldots, B_n$ is called the rule *body*. If this conjunction is empty, then r is called a fact. That is, a *fact* is a special case of a p-rule in which $n = 0$. Since a fact does not need a conjunction function, the triple associated with facts would be of the form $\langle f^d, f^p, \_ \rangle$.

For consistency reason, we require that *all the p-rules in a p-program defining the same head predicate should have the same associated disjunction function.*

2.3 Semantical Foundation

In this section we study and develop the semantics of programs in the parametric framework. Without loss of generality, we restrict our attention to Herbrand structures. We first develop the *declarative* semantics based on the notion of *valuations*, and show that every p-program has a least valuation, w.r.t. the ordering $\preceq$ on valuations obtained by extending the ordering $\preceq$ on $\mathcal{T}$. We will then develop a fixpoint theory for p-programs. Finally, we present the proof procedure for programs in the parametric framework and show that it is sound and complete. In Section 2.4, we will establish the equivalence of these semantics.

Let P be a p-program, and B_P the Herbrand base of P. A *valuation* v of P is a mapping from B_P to $\mathcal{T}$, i.e., v associates with each ground atom in B_P, a truth value in $\mathcal{T}$. A *ground instance* of a p-rule r in P is a ground p-rule obtained from r by replacing all occurrences of each variable in r with an element of the Herbrand domain. Since p-programs are function free, this domain is finite as it contains just the constant symbols mentioned in P. The *Herbrand instantiation* of P, denoted P^*, is the set of all ground instances of every p-rule in P. For an atom A, we denote by $\pi(A)$ the predicate symbol of A, also called the functor of A.

2.3.1 Declarative Semantics

To develop the declarative semantics, we first define the notion of satisfaction of p-programs by valuations, which follows.

Definition 2.3.1 (Satisfaction) *Let P be any p-program, r be any p-rule in P, and v be any valuation of P. Let $\rho \equiv (A \xleftarrow{\alpha_r} B_1, \ldots, B_n; \langle f^d, f^p, f^c \rangle) \in P^*$ be any ground instance of r. Then, we say that*

(a) v satisfies ρ, denoted $\models_v \rho$, iff $v(A) \succeq f^p(\alpha_r, f^c(\{\!|v(B_1), \ldots, v(B_n)|\!\}))$.

(b) $\models_v r$ iff v satisfies every ground instance of r.

(c) v satisfies P, $\models_v P$, iff (1) v satisfies every p-rule in P, and (2) $\forall A \in B_P$, $v(A) \succeq f^d(X)$, where f^d is the disjunction function associated with the predicate symbol of A, and

$$X = \{\!\{f^p(\alpha_r, f^c(\{\!\{v(B_1), \ldots, v(B_n)\}\!\})) \mid (A \xleftarrow{\alpha_r} B_1, \ldots, B_n; \langle f^d, f^p, f^c\rangle) \in P^*\}\!\}.$$

Note that f^d is the disjunction function associated with the predicate symbol of atom A, and X is the multiset of truth values, each of which is obtained by applying a ground rule in P^* with head A. Also note that it follows from Postulates 9 and 10 that whenever $n = 0$, $f^p(\alpha_r, f^c(\dot{\emptyset})) = \alpha_r$. As noted in [KL88] and [LS94a], although v may satisfy every p-rule in P, in general, it may fail to satisfy P. For v to also satisfy P, condition c(2) ensures that for each atom $A \in B_P$, the certainty assigned to A by v is *not* less than $f^d(X)$, w.r.t. the lattice ordering $\preceq$.

The ordering $\preceq$ on $\mathcal{T}$ can be extended to valuations in the well-known manner; for any valuations u and v of a p-program P, $v \preceq u$ iff $v(A) \preceq u(A)$, $\forall A \in B_P$. For all valuations u, v of P and for every ground atom $A \in B_P$, $(u \otimes v)(A) = u(A) \otimes v(A)$ and $(u \oplus v)(A) = u(A) \oplus v(A)$. We then have the following.

Lemma 2.3.2 *Let P be any p-program, and Υ_P be the set of valuations of P. Then $\langle \Upsilon_P, \otimes, \oplus\rangle$ is a complete lattice.*

Proof. The result follows upon noting that $\langle \Upsilon_P, \dot{\otimes}, \dot{\oplus}\rangle$ is obtained by a pointwise extension of the corresponding order/operation on the complete lattice $<\mathcal{T}, \preceq, \otimes, \oplus>$. ■

The least element of $\langle \Upsilon_P, \otimes, \oplus\rangle$ is a valuation, $v_\perp$, which maps every ground atom $A \in B_P$ to $\perp$, and the greatest element is a valuation, $v_\top$, which maps A to $\top$.

The following lemma is the counterpart of the model intersection property in standard logic programming and deductive databases.

Lemma 2.3.3 *Let P be a p-program and u, v be any valuations of P each of which satisfies P. Then $u \otimes v$ is also a valuation which satisfies P.*

Proof. Let A be any ground atom in B_P, and $\rho \equiv (A \xleftarrow{\alpha_r} B_1, \ldots, B_n;\ \langle f^d, f^p, f^c\rangle)$ be any ground p-rule in P^* defining A. Let $w = u \otimes v$. Since u satisfies P, by Definition

2.3.1, we have that $u(A) \succeq f^p(\alpha_r, f^c(\{\!|u(B_1), \ldots, u(B_n)|\!\}))$. Since, by our postulates, f^c and f^p are monotone functions, we further have that

$$u(A) \succeq f^p(\alpha_r, f^c(\{\!|u(B_1), \ldots, u(B_n)|\!\})) \succeq f^p(\alpha_r, f^c(\{\!|w(B_1), \ldots, w(B_n)|\!\})).$$

In a similar way we can show that $v(A) \succeq f^p(\alpha_r, f^c(\{\!|w(B_1), \ldots, w(B_n)|\!\}))$, which together with the previous one imply that $u(A) \otimes v(A) \succeq f^p(\alpha_r, f^c(\{\!|w(B_1), \ldots, w(B_n)|\!\}))$. This in turn implies that w satisfies the instance ρ of r, as $w = u \otimes v$. Moreover, it follows from Definition 2.3.1 that $u(A) \succeq f^d(X)$ and $v(A) \succeq f^d(X)$, where $X = \{\!|f^p(\alpha_r, f^c(\{\!|\beta_1, \ldots, \beta_k|\!\})) \mid (A \xleftarrow{\alpha_r} B_1, \ldots, B_n; \langle f^d, f^p, f^c \rangle) \in P^*|\!\}$ and $f^d = \mathcal{D}(\pi(A))$. We thus have $u(A) \otimes v(A) \succeq f^d(X)$, from which we may conclude that $w(A) \succeq f^d(X)$, since $w = u \otimes v$. It then follows from Definition 2.3.1 that w satisfies every p-rule in P which defines A. Since A was an arbitrary atom, we may conclude that $\models_w P$. ■

The notion of least valuations, introduced next, corresponds to the notion of least models in standard logic programming and deductive databases.

Theorem 2.3.4 *Let P be a p-program, and Υ_P the set of valuations of P. Then, $\otimes\{v \mid v \in \Upsilon_P$ and $\models_v P\}$ is the least valuation that satisfies P.*

Proof. Note that $v_\top : B_P \rightarrow \{\top\}$ is a valuation in Υ_P which satisfies P, and hence the above set is not empty. The result then follows from Lemma 2.3.3 and the definition of the least valuation. ■

2.3.2 Fixpoint Theory

We now present the fixpoint semantics of p-programs. As in standard deductive databases, we associate with a p-program P, an *immediate consequence operator*, T_P. We then show that the least fixpoint of T_P exists and define it as the fixpoint semantics of P. Later on in Section 2.4, we will establish the connection between the fixpoint and declarative semantics.

Definition 2.3.5 *Let P be any p-program, and P^* be the Herbrand instantiation of P. Also let Υ_P be the set of valuations of P. The immediate consequence operator T_P is a mapping from Υ_P to Υ_P, such that for every valuation $v \in \Upsilon_P$ and every ground*

atom $A \in B_P$, $T_P(v)(A) = f^d(X)$, *where* f^d *is the disjunction function associated with* $\pi(A)$, *the predicate symbol of* A, *and*

$$X = \{\!\{f^p(\alpha_r, f^c(\{\!\{v(B_1), \ldots, v(B_n)\}\!\})) \mid (A \xleftarrow{\alpha_r} B_1, \ldots, B_n;\ \langle f^d, f^p, f^c\rangle) \in P^*\}\!\}.$$

We define the bottom-up iteration of T_P, in the usual manner, as follows.

$$T_P^k = \begin{cases} v_\perp & \text{if } k = 0 \\ T_P(T_P^{k-1}) & \text{if } k \text{ is a successor ordinal} \\ \oplus\{T_P^\ell \mid \ell < k\} & \text{if } k \text{ is a limit ordinal} \end{cases}$$

Note that if $n = 0$, then it follows from Postulates 9 and 10 that $f^p(\alpha_r, f^c(\dot{\emptyset})) = \alpha_r$, where $\dot{\emptyset}$ is the empty multiset. Also note that, for any ground atom $A \in B_P$, if there exists no p-rule in P whose head can be unified with A, then it follows from the definition of T_P that $T_P(v)(A) = \perp$, for every valuation v of P.

Lemma 2.3.6 *The operator* T_P *is monotone and continuous.*

Proof. First, we show that T_P is monotone. Let u and v be any valuations of P such that $u \preceq v$. We will show that $T_P(u) \preceq T_P(v)$. Let A be any ground atom in B_P, and suppose $f^d = \mathcal{D}(\pi(A))$, i.e., f^d is the disjunction function associated with the predicate symbol of A. Then, by Definition 2.3.5, $T_P(u)(A) = f^d(X_u)$, where $X_u = \{\!\{f^p(\alpha_r, f^c(\{\!\{u(B_1), \ldots, u(B_n)\}\!\})) \mid (A \xleftarrow{\alpha_r} B_1, \ldots, B_n;\ \langle f^d, f^p, f^c\rangle) \in P^*\}\!\}$. Since $u \preceq v$, we have that $u(A) \preceq v(A)$, $\forall A \in B_P$. In particular, $u(B_k) \preceq v(B_k)$, $1 \leq k \leq n$. Since, by our postulates, the conjunction and propagation functions f^c and f^p are monotone, we may conclude that $f^d(X_u) \preceq f^d(X_v)$, where X_v is defined as X_u with u replaced by v. Thus, $T_P(u)(A) \preceq T_P(v)(A)$.

Next, we show that T_P is continuous. Let $v_0 \preceq v_1 \preceq \ldots$ be any chain of valuations of P, and A be any ground atom in B_P. To show that T_P is continuous, we show that

$$T_P(\oplus\{v_j(A) \mid j \geq 0\}) = \oplus\{T_P(v_j)(A) \mid j \geq 0\} \qquad (1)$$

First let us determine the left hand side of equation 1. By Lemma 2.3.2, since the set Υ_P of valuations of P, ordered by $\preceq$, is a complete lattice, the chain $v_0 \preceq v_1 \preceq \ldots$ has a least upper bound, say v. That is, $v = \oplus\{v_j \mid j \geq 0\}$. Now, consider all ground rules in P^* whose head is A. Since our language is function free, P^* is

finite, and hence there is a finite number of such rules, say $\rho_1, \ldots, \rho_k$, where $\rho_i \equiv (A \xleftarrow{\alpha_i} B_1^i, \ldots, B_{n_i}^i;\ \langle f_i^d, f_i^p, f_i^c \rangle)$. Note that $f_1^d = \ldots = f_k^d$. Let $f^d = f_i^d$, $1 \leq i \leq k$. Then, by Definition 2.3.5, we have

$T_P(v)(A) = f^d(\{\!|f_i^p(\alpha_i, f_i^c(\{\!|v(B_1^i), \ldots, v(B_{n_i}^i)|\!\})) \mid \rho_i \in P^*, 1 \leq i \leq k|\!\})$.

Next, we will determine the right hand side of equation 1 and show that it is identical to the last expression above. By Definition 2.3.5,

$\oplus\{T_P(v_j)(A) \mid j \geq 0\} =$

$\oplus\{f^d(\{\!|f_i^p(\alpha_i, f_i^c(\{\!|v_j(B_1^i), \ldots, v_j(B_{n_i}^i)|\!\})) \mid \rho_i \in P^*, 1 \leq i \leq k|\!\}) \mid j \geq 0\}$.

Since, by our postulates, f^d, f_i^p, and f_i^c are continuous, we may push the operator $\oplus$ inside the expression and maintain equality.[1] Doing so, we obtain the following sequence of equations.

$\oplus\{T_P(v_j)(A) \mid j \geq 0\} =$

$\oplus\{f^d(\{\!|f_i^p(\alpha_i, f_i^c(\{\!|v_j(B_1^i), \ldots, v_j(B_{n_i}^i)|\!\})) \mid \rho_i \in P^*, 1 \leq i \leq k|\!\}) \mid j \geq 0\} =$

$f^d(\{\!|\oplus\{f_i^p(\alpha_i, f_i^c(\{\!|v_j(B_1^i), \ldots, v_j(B_{n_i}^i)|\!\})) \mid j \geq 0\} \mid \rho_i \in P^*, 1 \leq i \leq k|\!\}) =$

$f^d(\{\!|f_i^p(\alpha_i, \oplus\{f_i^c(\{\!|v_j(B_1^i), \ldots, v_j(B_{n_i}^i)|\!\}) \mid j \geq 0\}) \mid \rho_i \in P^*, 1 \leq i \leq k|\!\}) =$

$f^d(\{\!|f_i^p(\alpha_i, f_i^c(\{\!|\oplus\{v_j(B_1^i) \mid j \geq 0\}, \ldots, \oplus\{v_j(B_{n_i}^i) \mid j \geq 0\}|\!\})) \mid \rho_i \in P^*, 1 \leq i \leq k|\!\})$.

Since v is the least upper bound of the chain, the last expression above is equal to $f^d(\{\!|f_i^p(\alpha_i, f_i^c(\{\!|v(B_1^i), \ldots, v(B_{n_i}^i)|\!\})) \mid \rho_i \in P^*, 1 \leq i \leq k|\!\})$, which was to be shown. ■

A valuation satisfies a p-program P if and only if it is a prefixpoint of T_P. This is stated as the following lemma, the proof of which follows from Definitions 2.3.5 and 2.3.1.

Lemma 2.3.7 *Let P be any p-program, and v be any valuation of P. Then, $\models_v P$ iff $T_P(v) \preceq v$.*

The *least fixpoint* of T_P, denoted $lfp(T_P)$, is a valuation v of P such that v is a fixpoint of T_P and, for every fixpoint u of T_P, $v \preceq u$.

In Section 2.4, we will establish the connection between the declarative and fixpoint semantics of p-programs.

[1] That is, for every combination function $f \in \mathcal{F}_d \cup \mathcal{F}_p \cup \mathcal{F}_c$,

$$\oplus\{f(\{\!|\sigma_1^j, \ldots, \sigma_n^j|\!\}) \mid j \geq 0\} = f(\{\!|\oplus\{\sigma_1^j \mid j \geq 0\}, \ldots, \oplus\{\sigma_n^j \mid j \geq 0\}|\!\}).$$

2.3.3 Proof Theory

Unlike in standard logic programming and deductive databases, a proof procedure in the context of programming with uncertainty *cannot* in general ignore alternate derivations of the same atom. This is because the certainties associated with different derivations of each atom should be "combined", reinforcing its overall certainty. We develop a sound and complete proof theory for the parametric framework based on the notion of disjunctive derivation trees (DDTs), adapted from [LS94a]. Intuitively, a DDT for a ground atom A w.r.t. a p-program P (including the EDB) is a finite collection of derivation trees each of which is a finite and/or tree encoding a proof of A from P. A formal definition of DDTs follows.

Definition 2.3.8 Let P be any p-program and $A \in B_P$ be any goal. A *disjunctive derivation tree* T_A (DDT, for short) for A w.r.t. P is a collection of finite derivation trees defined as follows.

1. Each node in T_A is either a *rule node* or a *goal node*. Each rule node is labeled by a ground instance of a p-rule in P. Each goal node is labeled by a ground atom in B_P. The root of T_A is a goal node labeled A.

2. If G is a goal node in T_A, then every child (if any) of G is a rule node labeled with some ground instance, say ρ, of a p-rule r in P, where the head of ρ is G. In this case, we associate with node ρ the propagation and conjunction functions associated with r, and associate with G the disjunction function associated with $\pi(G)$, the predicate symbol of G.

3. If $B \in B_P$ is an atom in the body of a rule node labeled u, then u has as a child a goal node labeled B. ■

Note that every label in a DDT, whether that of a goal node or a rule node, is ground. A *leaf* in a DDT is a node without children. A DDT T is called *proper* if whenever T has a goal leaf labeled A, there is no ground p-rule in P^* whose head is A. This essentially means that all the proof opportunities are exploited in T. A rule leaf is a *success* node while a goal leaf is a *failure* node. In the sequel, we only consider proper DDTs.

Definition 2.3.9 Let P be any p-program and T be any DDT w.r.t. P. Then, the following procedure defines the certainties assigned to the nodes in T.

1. Each failure node in T is assigned $\perp$, the least certainty in $\mathcal{T}$.
2. Each success node labeled with a ground instance ρ of a p-rule $r \in P$ is assigned α_r, the certainty associated with the implication in r.
3. If u is an internal node in T, i.e., a non-leaf node, then the certainty assigned to u is determined by one of the following two cases.
   (a) If u is a rule node labeled with $\rho \equiv (r : A \xleftarrow{\alpha_r} B_1, \ldots, B_n;\ \langle f^d, f^p, f^c \rangle)$, where ρ is a ground instance of $r \in P$, then $f^p(\alpha_r, f^c(\{\!|\beta_1, \ldots, \beta_n|\!\}))$ is the certainty assigned to u, where $\beta_i \in \mathcal{T}$ is the certainty assigned to node B_i, for $1 \leq i \leq n$.
   (b) If u is a goal node labeled with a ground atom A, then u has at least one subtree, since u is an internal node, by our assumption. Suppose $T_1, \ldots, T_m$ are all the subtrees at node u. Also suppose $\sigma_j \in \mathcal{T}$ is the certainty assigned to the root of T_j, for $1 \leq j \leq m$. Then, the certainty assigned to u is $f^d(\{\!|\sigma_1, \ldots, \sigma_m|\!\})$, where f^d is the disjunction function associated with $\pi(A)$, the predicate symbol of A. ■

Note that in a DDT, the rule nodes are at odd levels and the goal nodes at even levels, with the root being at level 0. It then follows from the definition of proper DDT that every such tree has even number of levels and the leaves are at the odd levels. We define the *height* of a DDT T as the number of even levels in T.

2.4 Semantics Equivalence

In this section, we will show the equivalence of the declarative, fixpoint, and proof-theoretic semantics developed for p-programs. This is done at two steps. First, we show that the declarative and fixpoint semantics coincide, and then show that the proof-theoretic semantics corresponds to the fixpoint semantics.

The following theorem, analogous to the van Emden-Kowalski theorem in standard logic programming [vK76], establishes the connection between the fixpoint and the declarative semantics of p-programs.

Theorem 2.4.1 *Let P be any p-program. Then $lfp(T_P) = \otimes\{v \mid \models_v P\}$.*

Proof. Follows from Lemma 2.3.6 in the standard way [vK76]. ■

We next show that the proof-theoretic semantics developed for the parametric framework is sound and complete, and establish its equivalence to the fixpoint semantics.

Theorem 2.4.2 (Soundness) *Let P be any p-program, and $A \in B_P$ be any goal. If T_A is a DDT for A w.r.t. P such that $\alpha \in \mathcal{T}$ is the truth value associated with the root of T_A, then $\alpha \preceq lfp(T_P)(A)$.*

Proof. Suppose k is the height of T_A. We prove the soundness of the proof procedure by induction on k.

Basis case: $k = 1$. Since T_A is of height 1, all children of the goal root A must be (rule) leaves. Let these leaves be labeled as $A \xleftarrow{\alpha_1}, \ldots, A \xleftarrow{\alpha_n}$. The certainty of the root is thus $\alpha = f^d(\{\!|\alpha_1, \ldots, \alpha_n|\!\})$. Clearly, $lfp(T_P)(A) = f^d(X)$, where $\{\!|\alpha_1, \ldots, \alpha_n|\!\} \dot{\subseteq} X$. From this we may conclude that $\alpha \preceq lfp(T_P)(A)$, upon noting that f^d and T_P are monotone.

Inductive hypothesis: Suppose for every DDT T_A of height k, the truth value α associated with the root of T_A is such that $\alpha \preceq lfp(T_P)(A)$.

Inductive step: Suppose T_A is a DDT of height $k+1$, for some atom A. Let $G_1, \ldots, G_m$ be all the goal nodes in T_A at level 2, and T_{G_i} be the subtree of T_A at node G_i, for $1 \leq i \leq m$. Note that each of these T_{G_i}'s is a DDT whose height is at most k. Suppose σ_i is the truth value associated with the root of T_{G_i}. Then, by the inductive hypothesis, $\sigma_i \preceq lfp(T_P)(G_i)$, for $1 \leq i \leq m$. Suppose f^d is the disjunction function associated with $\pi(A)$. It then follows from the definition of DDTs that the certainty assigned to the root of T_A is $\alpha = f^d(\{\!|\sigma_1, \ldots, \sigma_m|\!\})$. In this case, since $lfp(T_P)$ satisfies P, we have that $f^d(\{\!|\sigma_1, \ldots, \sigma_m, \ldots, \sigma_\ell|\!\}) \preceq lfp(T_P)(A)$, from which the result follows. ■

Theorem 2.4.3 (Completeness) *Let P be any p-program, and $A \in B_P$ be any goal such that $lfp(T_P)(A) = T_P^k(A)$, for some positive integer k. Then there exists a DDT, T_A, for A w.r.t. P such that $\alpha \in \mathcal{T}$ is the certainty associated with the root of T_A and $lfp(T_P)(A) \preceq \alpha$.*

Proof. The proof proceeds by induction on the iteration step k. Without loss of generality, assume $lfp(T_P)(A) \neq \bot$; otherwise, the result trivially follows.

Basis case: For $k = 1$, $lfp(T_P)(A) = T_P(A)$. In this case, P^* contains (at least) one ground p-rule of the form $A \xleftarrow{\alpha_i}$, for some $\alpha_i \in \mathcal{T}$, and every p-rule in P^* whose head is A must be a unit clause. Suppose $\rho_1, \ldots, \rho_j$ are all such p-rules in P^*, where $j \geq 1$. Then, the desired DDT T_A can be constructed as follows. The root of T_A is the goal node labeled A which has j children, where the i-th child is the rule node labeled ρ_i, $1 \leq i \leq j$. Then, by Definition 2.3.9, the certainty value assigned to the root of T_A is $\alpha = f^d(\{\!|\alpha_1, \ldots, \alpha_j|\!\})$, where f^d is the disjunction function associated with the predicate symbol of A, the root of T_A. Note that in this case, $lfp(T_P)(A) = \alpha$, by Definition 2.3.5 of T_P.

Inductive hypothesis: Suppose for some $k \geq 1$, whenever $lfp(T_P)(A) = T_P^k(A)$, there exists a DDT T_A for atom A such that the truth value α associated with the root of T_A is such that $lfp(T_P)(A) \preceq \alpha$.

Inductive case: Suppose $lfp(T_P)(A) = T_P^{k+1}(A)$. Then, P^* contains at least one rule instance with head A. Let $\rho_1, \ldots, \rho_m$ be all instances in P^* with head A. Let B be a ground atom appearing in the body of any one of these rule instances, say ρ_j. Since $lfp(T_P)(A) = T_P^{k+1}(A)$, we know that $lfp(T_P)(B) = T_P^k$. By inductive hypothesis, for each such atom B, there exists a DDT T_B such that the certainty associated with the root of T_B is α_B, where $lfp(T_P)(B) \preceq \alpha_B$. The desired DDT T_A can then be constructed as follows. The root of T_A is the goal node labeled A which has m children, the j-th of which is the rule node labeled ρ_j, $1 \leq j \leq m$. If $\rho_j \equiv (r : A \xleftarrow{\alpha_r} D_1, \ldots, D_i;\ \langle f^d, f^{p_j}, f^{c_j} \rangle)$, then the rule node ρ_j has the DDTs $T_{D_1}, \ldots, T_{D_i}$ as its children. This completes T_A's construction. Now we know that $lfp(T_P)(A) \preceq f^d(\{\!|f^{p_1}(\alpha_1, f^{c_1}(\{\!|\alpha_{B_1}, \ldots, \alpha_{B_n}|\!\})), \ldots, f^{p_m}(\alpha_m, f^{c_m}(\{\!|\alpha_{C_1}, \ldots, \alpha_{C_\ell}|\!\}))|\!\})$, since, by our postulates, the combination functions f^c, f^p, and f^d are monotone. The result then follows upon noting that the right hand side of the above inequality is the certainty associated with the root of T_A. ∎

Given any p-program P and any goal A, a DDT for A can be completely constructed if P contains no recursive predicates or the EDB contains no cycles. If P is recursive, this construction is possible, provided for every disjunction function f^d associated with some predicate in P and for every truth values $\alpha, \beta \in \mathcal{T}$, $f^d(\{\!|\alpha, \beta|\!\}) = \oplus(\{\!|\alpha, \beta|\!\})$. Intuitively, this happens whenever applying f^d to a multiset X returns the same value as f^d returns when applied to the multiset obtained by adding any element $\sigma \in \mathcal{T}$ to X, where $\sigma \preceq \oplus(X)$. For instance, the *pc* mode of disjunction defined in [LS94a], *max* in [van86], and the disjunction functions in [Fit88, Fit91, LS94b] satisfy this property.

2.5 Related Work

Uncertainty manifests itself in various different aspects, which can be classified in terms of its natures (semantics) and forms (syntax), or in terms of being qualitative, quantitative, or a hybrid of the two. The attempt has been to extend the standard relational database model for incorporating uncertainty. Quantitative modeling of uncertainty deals with incomplete information without associating any numerical value, which is known as null values introduced by Codd [Cod79]. A null value is a special symbol used as an attribute value in a tuple when the actual value is unknown or inapplicable. Barbara et al. [BGP92] proposed another approach to qualitative modeling of uncertainty at the attribute level, in which attribute values are replaced by sets of values. In their model, the actual attribute is uncertain, however, it is known that it must be one of the elements in the set. Grahne [Gra91] also studies the problem of incomplete information in relational databases. The basic idea in his work is that an incomplete relation represents a set of relations. In the quantitative approach to modeling uncertainty in databases, a numerical value is associated with each tuple in a relation, indicating the truth value of the tuple or the degree to which the tuple belongs to the relation. Probabilistic databases [BGP92, Joh94] and fuzzy databases [Zvi87, LL90] are examples of this approach. The hybrid approach combines the qualitative and quantitative approaches (See e.g. [Sad91]).

In all the above works, data is annotated with a certainty value of some kind. There is another aspect of uncertainty mainly considered in expert systems, in which in addition to data, the rules are associated with certainty values, for instance the

certainty factors in the MYCIN system [BS75] and Dempster-Shafer theory of evidence [GS90]. More recently, logic database programming, with its advantages of declarative and modularity, and its powerful top-down and bottom-up query processing techniques attracted the attention of researchers, and and numerous frameworks for deductive databases with uncertainty have been proposed. As should be clear from the discussion so far, the objective of this thesis is to develop a foundation for integrating uncertain databases with deduction in such a way that it unifies and/or generalizes the existing proposals for this integration.

In this section, we compare our work with previous work on similar unifying frameworks developed in different contexts, and (i) bring out the novelty and unique features of our work, and (ii) show why a new unifying framework was necessary for establishing the various results in this thesis. Notably, we consider Kifer and Li [KL88], Kifer and Subrahmanian [KS92], and Debray and Ramakrishnan [DR94], because, like this work, they generalize and/or unify some specific frameworks. We will also consider the probabilistic framework of Lakshmanan and Sadri [LS94a], because the parametric framework was inspired by it.

In the context of rule based systems with uncertainty, Kifer and Li [KL88] identified the main drawback of the earlier works on quantitative logics to be their lack of enough support for various conjunction and disjunction functions required in such systems: only *max* and *min* were supported as the disjunction and conjunction functions. They argue that, in practice, more general such functions are required, and propose a set of "reasonable" properties which should be satisfied by such functions. They also proposed an AB framework with generic combination functions, defined in terms of their properties. They essentially considered a "parametric" approach similar in spirit to ours. They developed a model and fixpoint semantics based on multisets. While their semantics does not rely on the various types of independence among the rules in a program, they assume supporting evidences for an atom should be absolutely independent of each other. The main problem with the proposed framework is that the T_P operator defined is not continuous. Also, it is unclear under what condition one can expect finite termination, although they give some sufficient conditions. The recurrence based evaluation they proposed, although elegant, is a departure from the conventional (semi)-naive evaluation method, and the possibility of efficient implementation of their method is not clear. In the theory of Generalized Annotated logic

Programming (GAP), discussed next, Kifer and Subrahmanian [KS92] solved several of these problems.

Kifer and Subrahmanian [KS92] proposed the GAP framework as a unifying framework which generalizes various results and treatments of temporal and multi-valued logic programming, e.g. [BS89, Fit91, KL89, Sha83, van86]. A main difference between GAP and any IB framework, including ours, is clearly in the approach. The two approaches have their own advantages and disadvantages, as discussed earlier in this chapter. In terms of expressive power, GAP can simulate the computation of some IB frameworks.[2] As the semantics developed for GAP is based on sets, it *cannot* simulate those IB frameworks which need multisets as the basis of their semantics. For instance, the probabilistic IB framework proposed in [LS94a], an instance which can be simulated within our parametric framework, cannot be simulated in GAP without changing its underlying semantics to be based on multisets. Although the GAP framework [KS92] allows an infinite family of disjunction functions $\sqcup_j$, $j \geq 0$, however, all these functions are induced by the lattice join $\sqcup$. This is quite restricted compared to the infinite family $\mathcal{F}_d$ of disjunction functions (which need not be lattice based) allowed in the parametric framework.

Debray and Ramakrishnan [DR94] proposed an axiomatic basis for Horn clause logic programming. Their framework nicely simulates a variety of "Horn-clause-like" computations, arising, for instance, in deductive databases, quantitative deduction, and inheritance systems, in terms of two operators, instance ($\hat{\otimes}$) and join ($\hat{\div}$). In the context of uncertainty reasoning, they show how their framework captures the computation in van Emden's language [van86]. In this context, it seems from their axioms that the proposed framework can also capture the fixpoint computation of IB frameworks which have sets as the basis for their semantical structures, e.g. [LS94b]. However, as explained below, the proposed setting is incompatible with probabilistic frameworks, such as [Lak94, LS94a, NS93, NS92].

First, in the formulation of the logic in [DR94], one of their axioms states that for any values t and t', the value $t \hat{\otimes} t'$ conveys no more information than does t.[3] In a probabilistic logic, this could be interpreted as saying in a weighted set of evidences,

[2] This is done through the annotation variables allowed in GAP.

[3] Axiom 4.1 in [DR94]. For any values $t, t' \in D$, (1) $\{t\} \hat{\oplus} \{t \hat{\otimes} t'\} \approx \{t\}$; and (2) $\{t\} \hat{\otimes} \{t \hat{\oplus} t'\} \approx \{t\}$. Note that $t \hat{\otimes} t'$ is an instance of t, denoted $t \hat{\otimes} t' \preceq t$.

adding an evidence, $t \widehat{\otimes} t'$, which is subsumed by another element in the set, t, does not increase the information "content" of the set. This may be inappropriate in a probabilistic setting. For, suppose, e is an event with probability 0.5. Then, according to this axiom, if we add a lower probability value, 0.4, to a set of such values, $\{0.5\}$, the content of the result set, $\{0.4, 0.5\}$, does not increase, which may not always be true, e.g. when the combination mode is "independence".

The second limitation of the framework presented in [DR94] is the distributivity requirement of their lattice. This is not suitable, in general, for probabilistic inferencing. In fact, the appropriate probabilistic conjunction and disjunction used in an application are driven by the underlying assumptions about the nature of event interaction, and they need not distribute over one another, in general. For instance, if α_1, α_2 and α_3 are confidences in the probabilistic framework of Lakshmanan and Sadri [LS94a], with the disjunction mode $\vee_{ign}$ (i.e. ignorance) and the conjunction mode $\wedge_{pc}$ (i.e. positive correlation), then $(\alpha_1 \vee_{ign} \alpha_2) \wedge_{pc} \alpha_3 \neq (\alpha_1 \wedge_{pc} \alpha_3) \vee_{ign} (\alpha_2 \wedge_{pc} \alpha_3)$.

We now consider the probabilistic deductive databases of Lakshmanan and Sadri [LS94a], which inspired the development of the parametric framework. In their framework, they considered "parameterized" conjunctions and disjunctions in the context of probabilistic deduction and developed a probabilistic calculus. Our framework differs from theirs in several ways. First, the uncertainty domain in our framework itself is parameterized. Also, we allow a family of propagation functions as yet another parameter, which allows to use in the same rule, a propagation function which is different from the conjunction function used. We illustrated in Example 1.2.7 that their framework is a special case of ours by an appropriate choice of the parameters. Finally, unlike all the above works, we address query optimization and establish necessary and sufficient conditions for containment of conjunctive queries, and also address the termination and complexity of the fixpoint evaluations of p-programs. We remark that Ioannidis and Ramakrishnan [IR95] considered the problem of query containment in a broad setting where relations may be interpreted as sets or as multisets. A comparison with their work appears in Chapter 3.

2.6 Summary and Concluding Remarks

In this chapter, we introduced the parametric framework, a powerful language for deductive databases with uncertainty, which generalizes and unifies the IB approach to uncertainty, in the sense the any existing IB framework becomes an instance of this framework, and new ones can be realized. We identified a collection of reasonable properties which should be satisfied by every family of disjunction, propagation, and conjunction functions employed in deduction with uncertainty. We introduced the syntax and developed the semantics of the framework. The proposed semantics uses multisets as the underlying semantical structures, as opposed to sets, conventionally used. We developed the declarative, fixpoint, and proof-theoretic semantics of programs in the parametric framework and established their equivalence by showing that (1) the declarative semantics of every program in the parametric framework coincides with the bottom-up fixpoint semantics of the program and (2) the fixpoint semantics coincides with the proof-theoretic semantics. We showed that the fixpoint operator T_P is monotone and continuous, and that the proof theory developed is sound and complete. In Chapter 5, we will study the expressive power of our framework compared to the AB approach.

Chapter 3

Containment of Parametric Conjunctive Queries

This chapter is devoted to the study of containment of conjunctive queries in the parametric framework.

3.1 Preliminaries

Containment and equivalence are the central notions used in query optimization in relational and deductive databases [Ull89]. Let us quickly review some basic notions from classical database theory, which will be used in the rest of this chapter. A *conjunctive query* is a rule of the form

$$Q: \quad p(X_1, \ldots, X_n) \longleftarrow q_1(Y_1, \ldots, Y_m), \ldots, q_k(Z_1, \ldots, Z_\ell)$$

where the q_i's are predicates referring to relations in the database while p can be thought of as the query predicate. It is assumed that the rule is range restricted, that is, every variable X_i in the rule head also appears in the body. Given a database D which has relations corresponding to the predicates in the body of Q, we define an *instantiation* of Q w.r.t. D as a ground substitution on the variables of Q. An instantiation θ of Q w.r.t. D is said to be *valid* provided the body of Q is true w.r.t. D, under the instantiation θ. In future, we mean only valid instantiation whenever we refer to instantiations. The set of all (valid) instantiations of Q w.r.t. D is denoted $inst(Q, D)$.

The *evaluation* of Q on D is defined as $Q(D) = \{p(X_1, \ldots, X_n)\theta \mid \theta \in inst(Q, D)\}$ whenever Q is viewed as a set, and as $\{\!\{p(X_1, \ldots, X_n)\theta \mid \theta \in inst(Q, D)\}\!\}$, whenever $Q(D)$ is viewed as a multiset. In both cases, notice that the input database D is always a classical (i.e. set-based) database. The context will make it clear how $Q(D)$ is viewed. Given two conjunctive queries Q_1, Q_2 defining the same query predicate $p(X_1, \ldots, X_n)$, we say that Q_1 is *contained* in Q_2, denoted $Q_1 \subseteq Q_2$, provided for every input database D, $Q_1(D) \subseteq Q_2(D)$. Q_1 and Q_2 are *equivalent*, denoted $Q_1 \equiv Q_2$, provided $Q_1 \subseteq Q_2$ and $Q_2 \subseteq Q_1$.

A simple and powerful tool for the study of conjunctive query containment in the classical case is the *containment mapping* [CM77]. Let Q_1 and Q_2 be any two conjunctive queries as shown below in schematic form.

$$Q_1 : H \longleftarrow B_1, \ldots, B_k.$$
$$Q_2 : H' \longleftarrow C_1, \ldots, C_m.$$

A symbol mapping from Q_2 to Q_1 is a mapping h from the variables occurring in Q_2 to those occurring in Q_1. Such a mapping h induces a mapping from the predicates in Q_2 to those in Q_1, as follows: $h(r(U_1, \ldots, U_k)) =_{def} r(h(U_1), \ldots, h(U_k))$, where $r(U_1, \ldots, U_k)$ is a predicate appearing in the head or body of Q_2. A symbol mapping h is a *containment mapping* (c.m.), provided $h(H') = H$, and for every $1 \leq i \leq m$, $h(C_i) = B_j$, for some $1 \leq j \leq k$. In the classical case, conjunctive query containment is completely characterized by the existence of containment mappings.

Theorem 3.1.1 [CM77] *Let Q_1 and Q_2 be conjunctive queries. Then $Q_1 \subseteq Q_2$ iff there exists a c.m. from Q_2 to Q_1.*

In the case of classical relational and deductive databases, the above result has played a central role in the development of many query optimization algorithms. It is important to note that even though conjunctive query containment was originally developed for non-recursive queries, the idea and even containment mappings have been extensively applied for recursive query optimization in deductive databases (see Ullman [Ull89] for a comprehensive account). An interesting question to ask is *how the techniques developed for classical containment can be extended to the framework of deductive databases with any of the various uncertainty formalisms considered in this paper?* In this section, we classify conjunctive queries in the parametric framework

based on the underlying conjunction and disjunction functions independently of the uncertainty formalisms. For each class of queries, we develop a complete characterization of containment, using the basic tools of (classical) containment mappings.

Chaudhuri and Vardi [CV93] studied the optimization problem for standard conjunctive queries under multiset semantics. They show that optimization techniques from the set-theoretic setting do not carry over to the multiset-theoretic setting. Furthermore, they found that under the multiset semantics, two conjunctive queries are equivalent precisely when they are isomorphic, and hence unlike under the set-theoretic semantics, it is not possible to minimize conjunctive queries by removal of conjuncts. As a consequence, subgoal reordering is the only possible optimization method under the multiset semantics. Our work differs from theirs in that the conjuncts in our case are set-theoretic.

In a recent paper, Ioannidis and Ramakrishnan [IR95] considered the problem of query containment in a broad setting where relations may be interpreted as sets or as multisets (akin to the treatment of relations in commercial relational database management systems). This is particularly relevant to our work, given the importance of multisets in the context of deduction with uncertainty. They show that as long as a framework (for database querying) can be reduced to viewing relations as sets, containment is completely characterized by containment mappings. The following example shows that unfortunately simply classifying the combination functions as "set-based" and "multiset-based" is not meaningful w.r.t. the parametric framework.

Example 3.1.2 Consider a p-rule of the form $A \xleftarrow{\alpha} B, C; \langle f^d, f^p, f^c \rangle$, where A, B, C are any ground atoms, $\alpha \in \mathcal{T}$ is the certainty of the rule, f^c and f^p are the conjunction and propagation functions associated with the rule, and f^d is the disjunction function associated with the predicate of A. Let $\mathcal{T} = [0, 1]$ be the certainty lattice, and v be a valuation such that $v(B) = v(C) = 0.5$. Then a "set-based" conjunction function would ignore the duplicate copies of 0.5. However this is not logically correct, as the certainties are associated with different ground atoms. The problem is that such a conjunction function cannot distinguish between the above p-rule and the p-rule $A \xleftarrow{\alpha} B, B; \langle f^d, f^p, f^c \rangle$. ■

We argue that the classification of the functions used in a parametric framework that is appropriate for our purposes should be based on how the functions are related

to the appropriate lattice operations. Specifically, we should look at how the conjunction functions compare with the meet $\otimes$ of the certainty lattice, and the disjunction functions with the lattice join $\oplus$.

In classical case, containment is based on set theory. When dealing with programs in deductive databases with uncertainty, we have to also take into account the certainty associated with atoms. We refer to conjunctive queries in the parametric framework as *parametric conjunctive queries.* A parametric conjunctive query $\mathbf{Q}$ is a non-recursive p-rule of the form

$$H \xleftarrow{\alpha} B_1, \ldots, B_k; \langle f^d, f^p, f^c \rangle.$$

We use bold $\mathbf{Q}$'s to denote parametric conjunctive queries, and Q's to denote classical conjunctive queries. When $\mathbf{Q}$ is a parametric conjunctive query, we will also denote by Q the underlying classical conjunctive query obtained from $\mathbf{Q}$ by stripping all the parameters.

We develop some notations. An *annotated tuple* is an expression of the form $t : \alpha$, where t is an ordinary fact, and $\alpha \in \mathcal{T}$ is a certainty value in the underlying lattice. An *annotated relation* is a finite set of annotated tuples. An *annotated database* is a finite set of annotated relations. Let D be an annotated database and A be a ground atom. Then we let $D(A)$ denote the certainty associated by D with A. Computation in the parametric framework can be conveniently captured in terms of annotated relations and databases. Notice that a Herbrand structure is essentially an annotated database. We define the evaluation of $\mathbf{Q}$ on a database D as the annotated relation for the predicate defined by $\mathbf{Q}$, where $\mathbf{Q}$ is of the form $H \xleftarrow{\alpha} B_1, \ldots, B_k; \langle f^d, f^p, f^c \rangle$. Formally, for any ground atom A,

$$\mathbf{Q}(D)(A) = f^d(\{\!| f^p(\alpha, f^c(D(B_1\theta), \ldots, D(B_n\theta)) \mid \exists\theta : \ A = H\theta |\!\}).$$

As in the classical case, we refer to a substitution such as θ above as an *instantiation*, and denote the set of such instantiations by $inst(\mathbf{Q}, D)$. Throughout this chapter, we assume that the input databases to parametric queries are annotated databases, while input databases to classical queries are conventional relational databases. It is important to note that in both cases, the input databases do *not* contain duplicates. The following lemma links computation of queries in terms of the operator T_P to computation in terms of annotated databases.

Lemma 3.1.3 *Let* $\mathbf{Q} : H \xleftarrow{\alpha} B_1, \ldots, B_k; \langle f^d, f^p, f^c \rangle$ *be a parametric conjunctive query, and* D *any annotated database. Then for any ground atom* A *corresponding to the predicate defined by* $\mathbf{Q}$, $\mathbf{Q}(D)(A) = lfp(T_{P \cup D})(A)$, *where* P *is the p-program containing just* $\mathbf{Q}$.

Proof. Trivial. ∎

The sole usefulness of this lemma is that it is more convenient to treat conjunctive query evaluation in terms of the notion of input/output (a la the definition $\mathbf{Q}(D)$) than in terms of the classical logic programming operator T_P. This lemma shows that the two treatments are equivalent as far as the predicate defined by the query is concerned.

In our study of containment of parametric conjunctive queries, we will find it useful to define containment between a pair of parametric queries based on several criteria. The following definitions make this precise.

Definition 3.1.4 *Let* $\mathbf{Q} : H \xleftarrow{\alpha} B_1, \ldots, B_k; \langle f^d, f^p, f^c \rangle$ *be a parametric conjunctive query,* D *an input annotated database, and* θ *an instantiation of* $\mathbf{Q}$ *w.r.t.* D. *Then* $\mathbf{Q}^\theta(D) = H\theta : \beta$, *where* $\beta = f^p(\alpha, f^c(\{\!|D(B_1\theta), \ldots, D(B_k\theta)|\!\}))$. *Intuitively,* $\mathbf{Q}^\theta(D)$ *denotes the derivation by* $\mathbf{Q}$ *of an instance of* H *(corresponding to* θ*) from the input database* D.

The partial order $\preceq$ on the certainty lattice $\mathcal{T}$ can be extended to annotated tuples in the obvious way: for any pair of annotated tuples $A : \alpha$ and $B : \beta$, we have $A : \alpha \preceq B : \beta$ iff A and B are identical ground atoms and $\alpha \preceq \beta$ according to the lattice ordering.

Definition 3.1.5 *Let* $\mathbf{Q}_1$ *and* $\mathbf{Q}_2$ *be any parametric conjunctive queries. We say that* $\mathbf{Q}_1 \subseteq \mathbf{Q}_2$, *provided for every EDB* D *and for every instantiation* $\theta_1 \in inst(\mathbf{Q}_1, D)$, *there exists an instantiation* $\theta_2 \in inst(\mathbf{Q}_2, D)$, *such that* $\mathbf{Q}_1^{\theta_1}(D) \preceq \mathbf{Q}_2^{\theta_2}(D)$, *where* $\preceq$ *denotes the partial order on annotated tuples, defined above.*

The set-theoretic containment defined in Definition 3.1.5 captures the intuition that on any input database D, every derivation by $\mathbf{Q}_1$ is matched or bettered by some derivation by $\mathbf{Q}_2$.

Definition 3.1.6 *Let* $\mathbf{Q}_1$ *and* $\mathbf{Q}_2$ *be as above. We say that* $\mathbf{Q}_1 \dot{\subseteq} \mathbf{Q}_2$, *provided for every input database* D, *there exists a 1-1 function* $m : inst(\mathbf{Q}_1, D) \rightarrow inst(\mathbf{Q}_2, D)$, *such that for each instantiation* $\theta \in inst(\mathbf{Q}_1, D)$, $\mathbf{Q}_1^{\theta}(D) \preceq \mathbf{Q}_2^{m(\theta)}(D)$.

The notion of *multiset-based containment* defined above captures the intuition that on any input database D, every derivation by $\mathbf{Q}_1$ is matched or bettered by a *unique* derivation by $\mathbf{Q}_2$.

We will also find use for the following notions of containment on classical conjunctive queries.

Definition 3.1.7 *Let* Q_1 *and* Q_2 *be classical conjunctive queries. Then* $Q_1 \subseteq Q_2$, *provided for every input database* D, *every tuple in* $Q_1(D)$ *is also in* $Q_2(D)$, *i.e.* $Q_1(D) \subseteq Q_2(D)$. *This is the conventional notion of containment.*

Definition 3.1.8 *Let* Q_1 *and* Q_2 *be classical conjunctive queries. Then* $Q_1 \dot{\subseteq} Q_2$, *provided for every input database* D, *every tuple in* $Q_1(D)$ *is also in* $Q_2(D)$, *and with no less multiplicity. That is,* $Q_1(D) \dot{\subseteq} Q_2(D)$. *(See Section 1.4 for a definition of* $\dot{\subseteq}$*).*

Definition 3.1.9 [Containment and Equivalence] *Let* $\mathbf{Q}_1$ *and* $\mathbf{Q}_2$ *be any two parametric conjunctive queries, and* D *any annotated database. We write* $\mathbf{Q}_1(D) \preceq \mathbf{Q}_2(D)$, *provided for every ground atom* $A \in B_P$ *corresponding to the query predicate, we have* $\mathbf{Q}_1(D)(A) \preceq \mathbf{Q}_2(D)(A)$, *according to the lattice ordering* $\preceq$. *We say* $\mathbf{Q}_1$ *is* contained *in* $\mathbf{Q}_2$, *denoted* $\mathbf{Q}_1 \leq \mathbf{Q}_2$, *provided for every annotated database* D, $\mathbf{Q}_1(D) \preceq \mathbf{Q}_2(D)$. *We say* $\mathbf{Q}_1$ *is* equivalent *to* $\mathbf{Q}_2$, *denoted* $\mathbf{Q}_1 \equiv \mathbf{Q}_2$, *iff* $\mathbf{Q}_1 \leq \mathbf{Q}_2$ *and* $\mathbf{Q}_2 \leq \mathbf{Q}_1$.

We write $f_1^p \preceq f_2^p$, provided $f_1^p(\alpha, \beta) \preceq f_2^p(\alpha, \beta)$, for all $\alpha, \beta \in \mathcal{T}$. For conjunction (or disjunction) functions f_1 and f_2, we write $f_1 \preceq f_2$, provided $f_1(X) \preceq f_2(X)$, for every multiset X of certainty values.

Remarks. Note there are three kinds of "containment" considered in the above definitions, $\subseteq, \dot{\subseteq}$, and $\leq$. Of these, $\leq$ denotes containment under the semantics of our parametric framework. Our results will show that this reduces to classical containment $\subseteq$ (e.g. Theorems 3.3.9), and to containment under multiset semantics $\dot{\subseteq}$ (e.g., Theorems 3.3.17 and 3.3.18).

In order to characterize when $\mathbf{Q}_1 \leq \mathbf{Q}_2$, we assume that the corresponding parameters in these queries are in certain relationship, formalized as follows.

Definition 3.1.10 *Let* $\mathbf{Q}_1$ *and* $\mathbf{Q}_2$ *be the following parametric conjunctive queries.*

$$\begin{array}{lll} \mathbf{Q}_1: & H \xleftarrow{\alpha_1} B_1, \ldots, B_k; & \langle f_1^d, f_1^p, f_1^c \rangle. \\ \mathbf{Q}_2: & H' \xleftarrow{\alpha_2} C_1, \ldots, C_m; & \langle f_2^d, f_2^p, f_2^c \rangle. \end{array}$$

We say that $(\mathbf{Q}_1, \mathbf{Q}_2)$ *is an* admissible pair[1] *provided (i)* $\perp \prec \alpha_1 \preceq \alpha_2$, *(ii)* $f_1^p \preceq f_2^p$, *(iii)* $f_1^c \preceq f_2^c$, *and (iv)* $f_1^d \preceq f_2^d$. *Sometimes we say that* $\mathbf{Q}_1$ *and* $\mathbf{Q}_2$ *are an admissible pair to mean that* $(\mathbf{Q}_1, \mathbf{Q}_2)$ *is an admissible pair.*

Note that when $(\mathbf{Q}_1, \mathbf{Q}_2)$ is an admissible pair, $(\mathbf{Q}_2, \mathbf{Q}_1)$ need not be so. It follows from the definition above that any pair of parametric conjunctive queries with $\alpha \succ 1$ and with identical parameters is admissible. In the absence of such reasonable assumptions on $\mathbf{Q}_1$ and $\mathbf{Q}_2$, there may be no hope for a syntactic characterization of containment of conjunctive queries with uncertainty. In most containment problems arising in practice (e.g., eliminating subgoals) the pair of queries considered is always admissible.

By a c.m. from $\mathbf{Q}_2$ to $\mathbf{Q}_1$, we mean a c.m. from Q_2 to Q_1. The necessity of containment mappings for containment is established next.

3.2 Necessary Condition for Containment

Lemma 3.2.1 *Let* $\mathbf{Q}_1$ *and* $\mathbf{Q}_2$ *be any pair of parametric conjunctive queries, shown below.*

$$\begin{array}{lll} \mathbf{Q}_1: & H \xleftarrow{\alpha_1} B_1, \ldots, B_k; & \langle f_1^d, f_1^p, f_1^c \rangle. \\ \mathbf{Q}_2: & H' \xleftarrow{\alpha_2} C_1, \ldots, C_m; & \langle f_2^d, f_2^p, f_2^c \rangle. \end{array}$$

Suppose $\mathbf{Q}_1 \leq \mathbf{Q}_2$. *Then, there exists a c.m. from* $\mathbf{Q}_2$ *to* $\mathbf{Q}_1$.

Proof. The proof is by an extension of Chandra and Merlin's proof for the classical case [CM77]. Suppose $\mathbf{Q}_1 \leq \mathbf{Q}_2$. Now, consider a database formed as follows. For each

[1] Ioannidis and Ramakrishnan [IR95] made similar assumptions about Q_1 and Q_2 in their treatment of containment.

variable X occurring in $\mathbf{Q}_1$, let c_X be a distinct constant. Let g be a mapping from the variables of $\mathbf{Q}_1$ to the constants constructed above such that $g(X) = c_X$, for each variable X in $\mathbf{Q}_1$. Clearly, g is a bijection. Let D be a database obtained by setting $D(r(g(U_1), \ldots, g(U_l)) = \top$, for every atom $r(U_1, \ldots, U_l)$ appearing in the body of $\mathbf{Q}_1$. Let $\sigma = \mathbf{Q}_1(D)(g(H))$. Since conjunction and propagation functions satisfy Property 10, and since $\alpha_i \succ \perp$, it follows that $\sigma \succ \perp$. Since $\mathbf{Q}_1 \leq \mathbf{Q}_2$, there must exist an instantiation θ such that for each atom C_i appearing in $\mathbf{Q}_2$'s body, $D(\theta(C_i)) \succ \perp$,[2] and such that $\mathbf{Q}_2(D)(\theta(H')) = \mathbf{Q}_2(D)(g(H)) \succ \sigma$. Now, the mapping $g^{-1} \circ \theta$ can be easily verified to be a c.m. from $\mathbf{Q}_2$ to $\mathbf{Q}_1$, just as in the classical case. ■

3.3 Sufficient Conditions for Containment

In general, the mere existence of a containment mapping from $\mathbf{Q}_2$ to $\mathbf{Q}_1$ is *not* sufficient for $\mathbf{Q}_1 \leq \mathbf{Q}_2$. This is because of the following observations.

1. Intuitively, for $\mathbf{Q}_1 \leq \mathbf{Q}_2$, we need to ensure that on any input database D, for each derivation of a ground atom A by $\mathbf{Q}_1$, there is a derivation of A by $\mathbf{Q}_2$, with no less certainty. Sometimes, this is not enough (see 3 below).
2. When $f_2^c \prec \otimes$, an *arbitrary* c.m. does *not* ensure that each derivation by $\mathbf{Q}_1$ will be matched or bettered by some derivation by $\mathbf{Q}_2$.
3. When the disjunction function $f_1^d \succ \dot{-}$, simply ensuring that each derivation by $\mathbf{Q}_1$ will be matched or bettered by some derivation by $\mathbf{Q}_2$ is *not* enough. Several derivations of the same atom by $\mathbf{Q}_1$ may "team up" and beat the derivations of the atom by $\mathbf{Q}_2$.

Let us illustrate observation 2 using the following example.

Example 3.3.1 Consider the following pair of parametric conjunctive queries.

$$\mathbf{Q}_1: \; p(X) \xleftarrow{1} q(X); \qquad \langle f^d, *, f^c \rangle.$$
$$\mathbf{Q}_2: \; p(X) \xleftarrow{1} q(X), q(X); \qquad \langle f^d, *, f^c \rangle.$$

where $\mathcal{T} = [0, 1]$, $f^c \prec \otimes$, and f^d is any disjunction function in $\mathcal{F}_d$. While there is a trivial c.m. from $\mathbf{Q}_2$ to $\mathbf{Q}_1$, it can be easily seen that taking $D = \{q(a) : \sigma\}$ as the

[2] Actually, $D(\theta(C_i))$ will be $\top$ for our particular database D.

input database, where σ is any element in the open interval $(0,1)$, the certainty of $p(a)$ derived by $\mathbf{Q}_2$ is strictly less than that derived by $\mathbf{Q}_1$, and thus $\mathbf{Q}_1 \not\leq \mathbf{Q}_2$. ■

The following example illustrates observation 3 above.

Example 3.3.2 Let $\mathbf{Q}_1$ and $\mathbf{Q}_2$ be the following pair of parametric conjunctive queries.

$$\mathbf{Q}_1: \; p(X,Y) \overset{[0.8,0.9]}{\longleftarrow} q(X,Y),\; q(X,Z); \qquad \langle \vee_{ind}, \wedge_{pc}, \wedge_{pc} \rangle.$$
$$\mathbf{Q}_2: \; p(X,Y) \overset{[0.8,0.9]}{\longleftarrow} q(X,Y); \qquad \langle \vee_{ind}, \wedge_{pc}, \wedge_{pc} \rangle.$$

Here, $\mathcal{T} = \mathcal{C}[0,1]$, the set of closed intervals in $[0,1]$, and $\vee_{ind}([\alpha_1,\beta_1],[\alpha_2,\beta_2]) = [\alpha_1+\alpha_2-\alpha_1\alpha_2,\ \beta_1+\beta_2-\beta_1\beta_2]$. Clearly, there exists a trivial c.m. from $\mathbf{Q}_2$ to $\mathbf{Q}_1$. Since $f_2^c = \wedge_{pc} = \otimes$, this implies that each derivation of $\mathbf{Q}_1$ is matched or bettered by some derivation of $\mathbf{Q}_2$ (see below). But since $f_1^d = \vee_{ind}$, (and hence $f_2^d \succ \oplus$), the multiplicity of derivations matters. For instance, consider $D = \{q(a,b) : [0.5,0.5],\ q(a,c) : [0.6,0.6]\}$. There are two derivations of $p(a,b)$ by $\mathbf{Q}_1$ on D, corresponding to the instantiations: $X \mapsto a$, $Y \mapsto b$, $Z \mapsto b$, and $X \mapsto a$, $Y \mapsto b$, $Z \mapsto c$. The associated certainties are: $[0.5,0.5]$ and $[0.5,0.5]$, which when combined using $\vee_{ind}$, lead to $[0.75,0.75]$. Whereas there is only one derivation of $p(a,b)$ by $\mathbf{Q}_2$ on D with certainty $[0.5,0.5]$. Thus, $\mathbf{Q}_1 \not\leq \mathbf{Q}_2$. ■

The approach we thus take for sufficiency is to classify the classes of parametric conjunctive queries based on the "behavior" of their underlying parameters w.r.t. the above observations. Specifically, we need to consider the following two cases.

Case of lattice-theoretic conjunction: The conjunction function f_2^c associated with $\mathbf{Q}_2$ coincides with the meet operator $\otimes$ in the certainty lattice.

Case of arbitrary conjunction: The conjunction function f_2^c is $\prec \otimes$.

Under each of the above two cases, we further distinguish the cases where the disjunction function f_1^d associated with $\mathbf{Q}_1$ coincides with the lattice join $\oplus$, and where $f_1^d \succ \oplus$, which are studied in Sections 3.3.1.1 and 3.3.1.2, respectively.

3.3.1 Case of Lattice-Theoretic Conjunction

In this section, we assume that the conjunction function f_2^c associated with $\mathbf{Q}_2$ coincides with the meet $\otimes$ operator in the certainty lattice $\mathcal{T}$. We will first consider the case in which $f_1^d = \oplus$. The case where $f_1^d \succ \oplus$ will then follow.

3.3.1.1 Case of Lattice-Theoretic Disjunction

The following result shows that the existence of a containment mapping is necessary and sufficient for containment in the case of lattice-theoretic conjunction and disjunction.

Theorem 3.3.3 *Let* $\mathbf{Q}_1$ *and* $\mathbf{Q}_2$ *be any admissible pair of parametric conjunctive queries, as shown below.*

$$\mathbf{Q}_1 : \; H \xleftarrow{\alpha_1} B_1, \ldots, B_k; \qquad \langle f_1^d, f_1^p, f_1^c \rangle.$$
$$\mathbf{Q}_2 : \; H' \xleftarrow{\alpha_2} C_1, \ldots, C_m; \qquad \langle f_2^d, f_2^p, f_2^c \rangle.$$

Suppose $f_2^c = \otimes$ *and* $f_1^d = \oplus$. *Then,* $\mathbf{Q}_1 \leq \mathbf{Q}_2$ *iff there exists a c.m. from* $\mathbf{Q}_2$ *to* $\mathbf{Q}_1$.

Proof. ($\Rightarrow$) Follows from Lemma 3.2.1.
($\Leftarrow$) Suppose the hypothesis of the theorem is satisfied and that there is a c.m., say h, from $\mathbf{Q}_2$ to $\mathbf{Q}_1$. Let D be any annotated input database. For each ground substitution g from the variables of $\mathbf{Q}_1$ to the constants occurring in D, consider the function $g \circ h$. Clearly $g \circ h$ is a ground substitution from the variables in $\mathbf{Q}_2$ to the constants in D. Moreover, by the definition of c.m., for each $i \in \{1, \cdots, m\}$, we have $h(C_i) = B_j$, for some $j \in \{1, \cdots, k\}$. Next, consider the tuple $g(H)$ derived by $\mathbf{Q}_1$ using g. Let $A \equiv g(H)$, for convenience. The certainty associated with A) is $\sigma_1 = f_1^p(\alpha_1, f_1^c(\{\!|D(g(B_1)), \ldots, D(g(B_k))|\!\}))$. Recall that $D(B)$ denotes the certainty associated with atom B in D. Clearly, $\mathbf{Q}_2$ will then derive the tuple $(g \circ h)(H') \equiv A$, with an associated certainty given by $\sigma_2 = f_2^p(\alpha_2, f_2^c(\{\!|D(g \circ h(C_1)), \ldots, D(g \circ h(C_m))|\!\}))$. Notice that $\otimes(\{\!|D(g \circ h(C_1)), \ldots, D(g \circ h(C_m))|\!\}) \succeq \otimes(\{\!|D(g(B_1)), \ldots, D(g(B_k))|\!\})$. From this and the facts that $f_1^c \preceq f_2^c$, $f_1^p \preceq f_2^p$, and $\perp \prec \alpha_1 \preceq \alpha_2$, it follows that $\sigma_1 \preceq \sigma_2$. Since the above argument holds for each ground substitution g used for any tuple A derived by $\mathbf{Q}_1$ from D, we have

$\oplus\{\!|\sigma_1 \mid \sigma_1 = f_1^p(\alpha_1, f_1^c(\{\!|D(g(B_1)),\ldots,D(g(B_k))|\!\})) \; \forall g : g(H) = A|\!\} \preceq$
$\oplus\{\!|\sigma_2 \mid \sigma_2 = f_2^p(\alpha_2, f_2^c(\{\!|D(g \circ h(C_1)),\ldots,D(g \circ h(C_m))|\!\})) \; \forall g : g(H) = A|\!\}$.
Since $\oplus = f_1^d \preceq f_2^d$, $\mathbf{Q}_1(D)(A) \preceq \mathbf{Q}_2(D)(A)$. Since D was an arbitrary database and A was any atom derived by $\mathbf{Q}_1$, we may thus conclude that $\mathbf{Q}_1 \leq \mathbf{Q}_2$. ■

The above result applies, for instance, to frameworks of van Emden [van86], Fitting [Fit88, Fit91], the deductive IST of Lakshmanan and Sadri [LS94b], and Dubois et al. [DLP91]. This result is also applicable to a class of conjunctive queries in the probabilistic deductive databases of Lakshmanan and Sadri [LS94a], when the conjunction and disjunction modes used are *positive correlation*, i.e., $\wedge_{pc}$ and $\vee_{pc}$. It should be pointed out that in a recent work, Lakshmanan and Sadri [LS96a] studied containment and uniform containment of conjunctive queries in deductive IST and established several results. By contrast, our results here apply to a wide variety of IB frameworks.

3.3.1.2 Case of Arbitrary Disjunction

We now study containment of parametric conjunctive queries when $f_2^c = \otimes$ and $f_1^d \succ \ominus$.

Recall that in our context the input relations are sets. However, as was shown earlier, we need to consider the collection of "derivations" of the atoms as multisets, in general. If the disjunction functions associated with the head predicates coincide with the lattice join $\dot{\ominus}$, then we can safely consider the collection as sets. Otherwise, simply ensuring that there is a "matching" or a "better" derivation by $\mathbf{Q}_2$ for each derivation by $\mathbf{Q}_1$ is not enough, since several "small" derivations by $\mathbf{Q}_1$ can team up and "beat" a few "large" derivations by $\mathbf{Q}_2$. It is thus important for us, even before we bring in the parameters into play in our study of containment, to answer the following question in the context of classical conjunctive queries. Given any pair of conjunctive queries Q_1 and Q_2, when can we say that $Q_1(D)$ as a multiset is "included" in $Q_2(D)$, for every *classical* (i.e. set-based) input database D? That is, when can we ensure that for every database D and every atom A, the number of derivations of A by Q_2 on D is no less than the number of derivations of A by Q_1 on D? When this holds, we write $Q_1(D) \dot{\subseteq} Q_2(D)$, as per Definition 3.1.8.

Therefore, given that $f_1^d \succ \ominus$, our approach in this case is (1) to make sure that

for every derivation of a ground atom A by $\mathbf{Q}_1$ on any input database D, there exists a distinct derivation of A by $\mathbf{Q}_2$ on D with at least the same certainty, and (2) to determine when the multiset of tuples obtained by applying Q_1 (i.e. $\mathbf{Q}_1$ without the parameters) to any input database is contained in $Q_2(D)$. In other words, when $Q_1 \dot{\subseteq} Q_2$? Our next result provides a sufficient condition for (2). Before that, we need the following definition.

Definition 3.3.4 *Let Q_1 and Q_2 be any conjunctive queries, and $Var(Q_i)$ be the set of variables used in Q_i, for $i = 1, 2$. Then, a c.m. θ from Q_2 to Q_1 is a* variable onto c.m. *provided, for every variable $V_1 \in Var(Q_1)$ there exists a variable $V_2 \in Var(Q_2)$ such that $\theta(V_2) = V_1$.*

Theorem 3.3.5 *Let Q_1 and Q_2 be any classical conjunctive queries as shown below in schematic form.*

$$Q_1 : H \longleftarrow B_1, \ldots, B_k.$$
$$Q_2 : H' \longleftarrow C_1, \ldots, C_m.$$

Then, for every input database D, $Q_1(D) \dot{\subseteq} Q_2(D)$, if there exists a variable onto c.m. from Q_2 to Q_1.

Proof. Let h be any variable onto c.m. from Q_2 to Q_1, and A be any ground atom in $Q_1(D)$. Note that $Q_i(D)$ is a multiset of tuples, for $i = 1, 2$. Let also $inst(Q_1, D)$ be the set of instantiations of Q_1 w.r.t. D, and $inst_A(Q_1, D) =_{def} \{\theta \in inst(Q_1, D) \mid H\theta = A\}$. Similarly, we define $inst_A(Q_2, D)$. Note that for every $\theta \in inst(Q_1, D)$, $B_i\theta$ is true in D, for all $1 \leq i \leq k$. To show that $Q_1(D) \dot{\subseteq} Q_2(D)$, we will show that $|inst_A(Q_1, D)| \leq |inst_A(Q_2, D)|$, where $|S|$ denotes the cardinality of set S. Now, define the mapping $f : inst_A(Q_1, D) \rightarrow inst_A(Q_2, D)$ such that $f(\theta) = \theta \circ h$. (Note that $\theta \circ h \in inst_A(Q_2, D)$, $\forall \theta \in inst_A(Q_1, D)$.) We will show that f is 1-1, from which follows the result. Suppose there are substitutions θ_1 and θ_2 in $inst_A(Q_1, D)$, such that $\theta_1 \neq \theta_2$. In this case, there is a variable $V \in Var(Q_1)$ such that $\theta_1(V) \neq \theta_2(V)$. Since h is a variable onto c.m., there is a variable $U \in Var(Q_2)$ such that $h(U) = V$. Then, $\theta_1 \circ h(U) = \theta_1(V) \neq \theta_2(V) = \theta_2 \circ h(U)$, which implies $f(\theta_1) \neq f(\theta_2)$ and hence f is 1-1. From this, we may conclude that $|inst_A(Q_1, D)| \leq |inst_A(Q_2, D)|$, which was to be shown. ■

The following example illustrates the existence of a variable onto c.m. is not necessary for the containment $Q_1 \dot{\subseteq} Q_2$.

Example 3.3.6 *Consider the following conjunctive queries Q_1 and Q_2.*

$$Q_1: \quad p(X_1, X_2) \longleftarrow e(X_1, X_2),\ e(X_2, U),\ e(U, V), e(U, W),\ e(W, T).$$
$$Q_2: \quad p(X_1, X_2) \longleftarrow e(X_1, X_2),\ e(X_2, U'), e(U', V'),\ e(U', W').$$

It can be shown that $Q_1 \dot{\subseteq} Q_2$, although there is no variable onto c.m. from Q_2 to Q_1. ■

Ioannidis and Ramakrishnan [IR95] study the problems of query containment and equivalence in a very general setting, where relations may be viewed as sets or multisets. Their work is applicable to containment of uncertain queries as well. They abstract various settings in terms of label systems, and classify them broadly into types A, B, A', and B', where A and A' are essentially set based, while B and B' are multiset based. Intuitively, the idea is to extend the relational model by attaching to each tuple a *label*, drawn from some special domain. The usage of label conform to to specific rules that enables one interpret them using appropriate semantics. Their results on containment for type B and type A' label systems are particularly relevant for our goal in this chapter. Type B label system captures multiset-based semantics. They show that a type B conjunctive query Q_1 is contained in another query Q_2 iff there is a *subgoal onto* c.m. from Q_2 to Q_1. (A c.m. is subgoal onto provided the mapping induced by it from subgoals of Q_2 to those of Q_1 is onto.) The key thing to note about containment for type B label systems is that *both the input and output* could be *multiset-based.* Specifically, the input is allowed to be a database whose relations are multisets. The following example illustrates this subtlety and shows that when the input is restricted to be classical (set-based) databases, subgoal onto containment mappings are *not* necessary for multiset-based containment.

Example 3.3.7 *Consider the following pair of conjunctive queries.*

$$Q_1: \ p(X, Y) \longleftarrow a(X, Y),\ b(X, Y).$$
$$Q_2: \ p(X, Y) \longleftarrow a(X, Y),\ a(X, Z).$$

The mapping $X \mapsto X, Y \mapsto Y, Z \mapsto Y$ from Q_2 to Q_1 is a variable onto (but not subgoal onto) c.m. from Q_2 to Q_1, and it is necessary and sufficient for $Q_1 \dot{\subseteq} Q_2$,

as proved in Theorem 3.3.8. Note that in our context, a subgoal onto c.m. is not *necessary* for $\dot{\subseteq}$. ■

For type A' label systems, Ioannidis and Ramakrishnan [IR95] show that a sufficient condition for containment is the existence of a variable onto containment mapping. While specific multiset-based systems such as MYCIN can be modeled under their type A' label system, type A' is not *inherently* multiset-based and does not *force* the output of queries to be multisets. Besides, their results do not yield a *necessary and sufficient* condition for multiset-based containment when the input is restricted to classical databases. Thus, their results on containment of conjunctive queries in various label systems do not directly yield a characterization of $Q_1 \dot{\subseteq} Q_2$.

We show in the following theorem that if Q_1 and Q_2 are classical conjunctive queries such that all the subgoals of Q_1 are distinct, then the existence of a variable onto c.m. is both necessary and sufficient for $Q_1 \dot{\subseteq} Q_2$, i.e., for any derivation of an atom A by Q_1 on any input database D, there is a distinct derivation of A by Q_2 on D. This result will play a key role in obtaining a characterization of containment of parametric conjunctive queries later in this chapter. Before that, we need the following definitions. We say that a conjunctive query Q has *distinct* subgoals, provided no predicate symbol appears more than once in the body. In a conjunctive query, every variable appearing in the head is called *distinguished variable*, and those appearing only in the body are *indistinguished variables*.

Theorem 3.3.8 *Let Q_1 be any conjunctive query which has distinct subgoals, and Q_2 be any conjunctive query. Then $Q_1 \dot{\subseteq} Q_2$ iff there exists a variable onto c.m. from Q_2 to Q_1.*

Proof. ($\Leftarrow$) Follows from Theorem 3.3.5.
($\Rightarrow$) The necessity of the existence of a c.m. follows from Theorem 3.1.1 and the fact that $Q_1 \dot{\subseteq} Q_2$ implies $Q_1 \subseteq Q_2$. So suppose there is a c.m. but no variable onto c.m. from Q_2 to Q_1. Let $h : Q_2 \rightarrow Q_1$ be any c.m. and let V be a variable in Q_1's body which is not covered by h. Let $q(\ldots, V, \ldots)$ be any subgoal in Q_1 which contains an occurrence of V. Clearly, Q_2 cannot contain any subgoal involving q, for if it did, such a subgoal would only be mapped to $q(\ldots, V, \ldots)$ in Q_1, contradicting our assumption that V is not covered by h. As a result, we see that V will not be covered by *any*

c.m. from Q_2 to Q_1. Another observation is that V must be a non-distinguished variable, since all distinguished variables must be covered by containment mappings, by definition. Consider the input database D, obtained as follows.

Let θ_1 be an instantiation that maps every variable Z other than V occurring in Q_1's body to a distinct constant c_Z, and maps V to the constant c_V^1. Let θ_2 be an instantiation identical to θ_1 except that it maps V to c_V^2, where $c_V^1 \neq c_V^2$ and both are distinct from all other constants. Now, $D = \{B\theta_1 \mid B$ appears in Q_1's body$\} \cup \{B\theta_2 \mid B$ appears in Q_1's body$\}$. By construction, we can see the following: (1) for every subgoal B_i not containing V, D has exactly one tuple; (2) For every subgoal B_j containing V, D has exactly two tuples. Since no c.m. from Q_2 to Q_1 can cover V, it is clear that no instantiation of Q_2 w.r.t. D will map any variable of Q_2 to either of c_V^1 or c_V^2. Let ι_1 and ι_2 be any two instantiations of Q_2 w.r.t. D, and let Y be any variable in Q_2's body. Let s be any subgoal in Q_2's body containing Y. It follows that both ι_1 and ι_2 must map $s(\ldots, Y, \ldots)$ in Q_2's body to tuples corresponding to relation s in D. However, by construction the relation for s has a unique tuple in D, since s cannot possibly contain V. This implies $\iota_1(Y) = \iota_2(Y)$, since Y is an arbitrary variable in Q_2, and we see that $\iota_1 = \iota_2$. Since ι_1 and ι_2 are arbitrary, this shows there is at most one instantiation of Q_2 w.r.t. D. However, there are two instantiations of Q_1 w.r.t. D, viz., θ_1 and θ_2 above. We thus have $Q_1(D) \dot{\not\subseteq} Q_2(D)$. This shows the necessity of a variable onto c.m. ■

Theorem 3.3.8 yields the following result for parametric conjunctive queries.

Theorem 3.3.9 *Let* $\mathbf{Q}_1$ *and* $\mathbf{Q}_2$ *be any pair of admissible parametric conjunctive queries, shown below.*

$$\mathbf{Q}_1 : \quad H \xleftarrow{\alpha_1} B_1, \ldots, B_k; \qquad \langle f_1^d, f_1^p, f_1^c \rangle.$$
$$\mathbf{Q}_2 : \quad H' \xleftarrow{\alpha_2} C_1, \ldots, C_m; \qquad \langle f_2^d, f_2^p, f_2^c \rangle.$$

where $\mathbf{Q}_1$ *has distinct subgoals. Suppose* $f_2^c = \otimes$ *and* $f_1^d \succ \oplus$. *Then,* $\mathbf{Q}_1 \leq \mathbf{Q}_2$ *iff there exists a variable onto c.m. from* $\mathbf{Q}_2$ *to* $\mathbf{Q}_1$.

Proof. When $f_1^d \succ \oplus$, $\mathbf{Q}_1 \leq \mathbf{Q}_2$ iff $\mathbf{Q}_1 \dot{\subseteq} \mathbf{Q}_2$. Since $f_2^c = \otimes$ and $\mathbf{Q}_1$,$\mathbf{Q}_2$ are an admissible pair, we have $\mathbf{Q}_1 \leq \mathbf{Q}_2$ iff $Q_1 \dot{\subseteq} Q_2$. The result then follows from Theorem 3.3.8. ■

The above result is applicable to conjunctive queries in the probabilistic deductive databases of Lakshmanan and Sadri [LS94a], when the conjunction function f_2^c used

in $\mathbf{Q}_1$ is $\wedge_{pc}$ and the disjunction function f_1^d used in $\mathbf{Q}_1$ is, e.g., $\vee_{ind}$, the *independent* mode.

3.3.2 Case of Arbitrary Conjunctions

Suppose the conjunction function associated with $\mathbf{Q}_2$ is not the meet $\otimes$ of the certainty lattice. We need the following definition.

Definition 3.3.10 *Let $h : Q_2 \rightarrow Q_1$ be a c.m. Then h is* subgoal 1-1, *provided the mapping induced by h on the subgoals of Q_2 is 1-1.*

Theorem 3.3.11 *Let $\mathbf{Q}_1$,$\mathbf{Q}_2$ be any admissible pair of parametric conjunctive queries, shown below.*

$$\mathbf{Q}_1 : \; H \xleftarrow{\alpha_1} B_1, \ldots, B_k; \qquad \langle f_1^d, f_1^p, f_1^c \rangle.$$
$$\mathbf{Q}_2 : \; H' \xleftarrow{\alpha_2} C_1, \ldots, C_m; \qquad \langle f_2^d, f_2^p, f_2^c \rangle.$$

Suppose $f_2^c \prec \otimes$. Then $\mathbf{Q}_1 \subseteq \mathbf{Q}_2$ if *there exists a subgoal 1-1 c.m. from $\mathbf{Q}_2$ to $\mathbf{Q}_1$.*

Proof. Let $h : \mathbf{Q}_2 \rightarrow \mathbf{Q}_1$ be any subgoal 1-1 c.m. from $\mathbf{Q}_2$ to $\mathbf{Q}_1$. Let D be an annotated input database. Let $D(A)$ denote the certainty of atom A in D. Consider any derivation of a ground atom A by $\mathbf{Q}_1$ on D. Let θ be the associated instantiation and σ_1 be the certainty derived. Then, $\sigma_1 = f_1^p(\alpha_1, f_1^c(\{\!|D(B_1\theta), \ldots, D(B_k\theta)|\!\}))$. On the other hand, $\sigma_2 = f_2^p(\alpha_2, f_2^c(\{\!|D(h(C_1)\theta), \ldots, D(h(C_m)\theta)|\!\})) \succeq \sigma_1$, where σ_2 is the certainty associated with the corresponding derivation of A by $\mathbf{Q}_2$ on D, and the associated instantiation is $\theta \circ h$. This is because $\{\!|D(h(C_i)\theta) \mid 1 \leq i \leq m|\!\} \dot{\subseteq} \{\!|D(B_j\theta) \mid 1 \leq j \leq k|\!\}$. ■

Corollary 3.3.12 *Let $\mathbf{Q}_1$,$\mathbf{Q}_2$ be any pair of admissible parametric conjunctive queries, as defined above. Suppose $f_2^c \prec \otimes$ and $f_1^d = \oplus$. Then $\mathbf{Q}_1 \leq \mathbf{Q}_2$* if *there exists a subgoal 1-1 c.m. from $\mathbf{Q}_2$ to $\mathbf{Q}_1$.*

Corollary 3.3.13 *Let $\mathbf{Q}_1$,$\mathbf{Q}_2$ be any pair of admissible parametric conjunctive queries, as defined above. Suppose $f_2^c \prec \otimes$ and $f_1^d \succ \oplus$. Then $\mathbf{Q}_1 \leq \mathbf{Q}_2$* if *there exists a c.m. from $\mathbf{Q}_2$ to $\mathbf{Q}_1$ which is subgoal 1-1 and variable onto.*

The following example illustrates that the existence of a subgoal 1-1 c.m. is *not* necessary for the containment $\mathbf{Q}_1 \leq \mathbf{Q}_2$.

Example 3.3.14 *Consider the following parametric conjunctive queries,*

$$\mathbf{Q}_1: \; p(X) \xleftarrow{\epsilon} q(X); \qquad \langle f^d, *, * \rangle.$$
$$\mathbf{Q}_2: \; p(X) \xleftarrow{1} q(X), q(X); \qquad \langle f^d, *, f_2^c \rangle.$$

where $f_2^c \prec \otimes$, and ϵ is a certainty value in $(0, 1]$ determined as follows. Suppose D contains only $q(a)$ with the associated certainty δ. The certainty of $p(a)$ derived from $\mathbf{Q}_2$ is $f_2^c(\{\!|\delta, \delta|\!\})$, and the one derived from $\mathbf{Q}_1$ is $\epsilon * \delta$. Since $f_2^c \prec \otimes$, we know that $f_2^c(\{\!|\delta, \delta|\!\}) \prec \otimes(\{\!|\delta, \delta|\!\}) = \delta$. Let $f_2^c(\{\!|\delta, \delta|\!\}) \succeq \delta - \epsilon * \delta$. Now, for the containment $\mathbf{Q}_1 \leq \mathbf{Q}_2$ to hold, we should have $\delta - \epsilon * \delta \succeq \epsilon * \delta$, which holds for every ϵ such that $\epsilon \preceq 0.5$. ■

Theorem 3.3.15 *Let $\mathbf{Q}_1$,$\mathbf{Q}_2$ be any admissible pair of parametric conjunctive queries. Suppose $\mathbf{Q}_2$ has distinct subgoals. Then $\mathbf{Q}_1 \subseteq \mathbf{Q}_2$ iff there exists a c.m. from $\mathbf{Q}_2$ to $\mathbf{Q}_1$.*

Proof. Follows immediately from Theorem 3.3.11 upon noting that all c.m.'s from $\mathbf{Q}_2$ to $\mathbf{Q}_1$ are necessarily subgoal 1-1. ■

Theorem 3.3.16 *Let $\mathbf{Q}_1$.$\mathbf{Q}_2$ be any pair of admissible parametric conjunctive queries with identical parameters. Suppose that $\mathbf{Q}_1$ has distinct subgoals, and that $f^c \prec \ominus$. Then $\mathbf{Q}_1 \subseteq \mathbf{Q}_2$ iff there is a subgoal 1-1 c.m. from $\mathbf{Q}_2$ to $\mathbf{Q}_1$.*

Proof. ($\Leftarrow$) Follows from Theorem 3.3.11.
($\Rightarrow$) Suppose there is no subgoal 1-1 c.m. from $\mathbf{Q}_2$ to $\mathbf{Q}_1$. Let $h : \mathbf{Q}_2 \rightarrow \mathbf{Q}_1$ be any c.m. Let S be the set of all subgoals in $\mathbf{Q}_1$ which are not mapped to by h. S is possibly empty. Our first observation is that if a subgoal of $\mathbf{Q}_1$ is not mapped to by h, it cannot be mapped to by any c.m. from $\mathbf{Q}_2$ to $\mathbf{Q}_1$. To see why, suppose $q(Y_1, \ldots, Y_p)$ is such a subgoal. Clearly, $\mathbf{Q}_2$ cannot have a subgoal involving predicate q, as the subgoals of $\mathbf{Q}_1$ are distinct. This in turn implies no c.m. from $\mathbf{Q}_2$ to $\mathbf{Q}_1$ can map to $q(Y_1, \ldots, Y_p)$. Let θ be an instantiation which maps each variable V in $\mathbf{Q}_1$ to a distinct constant c_V. Now, consider the input database D, constructed as

follows. For each subgoal $r(Z_1, \ldots, Z_n)$ of $\mathbf{Q}_1$ not in S, D contains a unique annotated tuple $r(c_{Z_1}, \ldots, c_{Z_n}) : \sigma$, where $\sigma \in \mathcal{T}$ is any certainty such that $\bot \prec \sigma \prec \top$. For each subgoal in S of the form $s(W_1, \ldots, W_\ell)$, D contains a unique annotated tuple $r(c_{W_1}, \ldots, c_{W_\ell}) : \top$. Now, we claim that $\mathbf{Q}_1(D) \not\subseteq \mathbf{Q}_2(D)$, from which the theorem follows. To see why this claim is true, consider the derivation of the head atom corresponding to the instantiation θ. First, notice that the subgoals in S play no role as the tuples corresponding to them in D are annotated with $\top$, which by Property 10, is an identity w.r.t. f^c. All subgoals of $\mathbf{Q}_1$ not in S correspond to tuples in D with annotation $\prec \top$. Let $\mathbf{Q}_1 : H \xleftarrow{\alpha} B_1, \ldots, B_k; \langle f^d, f^p, f^c \rangle$, and $\mathbf{Q}_2 : H' \xleftarrow{\alpha} C_1, \ldots, C_m; \langle f^d, f^p, f^c \rangle$. Consider $\mathbf{Q}_1^\theta(D)$ and $\mathbf{Q}_2^\pi(D)$, where $\pi \in inst(\mathbf{Q}_2, D)$ is an arbitrary instantiation, such that $H\theta = H'\pi$. Clearly, $\theta^{-1} \circ \pi$ is a c.m. from $\mathbf{Q}_2$ to $\mathbf{Q}_1$, since θ is a bijection. By our assumption, $\theta^{-1} \circ \pi$ cannot be subgoal 1-1 from $\mathbf{Q}_2$ to $\mathbf{Q}_1$, and hence π maps distinct subgoals of $\mathbf{Q}_2$, say C_i and C_j, to the same ground atom, say $B_\ell\theta$, in D. Let X_1 be the multiset obtained from $\{\!\{D(B_i\theta) \mid 1 \leq i \leq k\}\!\}$ by deleting all copies of $\top$ and let X_2 be similarly obtained from $\{\!\{D(C_j\pi) \mid 1 \leq j \leq m\}\!\}$. From the above arguments, it is clear that $\forall \gamma \in X_1$, the number of copies of γ in X_1 is no more than the number of copies of γ in X_2, and there exists a certainty element β in X_1 such that the number of copies of β in X_1 is strictly less than the number of copies of β in X_2. Since $f^c \prec \otimes$, this implies $f^c(\{\!\{D(B_i\theta) \mid 1 \leq i \leq k\}\!\}) = f^c(X_1) \succ f^c(X_2) = f^c(\{\!\{D(C_j\pi) \mid 1 \leq j \leq m\}\!\})$. This in turn implies $\mathbf{Q}_1^\theta(D) = H\theta : f^p(a, f^c(\{\!\{D(B_i\theta) \mid 1 \leq i \leq k\}\!\})) \succ H'\pi : f^p(a, f^c(\{\!\{D(C_j\pi) \mid 1 \leq j \leq m\}\!\})) = \mathbf{Q}_2^\theta(D)$. Our claim follows from this and the fact that π is an arbitrary instantiation in $inst(\mathbf{Q}_2, D)$. ■

Our results in this chapter yield necessary and sufficient conditions for containment of various classes of parametric conjunctive queries. In particular, these results are applicable to conjunctive queries in the framework of Lakshmanan and Sadri [LS94a] which in addition to the lattice theoretic conjunction and disjunction functions, allows some other such functions. The results can be also applied to conjunctive queries in MYCIN [BS75] (see Example 1.2.5). The following theorems can be proved by direct application of the previous results.

Theorem 3.3.17 *Let $\mathbf{Q}_1$ and $\mathbf{Q}_2$ be an admissible pair of parametric conjunctive queries. Suppose $\mathbf{Q}_2$ has distinct subgoals and $f_2^c \prec \otimes$.*
(1) Suppose $f_1^d = \oplus$. Then $\mathbf{Q}_1 \leq \mathbf{Q}_2$ iff there exists a c.m. from $\mathbf{Q}_2$ to $\mathbf{Q}_1$.

(2) Suppose $f_1^d \succ \oplus$. *Then* $\mathbf{Q}_1 \leq \mathbf{Q}_2$ *iff there exists a variable onto c.m. from* $\mathbf{Q}_2$ *to* $\mathbf{Q}_1$, *provided* $\mathbf{Q}_1$ *has distinct subgoals.*

Proof. Part (1) follows from Lemma 3.2.1 and proof of Theorem 3.3.15, upon noting that f_1^d ignores duplicates. For (2), the proof follows from Theorem 3.3.8 and the fact that $f_1^d \succ \oplus$. ■

Theorem 3.3.18 *Let* $\mathbf{Q}_1$ *and* $\mathbf{Q}_2$ *be any admissible pair of parametric conjunctive queries with identical parameters. Suppose* $\mathbf{Q}_1$ *has distinct subgoals and* $f^c \prec \otimes$.
(1) Suppose $f^d = \oplus$. *Then* $\mathbf{Q}_1 \leq \mathbf{Q}_2$ *iff there exists a subgoal 1-1 c.m. from* $\mathbf{Q}_2$ *to* $\mathbf{Q}_1$.
(2) Suppose $f^d \succ \oplus$. *Then* $\mathbf{Q}_1 \leq \mathbf{Q}_2$ *iff there exists a c.m. from* $\mathbf{Q}_2$ *to* $\mathbf{Q}_1$ *which is both subgoal 1-1 and variable onto.*

Proof. The proof of part (1) follows from Theorem 3.3.16. For (2), the proof follows from Theorems 3.3.8 and 3.3.16. ■

The above result can be used by an optimizer to determine, for instance, whether some subgoals in a rule body of a program can be removed, without altering the semantics of the program.

Table 1 contains a summary of the main results in this chapter on containment of parametric conjunctive queries.

3.4 Summary and Concluding Remarks

Several frameworks for deduction with uncertainty have already been proposed, so the first question is, why yet another framework. Our work was motivated by the observation that progress on query optimization for deductive databases with uncertainty has been quite limited. We wanted to approach this problem from a "generic" point of view, without necessarily focusing on any one framework. To this end, it was necessary to abstract away the two main factors characterizing known IB frameworks for deduction with uncertainty: (i) the underlying notion of uncertainty (fuzzy sets, probabilities, etc.), (ii) the way in which uncertainty is manipulated (e.g. whether based on some lattice operations, some specific combination functions, etc.). Our choice of the IB approach over the AB is because of the continuity of the fixpoint

Class	Proviso	Type of c.m.	Comments	Result
$f_2^c = \otimes,\ f_1^d = \oplus$	none	any	necessary & sufficient	Theorem 3.3.3
$f_2^c = \otimes,\ f_1^d \succ \oplus$	$\mathbf{Q}_1$ has distinct subgoals	variable onto	necessary & sufficient	Theorem 3.3.9
$f_2^c \prec \otimes,\ f_1^d = \oplus$	none	subgoal 1-1	sufficient	Corollary 3.3.12
	$\mathbf{Q}_2$ has distinct subgoals	any	necessary & sufficient	Theorem 3.3.17(1)
$f_2^c \prec \otimes,\ f_1^d \succ \oplus$	none	subgoal 1-1 & variable onto	sufficient	Corollary 3.3.13
	$\mathbf{Q}_1$,$\mathbf{Q}_2$ have distinct subgoals	variable onto	necessary & sufficient	Theorem 3.3.17(2)
Identical param's. $f^c \prec \otimes,\ f^d = \oplus$	$\mathbf{Q}_1$ has distinct subgoals	subgoal 1-1	necessary & sufficient	Theorem 3.3.18(1)
Identical param's. $f^c \prec \otimes,\ f^d \succ \oplus$	$\mathbf{Q}_1$ has distinct subgoals	subgoal 1-1 & variable onto	necessary & sufficient	Theorem 3.3.18(2)

Table 1: Summary of our main results on containment

operator in the former, which in general makes it more amenable to efficient computation. Also, based on our experience with several IB frameworks, query processing and unification procedure would be closer to the standard framework, thus allowing one to use the existing techniques and tools, with little or no modification, in the IB approach. Abstraction of the factors (i) and (ii) above has led to the parametric framework proposed in this thesis. With this as a basis, we were able to establish necessary and sufficient conditions for containment for various classes of parametric conjunctive queries.

Two benefits of our generic approach to this study are: (i) we can trace which of the properties (introduced in Section 2.1) on the families of combination functions are essential for our results on query optimization; (e.g. continuity is not a requirement); (ii) the parametric framework allows several different ways of uncertainty manipulation (e.g. some "set-based" and some "multiset-based") to be combined within one framework. As an illustration of the significance of our results, we showed in this chapter that large classes of programs in known IB frameworks can be optimized using classical tools.

An interesting direction for extending this work is on characterizing exactly when

the "no teaming up" property exhibited by unions of classical conjunctive queries carries over to the parametric framework. Future work should address the central questions of containment and equivalence of programs. For instance, how can we lift the chase technique (e.g. see Sagiv [Sag88]) to IB frameworks? Can we characterize exactly when program equivalence in IB frameworks reduces to classical equivalence?

Other important issues are termination and data complexity of the fixpoint evaluations of p-programs, which are studied in the following chapter. The study is organized based on a classification of p-programs according to the types of disjunction functions allowed in the parametric framework. We identify large classes of p-programs whose fixpoint evaluations on every input database (EDB) terminate in PTIME, in the size of the EDB. We also identify p-programs whose bottom-up fixpoint evaluations on some EDBs will not terminate. Finally, we identify p-programs whose fixpoint evaluations on some EDBs terminate in arbitrarily large, but finite, number of iterations, independent of the EDB size.

Chapter 4

Termination and Complexity of Bottom-up Fixpoint Evaluation

Given a logic program P and an EDB D, what is the number of iterations required for computing the fixpoint semantics of $P \cup D$ using the naive evaluation method? When P is in the standard framework of datalog, the answer is well known — PTIME in the number of constant symbols in P and D. What if P is in an extended framework with uncertainty? We will illustrate that unlike in the standard framework, three kinds of behaviors are possible when uncertainty is present, (i) termination in PTIME, (ii) non-termination[1], (iii) termination in an arbitrary large number of iterations, independent of the number of constants in the database.

4.1 Motivating Example

In the following example, we illustrate that analyzing/characterizing the termination behavior of programs with uncertainty is hard.

[1] Technically, this means termination at ω, as our fixpoint operator T_P is continuous.

Example 4.1.1 *Let P_1 be the following p-program.*

$$\begin{aligned}
r_0: &\ q(X,Y) \xleftarrow{\alpha} e(X,Y); && \langle max, f_0^p, _\rangle.\\
r_1: &\ p(X,Y) \xleftarrow{\beta} e(X,Y); && \langle ind, f_1^p, _\rangle.\\
r_2: &\ p(X,Y) \xleftarrow{\gamma} q(X,Z),\ p(Z,Y); && \langle ind, min, min\rangle.
\end{aligned}$$

where $\alpha \in (0,1)$, and β, γ are in $(0,1]$. f_0^p and f_1^p are any propagation functions allowed in the parametric framework, and $ind(\alpha,\beta) = \alpha + \beta - \alpha\beta$ is the disjunction function associated with the predicate symbols p and q. Note that the recursive p-rule r_2 uses the IDB subgoal $q(X,Z)$. Suppose $D = \{e(1,1) : \mu,\ e(1,2) : \eta\}$ is the EDB, where $\mu, \eta \in (0,1]$. Then, the bottom-up naive evaluation of P_1 on D proceeds as follows, as far as atom $p(1,2)$ is concerned. Let A denote $p(1,2)$, for convenience. Let also σ_i denote A's certainty obtained at iteration i of this evaluation, where $\sigma_0 = 0$. Assume the certainty associated with every EDB atom is known at iteration 0. Then, at iteration 1, we derive A with certainty $\sigma_1 = f_1^p(\beta,\eta)$, and derive $q(1,1)$ with certainty $\nu = f_0^p(\alpha,\mu)$. Note that $q(1,1)$ is the instance of $q(X,Z)$ in r_2 used in later iterations to derive A. At iteration 2, we apply r_1 and r_2 and derive A with certainties $f_1^p(\beta,\eta)$ (as before) and $min(\gamma, min(\nu,\sigma_1))$, respectively. There are two cases to consider.

Case 1: $\sigma_1 \geq min(\gamma,\nu)$. In this case, A's certainty derived by r_2 is $min(\gamma,\nu)$ which will continue to be derived by r_2 in the subsequent iterations. As this does not increase the overall certainty of A obtained, this evaluation would terminate in the next step.

Case 2: $\sigma_1 < min(\gamma,\nu)$. In this case, the overall certainty of A obtained at iteration 2 is $\sigma_2 = ind(\sigma_1,\sigma_1) = 1-(1-\sigma_1)^2$. It is straightforward to verify that $\sigma_i = 1-(1-\sigma_1)^i$, for all $i \geq 1$, whenever $\sigma_{i-1} < min(\gamma,\nu)$.

The question now is: given the certainties α, μ, γ, what is the least integer n such that $\sigma_n \geq min(\gamma,\nu)$, where $\nu = f_0^p(\alpha,\mu)$? The existence of this integer, if shown, would imply that this evaluation terminates at iteration $n+1$. In this example, as P_1 is recursive and the EDB D, viewed as a graph, is *cyclic*[2], we may conclude that whenever $\alpha < 1$, the fixpoint evaluation of P_1 on *every* EDB terminates in finite number of iterations, no matter what particular rule certainties are used in P_1 and

[2]An EDB D is *cyclic* w.r.t. a program P if there exists a ground atom A defined by P and D such that A has a disjunctive derivation tree which includes a goal node labeled A other than the root.

D. The particularity of these certainties determines the iteration step at which this evaluation terminates, which could be arbitrarily large, independent of the EDB size.

The iteration step n at which this evaluation terminates can be determined from the inequality $1-(1-\sigma_1)^n \geq min(\gamma,\nu)$. This yields:

$$n = \left\lceil \frac{\log(1-min(\gamma,\nu))}{\log(1-\sigma_1)} \right\rceil$$

assuming that this evaluation actually reaches to step n. We next show that n is a non-negative integer, and hence well defined. First note that $\log(x) < 0$, for all $x \in (0,1)$. Now, there are two cases to consider corresponding to the above two cases. (1) As shown in case 1 above, when $\sigma_1 = 1$ or $\sigma_1 \geq min(\gamma,\nu)$, the fixpoint evaluation of P_1 on D would terminate in one iteration, in which the formula above yields $n = 1$, as expected. (2) When $\sigma_1 < min(\gamma,\nu)$, which corresponds to case 2 above, we have $1-min(\gamma,\nu) > 1-\sigma_1$, which in turn implies that $n > 1$, since in this case, the numerator of the formula above for n would be larger than its denominator, and hence $\log(1-min(\gamma,\nu)) > \log(1-\sigma_1)$.

To show that n is well defined, we also need to show that the argument of the function log in the numerator of our formula above for n is not 0, or equivalently we have $min(\gamma,\nu) \neq 1$. This is true since otherwise, we would have $\gamma = 1$ and $\nu = 1$, which in turn would yield the contradiction $1 < 1$, as $1 = \nu = f_0^p(\alpha,\mu) \leq \alpha < 1$, since $\alpha < 1$, by our assumption, and $f_0^p(\alpha,\mu) \leq \alpha$, by Postulate 3 (the bounded-above property of propagation functions). We may thus conclude from our analysis above that *"for any instances of the rule certainties in P_1 and D, whenever $\alpha < 1$, we can always determine an integer n, using the above expression, such that the bottom-up naive evaluation of P_1 on D terminates at iteration $n+1$."*

Let us continue with more specific instances of this program to show how sensitive the termination behaviors of programs are w.r.t. the certainties used in the program. Consider again the p-program P_1 and suppose $\beta < 1$ and $\gamma = 1$. As before, assume that $\alpha < 1$. In then follows from our analysis that since $\alpha < 1$, the fixpoint evaluation of P_1 on every EDB terminates in a finite number of iterations.

Now consider an instance P_1' of P_1 in which $\alpha = 1$. As before, assume $\gamma = 1$ and $\beta < 1$. It is then easy to show that in this case, the fixpoint evaluation of P_1' will not terminate, for instance on $D' = \{e(1,1) : 1,\ e(1,2) : 1\}$. This is because we

would always have $\sigma_n < 1$, for all $n \geq 1$, upon noting that $ind(\alpha_1, \beta_1) < 1$, for all $\alpha_1, \beta_1 \in (0,1)$. That is, the certainty associated with $p(1,2)$ keeps increasing at every iteration and will never reach 1 (the top) unless in the limit.

On the other hand, if P_1' did not include the p-rule r_0, and hence q was, e.g., an EDB relation, we could easily determine an EDB D'' on which the fixpoint evaluation of $P_1' - \{r_0\}$ would not terminate. For instance, we could take $D'' = D' \cup \{q(1,1) : 1\}$. ■

Discussion. Let us summarize our observations so far. On one hand, we showed that whenever $\alpha < 1$, the bottom-up naive evaluation of P_1 on every EDB always terminates in a finite number of iterations. On the other hand, P_1 is using a disjunction function, *ind*, which is strictly monotone. That is, *ind* always returns a value which is larger than its argument values. This means we "expect" that the fixpoint evaluation of P_1' on some EDB will not terminate, unlike what we showed. How to deal with this dichotomy, which stems from the sensitivity of the fixpoint evaluation to rule certainties? The key point to note in the above example(s) is the role of the p-rule r_0 or more precisely, the role of the certainty α associated with this p-rule. If P_1 did not include r_0 or if $\alpha = 1$, then we could determine a desired EDB on which the fixpoint evaluation of the program would not terminate. However, when P_1 does include r_0 and $\alpha < 1$, the certainties of the EDB atoms passed to any instance of the $q(X, Z)$ in the body of r_2 would always be less than 1, which in turn would cause the fixpoint evaluation of P_1 on D to terminate in a finite number of iterations.

Our analysis above and the preceding examples show that how sensitive the termination behavior becomes w.r.t. rule certainties or the presence/absence of certain "innocuous" looking rule, e.g., r_0 in this example. This state of affairs complicates a general analysis of termination for a substantial class of programs. One way to distill the termination behavior of a p-program from the rule certainties is to look at a *strong termination property*, namely at the question of whether the program terminates on all EDBs, regardless of the certainties associated with rules.

Our idea for characterizing the termination property of p-programs is to ignore the specific rule certainties mentioned in a given p-program, and instead consider a p-program, obtained from the first one except possibly for rule certainties. This approach in a sense amounts to providing an approximate analysis of the termination

as opposed to an exact one. Our analysis results provides useful insights as to how combination functions affect the termination and data complexity of programs in the parametric framework.

Based on this idea, we will consider two "generic" p-programs, introduced in Examples 4.5.3 and 4.5.6, as *model programs* or *templates*, which are simple but general enough to study the problem. By a generic p-program we mean a program in which parameters are symbolic, rather than specific. We organize the study according to how the combination functions in instances of these templates compare with the meet and join operators in the underlying certainty lattice. For each instance, we consider the bottom-up naive evaluation of the program and provide a detailed analysis of its termination behavior. The results obtained in our analysis are then generalized to any p-program.

We remark that unlike in [KL88] in the context of the AB framework proposed by Kifer and Li, we are not using and/or suggesting the recurrence-based evaluations as a replacement for query evaluation. In this study, we use recurrence equations as an analysis aid to derive the certainty derived in a fixpoint computation.

In this chapter, we attempt to characterize the termination behaviors for a collection of IB frameworks. Rather than attempt this for individual frameworks, we do this in a "framework independent" manner, as we did in Chapter 2 to study the semantics of the parametric framework, and in Chapter 3 to study the containment of parametric conjunctive queries. We remark that our study in this chapter is in the context of the parametric framework, however, the analysis technique developed and the various results established are useful also in the context of AB frameworks.

The outline of this chapter is as follows. Next, we define some terms and concepts used in this chapter. In Section 4.4, we establish PTIME data complexity for large classes of p-programs, in which the disjunction function associated with every recursive predicate coincides with the lattice join. In the rest of the chapter, we consider other disjunction functions allowed in the parametric framework. Section 4.5.1 includes some more examples illustrating informally our approach and the kind of analysis we provide in this study. In Section 4.5, we consider p-programs in which some recursive predicates are associated with disjunction functions different than the join operator in the underlying certainty lattice. This is where we find p-programs

whose fixpoint evaluations may not terminate or may terminate in arbitrarily large, but finite, number of iterations, independent of the size of the EDBs. Concluding remarks appear in Section 4.6.

4.2 Preliminaries

We define the *complexity* of evaluating a program in the parametric framework as the number of iterations required for computing the least fixpoint of T_P, using the bottom-up naive evaluation method, as a function of the size of database [KL88, LS94a]. It is well known that the fixpoint evaluation of datalog programs always terminates in polynomial time (PTIME) in the number of constants specified in the database. The question is: "what can we say about termination and complexity of the bottom-up fixpoint evaluations of programs in datalog extended with uncertainty?" Kifer and Li [KL88] show that evaluating some programs in their AB framework does not terminate. Note that function symbols are not allowed in their framework, nor are they allowed in any program which we consider in this thesis. Lakshmanan and Sadri [LS94a] show that under certain conditions, programs in their IB framework enjoy PTIME data complexity. Recall that an EDB in our context is a set of annotated ground atoms, where annotations are elements of the underlying certainty domain $\mathcal{T}$. The *size* of an annotated EDB D is the number of distinct *constant symbols* occurring in D. That is, the size of an annotated EDB D is simply the size of D, with the certainties removed. Note that we do *not* consider the certainty values occurring in D as part of the input size.

We note that in datalog programs with uncertainty, the PTIME data complexity is preserved whenever (1) the program is not recursive, (2) the EDB is acyclic[3], or (3) the certainty domain is finite. When none of these conditions holds, we may still preserve the PTIME data complexity under certain restrictions, which will be characterized shortly. On the other hand, there are logic programs with uncertainty whose fixpoint evaluation on some EDB *will* not terminate, as shown in [KL88]. We identify a third class of programs with uncertainty whose fixpoint evaluations on some EDBs terminate in arbitrarily large but finite number of iterations, *independent* of

[3]We say an EDB D is *cyclic* if there is a recursive program P such that for some $A \in B_P$, there is a DDT of arbitrary height for A w.r.t. P and D. If no such program exists, we say D is *acyclic*.

the size of the EDB. To our knowledge, this latter class of programs has not been identified or discussed elsewhere.

When we talk about evaluating a p-program on a given EDB, we consider the bottom-up naive iterations of the fixpoint operator T_P introduced in Definition 2.3.5. This, however, does not rule out the possibility of using/developing other evaluation methods which perhaps could be more efficient.

Recall that the parametric framework includes a (potentially large) collection of IB frameworks as special cases. With this in mind and the various parameters allowed in each IB framework with wide range of complexities of the fixpoint evaluations of p-programs ranging from constant time to PTIME to arbitrary time and to non-terminating, the questions are: How to attack the problem while dealing with these varieties? What techniques can be used or need to be developed so that the conclusions drawn are useful and the results obtained are applicable to large classes of p-programs? In this chapter we attempt to address these questions.

4.3 Types of Disjunctions and Predicates

In this section, we define the notions of "types" of disjunction functions and "types" of predicates.

Depending on how a disjunction function in the parametric framework compares with the join operator in the underlying certainty lattice $\langle \mathcal{T}, \preceq, \otimes, \oplus \rangle$, we classify the family of disjunction functions in $\mathcal{F}_d$ into three types, defined as follows.

Definition 4.3.1 (Types of Disjunction Functions) *Let $f \in \mathcal{F}_d$ be a disjunction function in the parametric framework. Let $\mathcal{T}' = \mathcal{T} - \{\bot, \top\}$. Then, we say*

(a) f is of type 1 *provided, $f = \oplus$, i.e., f coincides with the lattice join.*

(b) f is of type 2 *provided, $\forall \alpha, \beta \in \mathcal{T}'$, $\oplus(\alpha, \beta) \prec f(\alpha, \beta) \prec \top$.*

(c) f is of type 3 *provided, $\exists \alpha, \beta \in \mathcal{T}'$, $\oplus(\alpha, \beta) \prec f(\alpha, \beta) \prec \top$, and $\exists \alpha', \beta' \in \mathcal{T}'$, $f(\alpha', \beta') = \top$.*

Note the distinction between types 2 and 3. For every certainty value different from the least and the greatest elements, the value returned by a type 2 disjunction function always keeps increasing when supplied with "better" argument values, while a type 3 function may return $\top$ for some such values. Also note that in the definition above, both α and β are required to be different from the bottom. This is because otherwise we would not be able to assert the relationship $\oplus \prec f$, noting that by our Postulate 3, $f(\bot, \bot) = \bot, \forall f \in \mathcal{F}_d$. Recall that for any pair of combination functions $f, g \in \mathcal{F}$, we defined $g \preceq f$, provided $g(\alpha, \beta) \preceq f(\alpha, \beta)$, $\forall \alpha, \beta \in \mathcal{T}$. The ordering $\prec$ on $\mathcal{F}$ is defined in the obvious way. A similar remark holds for requiring $\alpha \neq \top$ and $\beta \neq \top$, noting that by our Postulate 4, $f(\alpha, \top) = \top$, for all $f \in \mathcal{F}_d$ and for all $\alpha \in \mathcal{T}$.

Definition 4.3.2 (Types of Predicates) *Let P be a p-program, and p be a predicate defined in P. We say that p is of type i, for $i = 1, 2, 3$, provided the disjunction function associated with p is of type i.*

For ease of presentation and clarity, we define the following terms. By *evaluation of a p-program P* on an EDB D, we mean computing the fixpoint semantics of $P \cup D$ using the *bottom-up naive evaluation method.* When the semi-naive method is meant, we will make this explicit. We say *an evaluation terminates* if it terminates in a finite number of iterations. Abusing the terminology, we may sometimes say a program terminates to mean its evaluation terminates. We say *an evaluation does not terminate* if it does not terminate in finite time or, equivalently, it terminates only at ω. It follows from the continuity of the fixpoint operator, T_P, established in Lemma 2.3.6 that every program terminates at most at ω. We say *an evaluation terminates in arbitrary number of iterations* if it terminates in a number of iterations which is independent of the size of the EDB. In this case, the exact iteration number at which the evaluation terminates is determined by the *particular rule certainties* involved and the *combination functions* used. We will use "termination in finite time" and "termination in an arbitrarily large number of iterations" interchangeably. Clearly, termination in finite time includes termination in PTIME, however, we will make it explicit whenever we mean termination in PTIME. That is, unless explicitly mentioned, termination in finite time does not necessarily imply termination in PTIME.

4.4 PTIME Data Complexity

In this section, we establish PTIME data complexity for large classes of p-programs. Our first result deals with p-programs whose underlying certainty domains are finite.

Proposition 4.4.1 *Let P be any p-program whose underlying certainty domain $\mathcal{T}$ is finite. Then the bottom-up naive evaluation of P on every EDB D always terminates in PTIME, in the size of D and $\mathcal{T}$.*

Proof. Let n be the number of distinct constants occurring in D. The size of the Herbrand base is equal to the number of possible ground atoms and this is $\leq |\mathcal{T}|n^k$, where $|\mathcal{T}|$ denotes the size of $\mathcal{T}$, and k is the maximum of the arities of the predicates in P. Then the number of iterations required to obtain the least fixpoint of T_P on the input $P \cup D$ is at most $|\mathcal{T}|n^k$. This is because each iteration of the bottom-up evaluation of P on D produces at least one fact with a "better" certainty than before, or terminates. Note that initially every atom is associated with the least certainty $\bot \in \mathcal{T}$. Therefore, the total number of possible iterations before termination is $O(|\mathcal{T}|n^k)$, which was to be shown. ∎

Note that according to this result, combination functions do not play any role in the termination of p-programs which use a finite domain as the underlying certainty lattice. Also in our proof, we were not concerned with the amount of work done at every iteration. We just remark that in addition to the join and union operations performed at every iteration in the classical case which can be done in PTIME in the size of the database, we also perform some certainty computations using the combination functions, which we do not consider in our complexity analysis and results.

We also remark that when the size n of D is much larger than the size of $\mathcal{T}$, we may ignore $|\mathcal{T}|$ and say the complexity of the fixpoint evaluation of P on D is $O(n^k)$.

This proposition is applicable to datalog, which uses $\mathcal{T} = \{0, 1\}$ as the underlying certainty domain, and establishes once again the well-known PTIME data complexity for evaluating programs in datalog. It is also applicable to the deductive IST framework of Lakshmanan and Sadri [LS94b], which uses $\mathcal{T} = L^\ell$ as the underlying certainty domain, where $L = \{\bot, -1, 1, \top\}$ and ℓ is the number of sources which

contribute to the information in the database. As remarked in the above, the PTIME data complexity in this case is preserved when ℓ is fixed. When ℓ is much less than the size of D, which is normally the case, ℓ can be regarded as a constant. When these assumptions are not valid, the size of the Herbrand base is exponential in ℓ. Therefore, the bottom-up fixpoint evaluation of programs in this framework can be done in PTIME in the size of the database D but exponential in ℓ. More precisely, we have shown that this can be achieved in $O(4^{\ell}n^{k})$, where k is the maximum of the arities of the predicates in P and n is the number of elements in D [Shi93]. Note that this in itself does *not* exclude the possibility of a more efficient algorithm for computing the least fixpoint of programs in [LS94b]. This remark is valid in general for our results in this chapter. Lakshmanan [Lak94] proposed a framework which uses $\mathcal{T} = 2^{T^M} \times 2^{T^N}$ as the certainty domain, where $T = \{1, -1, \perp\}$, and M and N are positive integers. The Herbrand base of programs in this framework is exponential in the size of $max(M, N)$, which can be thought as the number ℓ of sources in the framework proposed in [LS94b]. By our proposition above, the fixpoint evaluation of programs in [Lak94] can be done in PTIME in the size of the database but exponential in $max(M, N)$.

In the rest of this chapter, we assume the underlying certainty domain is infinite. Since in some cases, we study data complexity of evaluating a program as a function of the EDB size, we assume, without loss of generality, that the EDB is not part of the user program, and that constant symbols are not used in the program. Furthermore, we assume that every combination function can be computed with arbitrary precision. The strictness property of the propagation and conjunction functions defined as Postulate 11 in Subsection 2.1.1 is important for our conclusions. This postulate is similar, in spirit, to the "strictness" considered by Fagin [Fag96] in establishing bounds on the complexity of conjunctive query evaluation in fuzzy information systems. Intuitively, it says that conjunction of a bunch of certainties is "non-zero" iff each one of them is non-zero. The dual of this postulate, called *dual strictness* in the same paper, says that disjunction of a bunch of certainties is less than "1" iff each one of them is less than "1". This latter property is built into our definition of type 2 disjunction functions. There is a compelling intuition associated with these postulates.

Our next result establishes PTIME data complexity for a large class of p-programs whose underlying certainty domains are infinite. It ensures that when the disjunction function associated with every recursive predicate defined in a p-program coincides with the lattice join, then the fixpoint evaluation of the p-program on every EDB *always* terminates in PTIME, in the size of the EDB. Note that in this case, it is not important what propagation and conjunction functions are used in the p-program, or what disjunction functions are associated with the non-recursive predicates defined.

Theorem 4.4.2 *Let P be any p-program in which every recursive predicate defined, if any, is of type 1. Then the bottom-up naive evaluation of P on every EDB D always terminates in PTIME, in the size of D.*

Proof. The proof uses disjunctive derivation trees (DDTs) introduced in Definition 2.3.8, and is based on a generalization of the proof of PTIME data complexity for programs in [LS94a]. We will show that non-simple DDTs need not be considered in deriving the certainty of any atom. The result then follows upon noting that the height of simple DDTs is bounded by n^k, where n is the number of constants in D and k is the maximum of the arities of the predicates in P.

Let $A \in B_P$ be any atom, and T be any DDT of height h for A w.r.t. P and D. We know that if there is a DDT of height h for A which computes a certainty α for its root, then $A : \alpha'$ will be derived by the bottom-up evaluation at iteration h where $\alpha \preceq \alpha'$. Now, the reason why non-simple DDTs need not be considered is as follows. When the disjunction function is of type 1, we can show that for every DDT T that computes a certainty α for A, there is a simple DDT, say T' for A, of height no more than that of T, such that it computes the same certainty for A. Suppose $h > n^k$. Since there are only n^k distinct ground atoms in the Herbrand base of $P \cup D$, then there should be a path in T from the root to a leaf node of height more than n^k such that there is a goal node G occurring at least twice on the path, and its first occurrence is at a height less that n^k. This would then imply the predicate $\pi(G)$ of G is recursive, which by our assumption is of type 1. Let G_1 be the first occurrence of G from the root and G_2 be the second. That is, G_1 occurs at a height less than n^k, and G_2 occurs as a (immediate or otherwise) subtree T' of G_1. It follows from the procedure for assigning certainties to the nodes in a DDT (given in Definition 2.3.9) that $\alpha_1 \preceq \alpha_2$, where α_i is the certainty associated with G_i, $1 \leq i \leq 2$. This is because

in this case, α_2 contributes to the value derived for the root of T' and this root is a subgoal in the body of the rule node defining G_1. Since the conjunction and propagation functions are bounded above (Postulate 3), and the disjunction function associated with $\pi(G)$ coincides with the lattice join, the certainty α_1 cannot be more than α_2, w.r.t. $\preceq$. This implies that the subtree T' of T at G_1 can be replaced with the subtree at G_2 without changing the certainty derived for the root A of T. This replacement procedure can be applied recursively to every DDT of height more than n^k, eventually turning it into a DDT of height $\leq n^k$, while preserving the certainty derived at the root. The result then follows noting that G was an arbitrary goal node and T an arbitrary DDT. ■

Simple DDTs were first introduced by Lakshmanan and Sadri [LS94a] as a generalization of simple paths in graphs, and used to establish PTIME data complexity of a certain class of programs in their framework, namely those in which the disjunction function associated with the recursive predicates is the *positive correlation*, *pc*, which is also used as the join operator in their certainty lattice. Our result generalizes their's, since in addition to [LS94a], it is also applicable to van Emden's language [van86] and the possibilistic framework of Dubois et al. [DLP91], for instance.

4.5 Arbitrary Disjunctions

In this section, we consider the case of arbitrary disjunction functions in our study of the termination property of p-programs. Before we get into formal discussion, we consider a number of examples illustrating the role of combination functions in this study. These examples includes p-programs defining recursive predicates of type 2 and/or type 3. In most, but not all cases, we draw conclusions of the following form. Given a p-program P, there are p-programs Q_1 and Q_2, identical to P, except possibly for rule certainties, and there are EDBs D_1 and D_2 such that the bottom-up naive evaluation of Q_1 on D_1 terminates in finite number of iterations, and that of Q_2 on D_2 does *not* terminate. Since there are programs whose bottom-up fixpoint evaluations do not terminate or terminate in arbitrarily large number of iterations, we may not expect to be able to provide a complexity analysis of the fixpoint algorithm. In fact, it may not even be meaningful to discuss about the data complexity in such cases.

Also note that when we make a conclusion of the form "there are Q and D such that $\cdots$", we assume that not all rule certainties in Q and D are $\top$, since otherwise we would be dealing with standard datalog programs, which can be evaluated in PTIME in the size of the database.

4.5.1 More Examples

Let us consider again the p-program P_1 of Example 4.1.1, for which we showed that when $\alpha < 1$, the fixpoint evaluation of the program on every EDB would always terminate in finite time. In this example, if we change the conjunction and/or propagation functions, the resulting p-program may have a different termination behavior, depending on the specific rule certainties involved. For instance, it can be shown that when the conjunction function associated with r_2 is changed to multiplication $*$, we can determine instances of rule certainties in P_1 and the EDB D, such that P_1's evaluation on D does not terminate. (Note that $* < min$, on certainties different from 0 and 1.) For example, let $\eta = 0.5$, and α, β, γ, and η be 1. It is then straightforward to verify that the bottom-up evaluation of $P_1 \cup D$ does not terminate.

Next, we illustrate that when in a p-program P, a type 2 disjunction function is associated with some recursive predicate, then there are p-programs Q_1 and Q_2, which are identical to P except possibly for the rule certainties, and there are EDBs D_1 and D_2, such that Q_1's evaluation on D_1 terminates in a finite number of iterations, and Q_2's evaluation on D_2 does not terminate. Note that Q_1 and Q_2 could be identical. Since termination in arbitrary time includes PTIME as a particular case, we will not consider it as a separate case.

Example 4.5.1 Let P_2 be the p-program below.

$$r_1: \; p(X,Y) \xleftarrow{\alpha} e(X,Y); \qquad \langle ind, min, \_ \rangle.$$
$$r_2: \; p(X,Y) \xleftarrow{\beta} e(X,Z),\ p(Z,Y); \qquad \langle ind, min, * \rangle.$$

Suppose $\alpha = 1$, and β is any certainty in $(0, 0.3]$. Then it can be verified that the fixpoint evaluation of P_2 on $D_1 = \{e(1,1) : 0.5,\ e(1,2) : 0.6\}$ terminates in 3 iterations. On the other hand, when $\alpha = \beta = 1$, it can be verified that the fixpoint evaluation of P_2 on D_1 will not terminate. As the third case, we can determine α and β such that the fixpoint evaluation of P_2 on D_1 terminates in a finite number of

iterations. To see this, let $\alpha = 1$. We then have the recurrence equations below in which σ_i denotes the certainty associated with $p(1,2)$ at iteration i.

$$\begin{aligned} \sigma_1 &= 0.6 \\ \sigma_{n+1} &= 0.5 * \sigma_n + 0.6 - 0.3 * \sigma_n, \qquad n \geq 1, \end{aligned}$$

For atom $p(1,1)$, we have the following recurrence equations.

$$\begin{aligned} \gamma_1 &= 0.5 \\ \gamma_{n+1} &= 0.5 * \gamma_n + 0.5 - 0.25 * \gamma_n, \qquad n \geq 1, \end{aligned}$$

Choosing $\beta = 0.5 * min(\sigma_i, \gamma_i)$ would ensure that when both of σ_i and γ_i "cross" this value, subsequent iterations won't change the certainties of $p(1,1)$ and $p(1,2)$. Note that $0.6 \leq \sigma_i \leq 0.75$ and $0.5 \leq \gamma_i \leq 2/3$. Therefore, $0.5 * 0.5 \leq \beta \leq 0.5 * 2/3$, where 0.5 in the lower limit of β is the minimum of the lower limits of σ_i and γ_i, and $2/3$ in the upper limit of β is the minimum of the upper limits of σ_i and γ_i. Equivalently, $0.25 \leq \beta \leq 1/3$. The certainly associated with $p(1,1)$ and $p(1,2)$ in the fixpoint would be $2/3$ and $22/30$, respectively, and this happens when $\beta = 1/3$, in which case these certainties are attained at ω. ■

Now let P_3 be the same as the p-program P_2 with $\alpha = \beta = 1$, and with the propagation and conjunction functions in r_2 switched. Then, as in the previous example, we can determine EDBs D_1, D_2 and D_3 such that P_3's evaluation on D_1 terminates in a few number of iterations. P_3's evaluation on D_2 terminates in arbitrarily large number of iterations, and finally P_3's evaluation on D_3 will not terminate. For instance, take $D_1 = \{\epsilon(1,1) : 0.5,\ \epsilon(1,2) : 0.6\}$, $D_2 = \{\epsilon(1,1) : \nu,\ \epsilon(1,2) : 0.6\}$, where $0 < \nu = \sigma_i < 1$, and σ_i is the i-th term defined by the corresponding recurrence equation for the atom $p(1,2)$. As D_3 we can take $\{e(1,1) : 1,\ e(1,2) : 0.6\}$. This example shows that, again, switching the propagating and conjunction functions in P_2 does not change its termination property, in the sense that for both P_2 and P_3, we could determine Qs and Ds such that Q_2's evaluation on D_2 terminates in finite time and Q_3's evaluation on D_3 will not terminate. This is a pattern we found suitable for formulating the termination property presented in Subsection 4.5.

As another interesting example, let P_4 be the same as P_2 of Example 4.5.1, with $\alpha = 0.5$, $\beta = 1$, and $\langle ind, min, min \rangle$ as the triple of the combination functions associated with r_2. It can be easily shown that the fixpoint evaluation of P_4 on

$\{e(1,1) : 1\}$ will not terminate. We will return to P_4 shortly.

The following example illustrates that evaluating a p-program defining a recursive predicate of type 3 may exhibit different behavior than those defining recursive predicates of type 2. An example of a type 3 disjunction function is $nc(\alpha, \beta) = min(1, \alpha + \beta)$, denoting the *negative correlation*, in the probabilistic sense.[4]

Example 4.5.2 Let P_5 be the following p-program.

$$r_1: \quad p(X,Y) \xleftarrow{0.5} e(X,Y); \qquad \langle nc, min, \_ \rangle.$$
$$r_2: \quad p(X,Y) \xleftarrow{1} e(X,Z),\ p(Z,Y); \qquad \langle nc, min, min \rangle.$$

Note that this is the same as P_4 except that here, the disjunction function associated with the predicate p is nc. Let $D_1 = \{e(1,1) : 1,\ e(1,2) : 0.2\}$ be an EDB. Then, the bottom-up evaluation of P_5 proceeds as follows, as far as atom $p(1,2)$ is concerned. In step 2, we derive $p(1,1) : 0.5$ and $p(1,2) : 0.2$, and in step 3 we get $p(1,1) : 0.5$ and $p(1,2) : 0.2$. Using nc, the corresponding atoms from these two iterations are then combined which yields $p(1,1) : 1$ and $p(1,2) : 0.4$. This evaluation goes on to step 6, at which we obtain $p(1,2) : 1$, and terminates at step 7, at which we obtain no "new" fact. Given $D_2 = \{e(1,1) : 1,\ e(1,2) : 0.02\}$ as the EDB, it can be verified that evaluation of P_5 on D_2 terminates in 51 steps. Note that P_4 and P_5 differ only on the disjunction function associated with the predicate p. Also note that the fixpoint evaluation of P_4 on the EDB $\{e(1,1) : 1\}$ will not terminate, while it can be shown that evaluating P_5's on every EDB terminates, no matter what certainties are associated with the p-rules. ■

Consider the p-program P_5 again. If every occurrence of min in P_5 is replaced by $*$, then unlike in the previous case, the fixpoint evaluation of the resulting program will not terminate on some EDB, e.g., $\{e(1,1) : 0.2\}$. It can be verified that the certainties of $p(1,1)$ obtained at the sequence of iterations $1, 2, \ldots$ are $0.2,\ 0.22, \ldots$, respectively, where the i-th value, $0.22\ldots2$, has i occurrences of 2. Note that the role of $*$ is crucial in this example. Also note that we would get a non-terminating p-program, on some EDB, if $\langle nc, min, * \rangle$ is the triple associated with r_2. Our analysis

[4] nc is the exact opposite of the positive correlation. It means the occurrences of the events overlap minimally. If the probabilities of the events in this mode sum to less than 1, they do not (have to) overlap at all. Otherwise, they overlap to the extent by which the sum exceeds 1 [LS94a]

in the following section provides insights of the termination property and explains the computational behavior observed in these examples.

We are now ready to provide a formal analysis of the termination property of p-programs defining recursive predicates of type 2 and/or type 3. Depending on the types of disjunction functions associated with the recursive predicates in a given p-program, we consider four classes of generic p-programs, as follows. The first class, studied in Subsection 4.4, includes those p-programs in which every recursive predicate is of type 1. Subsection 4.5.2 considers p-programs which define at least one recursive predicate of type 2 and no recursive predicate of type 3. The third class, studied in Subsection 4.5.3, includes the p-programs which define at least one recursive predicate of type 3 and no recursive predicate of type 2. In Subsection 4.5.4, we consider the class of p-programs defining at least two recursive predicates, one of type 2 and the other of type 3. In addition to the particular types of predicates defined in each of the cases mentioned here, a p-program considered may also define (recursive) predicates of type 1 and non-recursive predicates of any types.

To study the role of combination functions in the termination property, we introduce two "generic" p-programs, PM_1 and PM_2, as program templates, which essentially compute the transitive closures of some EDB relations. Although the underlying certainty lattice employed in these templates is $\langle [0,1], \leq, min, max \rangle$ and the combination functions considered are defined over $[0,1]$, our results are not peculiar to these particular choices of parameters. This is mainly due to the fact that the results are based on the properties of the parameters, as opposed to specific ones. This makes the results applicable to large classes of p-programs. The template PM_1, defined below, is used for the first three classes of p-programs mentioned above, and PM_2, defined in Example 4.5.6, is used for the fourth class.

Example 4.5.3 (Program Template 1) Let PM_1 be the following p-program:

$$r_1: \quad p(X,Y) \xleftarrow{\alpha} e(X,Y); \qquad \langle f^d, f_1^p, - \rangle.$$
$$r_2: \quad p(X,Y) \xleftarrow{\beta} e(X,Z),\ p(Z,Y); \qquad \langle f^d, f_2^p, f_2^c \rangle.$$

where $\alpha, \beta \in (0,1]$ are rule certainties f^d is a disjunction function in $\mathcal{F}_d$ associated with the predicate p, $f_i^p \in \mathcal{F}_p$ is the propagation function associated with r_i, for $i = 1,2$, and $f_2^c \in \mathcal{F}_c$ is the conjunction function associated with r_2. The EDB

considered in the following section is $D = \{e(1,1) : \mu,\ e(1,2) : \eta\}$, where $\mu, \eta \in (0,1]$.
■

4.5.2 Disjunction Functions of Type 2

In the following four subsections, we consider instances of PM_1 with $f^d = ind$, where $ind(\alpha, \beta) = \alpha + \beta - \alpha\beta$ is a disjunction function of type 2. Depending on the pair of propagation and conjunction functions associated with the recursive rule r_2, the four cases considered are: $\langle min, min\rangle$, $\langle *, min\rangle$, $\langle min, *\rangle$, and $\langle *, *\rangle$. Note that min is the meet operator in the certainty lattice and $* < min$, i.e., for all certainties α, β different from 0 and 1, we have $\alpha * \beta < min(\alpha, \beta)$. One may ask: why we only consider these pairs of functions? We remark that any function f such that $f \prec \otimes$ (on all certainties different that $\bot$ and $\top$) could be chosen for our purposes; $*$ chosen is just one such function. We take $\mathcal{T} = [0,1]$ as the underlying certainty lattice for our examples in this chapter.

4.5.2.1 Lattice-Theoretic Propagation and Conjunction

Let P_6 be an instance of PM_1 with $f^d = ind$ and $f_2^p = f_2^c = min$. This case was essentially discussed in Example 4.1.1. A summary of the results adapted to this case are as follows. In the naive evaluation of P_6 on the EDB D above, the certainty σ_n of $p(1,2)$ obtained at iteration $n \geq 1$ is $\sigma_n = 1 - (1 - \sigma_1)^n$, whenever $\sigma_n < min(\beta, \mu)$, where $\sigma_1 = f_1^p(\alpha, \eta)$ is the certainty of $p(1,2)$ derived by r_1 at iteration 1. Clearly, this evaluation terminates at iteration $n+1$, whenever $\sigma_n \geq min(\beta, \mu)$, for any $n \geq 1$.

4.5.2.2 Lattice-Theoretic Conjunction but Arbitrary Propagation

Consider the instance P_7 of PM_1 in which $f^d = ind$, $f_2^p = *$ and $f_2^c = min$. Then, the bottom-up naive evaluation of P_7 on D proceeds as follows, as far as atom $p(1,2)$ is concerned. Let us call $p(1,2)$ as A, for convenience, and let σ_n denote A's certainty at iteration n. Then, $\sigma_1 = f_1^p(\alpha, \eta)$, obtained by applying r_1. At iteration 2, we have two derivations of A, one by r_1 with certainty σ_1, and the other by r_2 with certainty $\sigma_1\beta$. Therefore, $\sigma_2 = ind(\sigma_1, \sigma_1\beta) = \sigma_1 + \sigma_1\beta(1 - \sigma_1)$. At iteration 3, we can apply again r_1 and r_2 to derive A, and thus $\sigma_3 = ind(\sigma_1, \sigma_2\beta) = \sigma_1 + \sigma_1\beta(1-\sigma_1) + \sigma_1\beta^2(1-\sigma_1)^2$.

It can be verified that A's certainty at iteration $n \geq 1$ is:

$$\sigma_n = \sigma_1 \sum_{i=0}^{n-1} \beta^i (1-\sigma_1)^i = \frac{\sigma_1[1-\beta^n(1-\sigma_1)^n]}{1-\beta(1-\sigma_1)}.$$

This evaluation terminates at iteration $n+1$ if $\sigma_n \geq \mu$, for any integer $n \geq 1$. This is because in this case, A's certainty derived by r_2 would be $\beta\mu$, which will continue to be derived at every following iteration. If at every iteration n, $\sigma_n < \mu$, then the fixpoint evaluation of P_7 on D terminates only at ω, at which the certainty obtained for A is:

$$\sigma = \lim_{n \to \omega} \sigma_n = \frac{\sigma_1}{1-\beta(1-\sigma_1)}.$$

It is clear from the above formula for σ that when $\mu \geq \sigma_1/(1-\beta+\sigma_1\beta)$, this evaluation terminates only at ω. Note that the certainty σ of A in the limit could also be determined by solving the the recurrence equation $\sigma_{n+1} = ind(\sigma_1, \sigma_n\beta) = \sigma_1 + \sigma_n\beta - \sigma_1\sigma_n\beta$, upon noting that in the limit, $\sigma_{n+1} = \sigma_n = \sigma$.

We remark that in this case, for any choice of the rule certainties in P_7, one can always find an EDB D on which the fixpoint evaluation of P_7 terminates in a finite number of iterations. This follows from the termination condition $\sigma_n \geq \mu$, which depends on the certainty μ coming from the EDB, or to be more precise, the certainty μ is associated with an instance of the EDB atom used in the body of r_2. Therefore, given any specific instances of the rule certainties α, β in P_7, we can always determine a desired EDB D on which the evaluation of P_7 terminates in finite steps. However, in an arbitrary p-programs with type 2 disjunction functions, μ need not directly come from the EDB. In general, we can view μ as an abstraction of the certainty of the conjunction of all, but the recursive, subgoals in the rule body. In this case, the certainty of this conjunction of subgoals is determined (directly) by the combination functions employed in P_7 and the specific rule certainties used.

4.5.2.3 Lattice-Theoretic Propagation but Arbitrary Conjunction

Let P_8 be an instance of PM_1 with $f^d = ind$, $f_2^p = min$, and $f_2^c = *$. Then, the bottom-up naive evaluation of P_8 on D proceeds as follows, as far as atom $A = p(1,2)$ is concerned. At iteration 1, rule r_1 defines A with certainty $\sigma_1 = f_1^p(\alpha, \eta)$. At iteration 2, there are two derivations of A by r_1 and r_2 with certainties σ_1 and $\sigma_1\mu$, respectively, which when combined, using ind, yields $\sigma_2 = ind(\sigma_1, \sigma_1\mu) = \sigma_1 +$

$\sigma_1\mu(1-\sigma_1)$. At iteration 3, we apply r_1 and r_2 again and derive A with certainties σ_1 and $\sigma_2\mu$, and thus the overall certainty of A at iteration 3 is $\sigma_3 = ind(\sigma_1, \sigma_2\mu) = \sigma_1 + \sigma_1\mu(1-\sigma_1) + \sigma_1\mu^2(1-\sigma_1)^2$. It can be verified that A's certainty at iteration $n \geq 1$ is:

$$\sigma_n = \sigma_1 \sum_{i=0}^{n-1} \mu^i (1-\sigma_1)^i = \frac{\sigma_1[1-\mu^n(1-\sigma_1)^n]}{1-\mu(1-\sigma_1)}.$$

This evaluation terminates at iteration $n+1$ if $\sigma_n\mu \geq \beta$, for any integer $n \geq 1$. This is because at iteration n, A's certainty derived by r_2 is $\sigma_n\mu$, which will continue to be derived in the future. When $\beta \geq \sigma_1\mu/(1-\mu+\sigma_1\mu)$, this evaluation terminates only at ω, at which we obtain $\sigma_1/(1-\mu+\sigma_1\mu)$ as A's certainty, derived by computing the limit of σ_n as n reaches ω. Note that σ could be also determined from the corresponding recurrence equation $\sigma_{n+1} = ind(\sigma_1, \sigma_n\mu) = \sigma_1 + \sigma_n\mu - \sigma_1\sigma_n\mu$, noting that in the limit, $\sigma_{n+1} = \sigma_n = \sigma$. Also note that when $\beta = 1$, the certainty associated with A in the fixpoint is 1, whenever this evaluation does not terminate, and this happens exactly when $\mu = 1$ and $\sigma_1 < 1$.

4.5.2.4 Arbitrary Propagation and Conjunction

Consider the instance P_9 of PM_1 in which $f^d = ind$ and $f_2^p = f_2^c = *$. That is, the propagation and conjunction functions associated with r_2 are both less than the lattice meet, w.r.t. the ordering $<$. Then, the bottom-up naive evaluation of $P_9 \cup D$ proceeds as follows, as far as $p(1,2)$ is concerned. Let $A = p(1,2)$. Applying r_1 at iteration 1, we derive A with certainty $\sigma_1 = f_1^p(\alpha, \eta)$. At iteration 2, r_1 and r_2 define A with certainties σ_1 and $\sigma_1\beta\mu$, respectively, and hence $\sigma_2 = ind(\sigma_1, \sigma_1\beta\mu) = \sigma_1 + \sigma_1\beta\mu(1-\sigma_1)$. At iteration 3, we apply r_1 and r_2 again and derive A with certainties σ_1 and $\sigma_2\beta\mu$. The overall certainty of A at this iteration is thus $\sigma_3 = ind(\sigma_1, \sigma_2\beta\mu) = \sigma_1 + \sigma_1\beta\mu(1-\sigma_1) + \sigma_1\beta^2\mu^2(1-\sigma_1)^2$. It can be verified that A's certainty at iteration $n \geq 1$ would be:

$$\sigma_n = \sigma_1 \sum_{i=0}^{n-1} \beta^i \mu^i (1-\sigma_1)^i = \frac{\sigma_1[1-\beta^n\mu^n(1-\sigma_1)^n]}{1-\beta\mu(1-\sigma_1)}.$$

Let σ be the certainty of A obtained at ω. Thus,

$$\sigma = \lim_{n \to \omega} \sigma_n = \frac{\sigma_1}{1-\beta\mu(1-\sigma_1)}.$$

Subsection	f_2^p & f_2^c	σ_n	Terminates at iteration $n+1$ if	$\sigma = \lim_{n\to\omega} \sigma_n$
4.5.2.1	$f_2^p = f_2^c = min$	$1-(1-\sigma_1)^n$	$\sigma_n \geq min(\beta, \mu)$	1
4.5.2.2	$f_2^p = * \;\&\; f_2^c = min$	$\frac{\sigma_1[1-\beta^n(1-\sigma_1)^n]}{1-\beta(1-\sigma_1)}$	$\sigma_n \geq \mu$	$\frac{\sigma_1}{1-\beta(1-\sigma_1)}$
4.5.2.3	$f_2^p = min \;\&\; f_2^c = *$	$\frac{\sigma_1[1-\mu^n(1-\sigma_1)^n]}{1-\mu(1-\sigma_1)}$	$\sigma_n\mu \geq \beta$	$\frac{\sigma_1}{1-\mu(1-\sigma_1)}$
4.5.2.4	$f_2^p = f_2^c = *$	$\frac{\sigma_1[1-\beta^n\mu^n(1-\sigma_1)^n]}{1-\beta\mu(1-\sigma_1)}$	$\sigma_n = 1$	$\frac{\sigma_1}{1-\beta\mu(1-\sigma_1)}$

Table 2: Termination property of PM_1 when $f^d = ind$

We could also determine σ by solving the recurrence equation $\sigma_{n+1} = ind(\sigma_1, \sigma_n\beta\mu)$, noting that in the limit, $\sigma_{n+1} = \sigma_n = \sigma$.

Note that this evaluation terminates at iteration $n+1$ whenever $\sigma_n = 1$, for some integer n. In this case, $\sigma = 1$, which happens only if $\sigma_1 = 1$ or $\beta\mu = 1$. When both σ_1 and $\beta\mu$ are less than 1, it can be shown that $\sigma < 1$. To see this, first note that both σ_1 and $\beta\mu$ are non-zero. Then, since $\beta\eta < 1$, we can multiply both sides of this inequality by $1-\sigma_1$, which yields $\beta\mu(1-\sigma_1) < 1-\sigma_1$. This in turn implies that $\sigma_1 < 1-\beta\mu(1-\sigma_1)$, i.e., the numerator of the fraction above for σ is less than its denominator, and hence $\sigma < 1$.

Table 2 summarizes our termination results for various instances of PM_1 with $f^d = ind$. The table shows that when $\beta = 1$, the choice of the propagation function f_2^p is not crucial in characterizing termination, since in this case, the first two rows under the column σ_n would be identical. The same remark holds for the values under the column σ. These observations of course should not be surprising as they follow from Postulate 10, which asserts $f(\alpha, \top) = \alpha$, for every propagation and conjunction function f and for every certainty $\alpha \in \mathcal{T}$. For the same reason, it can be seen from the table that when $\mu = 1$, i.e. when the certainty of the instance $e(1,1)$ of $e(X,Z)$ in the body of r_2 is $\top$, then the choice of the conjunction function f_2^c is not crucial for termination. We may thus conclude from the above that for the instance P_1 of the template PM_1, if the certainty associated with the recursive rule r_2 defining a type 2 predicate is 1 and also when the certainties of every subgoal in r_2 other than the instance A of the p-atom defined by r_2 is 1, then no matter what propagation

and conjunction functions are associated with r_2, the certainty of A at iteration n is $1-(1-\sigma_1)^n$, and at ω is 1.

In all the cases studied in this section using the instances of PM_1, we observed that one should expect evaluation which may require arbitrarily (large) number of iterations to terminate or expect non-termination. Our observations in the preceding subsections on the termination property of p-programs defining recursive predicates of type 2 can be generalized as follows.

Conjecture 4.5.4 *Let P be a p-program with $\langle \mathcal{T}, \preceq, \otimes, \oplus \rangle$ as the underlying certainty lattice. Suppose all the recursive predicates in P are of types 1 or 2, and at least one of them is of type 2. Let $r_1, \ldots, r_k$ be the recursive p-rules in P defining type 2 predicates.*

(a) If for every r_i, $1 \leq i \leq k$, the propagation and conjunction functions associated with r_i are $\otimes$, and the certainty associated with r_i is $\prec \top$, then the bottom-up naive evaluation of P on every *EDB* always *terminates in finite time.*

(b) Suppose for some r_i, $1 \leq i \leq k$, either (1) $\otimes$ is the associated propagation and conjunction functions and $\top$ is the certainty of r_i, or (2) at least one of the propagation and conjunction functions associated with r_2 is $\prec \otimes$. Then there are p-programs Q_1 and Q_2 identical to P, except possibly for rule certainties, and there are EDBs D_1 and D_2 such that (i) Q_1's evaluation on D_1 terminates in finite time, and (ii) Q_2's evaluation on D_2 will not terminate.

The idea of a proof of this conjecture is to generalize our observations in the detailed analysis performed above. The key point to note is that in an arbitrary p-program, we may have in the body of recursive p-rules, any subgoal B other than the recursive one, while we had an EDB fact, $e(X, Z)$, in the body of r_2 in PM_1. The difference is that in the former, B's certainty may also increase at every iteration, while in PM_1 it stays put, i.e., the certainty of $e(1,1)$ is fixed and is μ.

4.5.3 Disjunction Functions of Type 3

In this section, we consider again the program template PM_1, defined in Example 4.5.3. In all the instances of PM_1 studied in this section, we use $f^d = nc$ as the

disjunction function, which is of type 3 and is defined as $nc(\alpha, \beta) = min(1, \alpha + \beta)$. As in the previous section, we will consider four cases, depending on how the pair of propagation and conjunction functions associated with the recursive rule r_2 compares with the meet operator in the certainty lattice. Also, as before, the EDB considered in this section is $D = \{e(1,1) : \mu,\ e(1,2) : \eta\}$.

4.5.3.1 Lattice-Theoretic Propagation and Conjunction

Let P_{10} be an instance of PM_1 in which $f^d = nc$ and $f_2^p = f_2^c = min$. Then, the bottom-up naive evaluation of P_2 on D proceeds as follows, as far as $p(1,2)$ is concerned. Let $A = p(1,2)$. At iteration 1, r_1 defines A with certainty $\sigma_1 = f_1^p(\alpha, \eta)$. At iteration 2, there are two derivations of A, one by r_1 with certainty σ_1 and the other by r_2 with certainty $min(\beta, min(\mu, \sigma_1))$. These certainties should be combined, using nc, which gives $\sigma_2 = 2\sigma_1$, assuming that $\sigma_1 \leq min(\beta, \mu)$. Clearly, for any $n \geq 1$, whenever $\sigma_n > min(\beta, \mu)$, this evaluation terminates at iteration $n + 1$. This assumption thus allows the evaluation proceed to the next iterations, if possible. At iteration 3, the certainty of A is $\sigma_3 = nc(\sigma_1, \sigma_2) = 3\sigma_1$. It is straightforward to verify that A's certainty at iteration n is $\sigma_n = n\sigma_1$. From this, we may conclude that in this case of P_{10}, the least integer n such that $\sigma_n > min(\beta, \mu)$ is at most $\lceil \frac{1}{\sigma_1} \rceil$. This in turn implies that the fixpoint evaluation of P_{10} on every EDB terminates in a finite number of iterations, the exact number of which depends on the particular rule certainties involved in P_{10} and the EDB. Note that the certainty σ of A at iteration $n = \lceil \frac{1}{\sigma_1} \rceil$ is 1, whenever $\sigma_i < min(\beta, \mu)$, for all $1 \leq i < \lceil \frac{1}{\sigma_1} \rceil$. Otherwise, $\sigma < 1$.

4.5.3.2 Lattice-Theoretic Conjunction but Arbitrary Propagation

Let P_{11} be the instance of PM_1 in which $f^d = nc$, $f_2^p = *$, and $f_2^c = min$. Then, the bottom-up naive evaluation of P_{11} on the EDB D proceeds as follows, as far as $A = p(1,2)$ is concerned. At iteration 1, we derive A with certainty $\sigma_1 = f_1^p(\alpha, \eta)$. At iteration 2, A is derived with certainties σ_1 and $\sigma_1\beta$, combining which yields $\sigma_2 = nc(\sigma_1, \sigma_1\beta) = \sigma_1(1 + \beta)$, assuming that $\sigma_1 + \sigma_1\beta \leq 1$. If this assumption does not hold at some iteration n, then the evaluation terminates at iteration $n + 1$. At iteration 3, there are two derivations of A with certainties σ_1 and $\sigma_2\beta$, and hence $\sigma_3 = nc(\sigma_1, \sigma_2\beta) = \sigma_1(1 + \beta + \beta^2)$. It can be shown that A's certainty at iteration

$n \geq 1$ is:

$$\sigma_n = \sigma_1(1 + \beta + \cdots + \beta^{n-1}),$$

or equivalently,

$$\sigma_n = \frac{\sigma_1(1 - \beta^n)}{1 - \beta},$$

where $\beta < 1$. Note that since $f_2^p = *$, the case where $\beta = 1$ reduces to the previous case, where we showed the number of iterations required for computing the fixpoint of P_{11} on any EDB is at most $n = \lceil \frac{1}{\sigma_1} \rceil$. Also note that this evaluation terminates at iteration $n + 1$ whenever $\sigma_n \geq \mu$. It then follows from the above formula for σ_n that A's certainty in the limit is $\sigma = \lim_{n \to \omega} \sigma_n = \sigma_1/(1 - \beta)$.

Our remarks in Subsection 4.5.2.2 for the corresponding case with type 2 disjunction function also hold in this case.

4.5.3.3 Lattice-Theoretic Propagation but Arbitrary Conjunction

Let P_{12} be an instance of our program template PM_1 with $f^d = nc$, $f_2^p = min$, and $f_2^c = *$. Also let $A = p(1, 2)$. Then the bottom-up evaluation of P_{12} on D proceeds as follows, as far as atom A is concerned. At iteration 1, we prove A with certainty $\sigma_1 = f_1^p(\alpha, \eta)$. At iteration 2, there are two derivations of A with certainties σ_1 and $\sigma_1\mu$, and hence $\sigma_2 = \sigma_1 + \sigma_1\mu$, assuming that the sum is at most 1, by definition of nc. At iteration 3, we derive A with certainties σ_1 and $\sigma_2\mu$, and thus $\sigma_3 = \sigma_1 + \sigma_2\mu = \sigma_1(1 + \mu + \mu^2)$. It can be verified that at iteration $n \geq 1$, we have $\sigma_n = \sigma_1(1 + \mu + \cdots + \mu^{n-1})$ or, equivalently, $\sigma_n = \sigma_1(1 - \mu^n)/(1 - \mu)$, if $\mu \neq 1$. If $\mu = 1$, then clearly this evaluation terminates at iteration $j + 1$, where $j = \lceil \frac{1}{\sigma_1} \rceil$. It also terminates at iteration $k+1$, whenever $\sigma_k\mu \geq \beta$. When this evaluation terminates only at ω, then $\sigma = \sigma_1/(1 - \mu)$ is the certainty of A obtained by computing the limit of σ_n as n reaches ω.

4.5.3.4 Arbitrary Propagation and Conjunction

Consider an instance P_{13} of PM_1 in which $f^d = nc$ and $f_2^p = f_2^c = *$. Then the bottom-up naive evaluation of P_{13} on D proceeds as follows, considering the ground atom $A = p(1, 2)$. Clearly, $\sigma_1 = f_1^p(\alpha, \eta)$. At iteration 2, we apply r_1 and r_2 and derive A with certainties σ_1 and $\sigma_1\beta\mu$, and hence $\sigma_2 = nc(\sigma_1, \sigma_1\beta\mu) = \sigma_1 + \sigma_1\beta\mu(1 - \sigma_1)$.

Subsection	f_2^p & f_2^c	σ_n	Terminates at iteration $n+1$ if	$\sigma = \lim_{n\to\omega} \sigma_n$
4.5.3.1	$f_2^p = f_2^c = min$	$n\sigma_1$	$\sigma_n \geq min(\beta, \mu)$	1
4.5.3.2	$f_2^p = * \;\&\; f_2^c = min$	$\frac{\sigma_1(1-\beta^n)}{1-\beta}$	$\sigma_n \geq \mu$	$\frac{\sigma_1}{1-\beta}$
4.5.3.3	$f_2^p = min \;\&\; f_2^c = *$	$\frac{\sigma_1(1-\mu^n)}{1-\mu}$	$\sigma_n\mu \geq \beta$	$\frac{\sigma_1}{1-\mu}$
4.5.3.4	$f_2^p = f_2^c = *$	$\frac{\sigma_1(1-\beta^n\mu^n)}{1-\beta\mu}$	$\sigma_n = 1$	$\frac{\sigma_1}{1-\beta\mu}$

Table 3: Termination property of PM_1 when $f^d = nc$

At iteration 3, we obtain $\sigma_3 = nc(\sigma_1, \sigma_2\beta\mu) = \sigma_1 + \sigma_1\beta\mu(1-\sigma_1) + \sigma_1\beta^2\mu^2(1-\sigma_1)^2$. It is straightforward to verify that A's certainty at iteration $n \geq 1$ is:

$$\sigma_n = \sigma_1 \sum_{i=0}^{n-1} \beta^i\mu^i = \frac{\sigma_1(1-\beta^n\mu^n)}{1-\beta\mu}.$$

The certainty of A in the limit would be:

$$\sigma = \lim_{n\to\omega} \sigma_n = \frac{\sigma_1}{1-\beta\mu},$$

if this evaluation terminates only at ω. This happens whenever $\beta\mu < 1$, i.e., whenever $\beta < 1$ or $\mu < 1$.

Note that when $\beta = \mu = 1$, the certainty of A at iteration n is $n\sigma_1$, no matter what propagation and conjunction functions are associated with the recursive p-rule r_2. In this case, the fixpoint evaluation of P_{13} on every EDB always terminates at iteration $n+1$, where $n = \lceil \frac{1}{\sigma_1} \rceil$.

Table 3 summarizes the results of our analysis above on the termination property for the generic p-program P_2 with nc as the disjunction function. As expected, the table shows that when $\beta = 1$, or in general when $\beta = \top$, the choice of the propagation function f_2^p is not crucial for termination. That is, when $\beta = 1$, the values of σ_n in the first two rows in the table would be the same. The same remark holds for the values of σ in these two rows. The table also indicates that when $\mu = 1$, i.e. when the certainty of the instance $e(1,1)$ of the subgoal $e(X,Z)$ in r_2 is the top, then the particular choice of the conjunction function f_2^c is not crucial for termination. It then

follows from these remarks that when $\beta = \mu = 1$, all four rows in the table would collapse into one, and hence the distinction between them goes away.

It can be seen from Tables 2 and 3, that the main difference on the termination behavior between the instances of PM_1 in which $f^d = ind$ and $f^d = nc$ is when the propagation and conjunction functions associated with the recursive rule r_2 are both lattice-theoretic. In this case, when $f^d = nc$ is the disjunction function associated with the recursive predicate p in an instance of PM_1 considered, the bottom-up evaluation of the program on *any* EDB would *always* terminate in finite time, while this may not be the case when $f^d = ind$, as was shown in Subsection 4.5.2.1.

The following conjecture generalizes our analysis results in this section to any p-program defining recursive predicates of type 3 (and possibly of type 1 as well). A proof of this relies on our observations in the special case analysis performed in this section.

Conjecture 4.5.5 *Let P be a p-program, on the lattice $\langle \mathcal{T}, \preceq, \otimes, \oplus \rangle$, such that every recursive predicate in P is of type 1 or 3, and at least one of them is of type 3. Let $r_1, \ldots, r_k$ be all the recursive p-rules in P defining type 3 predicates.*

(a) If for every r_i, $1 \leq i \leq k$, the propagation and conjunction functions associated with r_i are $\otimes$, then the bottom-up naive evaluation of P on every *EDB* always *terminates in finite time.*

(b) Suppose for some r_i, $1 \leq i \leq k$, at least one of the propagation and conjunction functions associated with r_2 is $\prec \otimes$. Then there are p-programs Q_1 and Q_2 identical to P, except possibly for rule certainties, and there are EDBs D_1 and D_2 such that (i) Q_1's evaluation on D_1 terminates in finite time, and (ii) Q_2's evaluation on D_2 will not terminate.

4.5.4 Disjunction Functions of Types 2 and 3

So far in our analysis of the termination property of p-programs, we have considered p-programs defining recursive predicates of type 2 or type 3, but not both. In this section, we study programs defining both types of predicates. The program template we use for this study is as follows, which defines two mutual recursive predicates p and q, where p is of type 2 and q is of type 3.

Example 4.5.6 (Program Template 2) Let PM_2 be the following p-program:

$$\begin{array}{lll} r_1: & p(X,Y) \xleftarrow{\alpha_1} e(X,Y); & \langle ind, f_1^p, \_ \rangle. \\ r_2: & p(X,Y) \xleftarrow{\alpha_2} q(X,Z),\ p(Z,Y); & \langle ind, f_2^p, f_2^c \rangle. \\ r_3: & q(X,Y) \xleftarrow{\alpha_3} r(X,Y); & \langle nc, f_3^p, \_ \rangle. \\ r_4: & q(X,Y) \xleftarrow{\alpha_4} p(X,Z),\ q(Z,Y); & \langle nc, f_4^p, f_4^c \rangle. \end{array}$$

where $\alpha_i \in (0,1]$ is the certainty associated with r_i, for $1 \leq i \leq 4$, and f_i^p and f_i^c are the propagation and conjunction functions associated with r_i. The underlying certainty lattice used in this program is $\langle [0,1], \leq, min, max \rangle$. For ease of presentation, and without loss of generality, let us assume the EDB relations r and e are identical. As the EDB D, we will sometimes consider $\{e(1,1) : \mu,\ e(1,2) : \eta\}$ and some other times consider $\{e(1,1) : \mu\}$, where $\mu, \eta \in (0,1]$. ■

We remark that one could consider as the program template in this case, a generic p-program P' in which the recursively defined predicates p and q were "less related" or "unrelated", as opposed to being mutually recursive. In this case, however, program's termination could be characterized by combining, in an appropriate manner, the "relevant" individual results obtained in Subsections 4.5.2 and 4.5.3. For instance, suppose min is both the propagation and conjunction functions associated with every recursive p-rule in an instance of such a program template. Then, the fixpoint evaluation of this program on every EDB reaches the fixpoint in a finite number of iterations, as far as q-atoms are concerned, as shown in part (a) of Theorem 4.5.5. In this case, termination of this evaluation depends on how it proceeds w.r.t. p-atoms, which is characterized by Theorem 4.5.4. Also note that, in general, the number of iterations required for evaluating an instance P of PM_2 on any EDB D, whenever this evaluation terminates in finite time, is no more than that required for evaluating P' on D. Intuitively, this is because the certainty associated with every IDB subgoal in r_2 (and in r_4) increases at every iteration, in general, which in turn would cause the evaluation of $P \cup D$ to terminate in an earlier iteration than evaluation of $P' \cup D$ would.

Our observation in the cases studied in this subsection is that, in general, they can be dealt with using the results obtained in the preceding two subsections. In other words, the issues involved here are fundamentally the same as those discussed in Subsections 4.5.2 and 4.5.3, although the recurrence equations we have to deal

with here are simultaneous non-linear, unlike those we dealt with previously which were linear. This introduces some complications in solving such equations. What we will do in this case is, by making some assumptions on the actual certainties used in a given p-program, we attempt to reduce the system of non-linear simultaneous recurrence equations to a linear one. What we hope to achieve in this way is the ability to "predict" the termination behavior of p-programs defining recursive predicates of mixed types, including types 2 and 3. The kind of result we thus establish in this subsection is as follows. Given an instance P of PM_2, there are p-programs Q_1 and Q_2, obtained from P perhaps with different rule certainties, and there are EDBs D_1 and D_2, such that the fixpoint naive evaluation of Q_1 on D_1 terminates in finite time, and that of Q_2 on D_2 will not terminate.

As in the preceding subsections, in what follows we consider four cases depending on how the propagation and conjunction functions associated with the recursive p-rules in PM_2 compare with the meet operator in the underlying certainty lattice.

4.5.4.1 Lattice-Theoretic Propagation and Conjunction

Consider an instance P_{14} of PM_2 in which $f_2^p = f_2^c = f_4^p = f_4^c = min$, i.e., the propagation and conjunction functions associated with every recursive p-rule in the program coincides with the lattice meet. Consider the bottom-up naive evaluation of P_{14} on $D = \{\epsilon(1,1) : \mu.\ \epsilon(1,2) : \eta\}$, for which we will be concerned only with atoms $p(1,2)$ and $q(1,2)$, for ease of presentation. Clearly, we can always ensure that ignoring $p(1,1)$ and $q(1,1)$ will not invalidate our conclusions. One way to do this is to choose the rule certainties in P_{14} and D such that the certainties associated with $p(1,1)$ and $q(1,1)$ reach their peak values before the certainties of $p(1,2)$ and $q(1,2)$ reach theirs. Therefore, when we say that the fixpoint evaluation of PM_2 on D terminates at certain iteration, we assume the best possible certainties of $p(1,1)$ and $q(1,1)$ are already obtained in some earlier iterations.

We have the following system of recurrence equations, which captures this evaluation of P_{14} on D, where σ_n and δ_n denote, respectively, the certainty of $p(1,2)$ and

$q(1,2)$ obtained at iteration n.

$$\begin{aligned}
\sigma_1 &= f_1^p(\alpha_1, \eta) \\
\sigma_{n+1} &= ind(\sigma_1, min(\alpha_2, min(\sigma_n, \delta_n))), \quad n \geq 1 \\
\delta_1 &= f_3^p(\alpha_3, \eta) \\
\delta_{n+1} &= nc(\delta_1, min(\alpha_4, min(\sigma_n, \delta_n))), \quad n \geq 1
\end{aligned}$$

Since the propagation and conjunction functions associated with r_4 are min, it follows from our analysis in Subsection 4.5.3.1 that the fixpoint w.r.t. q-atoms is obtained in a finite number of iterations. Let δ be the certainty of $q(1,2)$ in the fixpoint, obtained at iteration j, where $1 \leq j \leq \lceil \frac{1}{\delta_1} \rceil$. Note that δ need not be 1 and is determined by σ_j, α_4, α_3, and η. Also as explained earlier, the existence of the IDB $p(X,Z)$ as a subgoal in r_4 may help the certainty δ_j reach its peak, δ, faster than it would if we had an EDB subgoal instead, as in the case studied in Subsection 4.5.3.1. This is because here, the certainty of the instance $p(1,1)$ in r_4 involved in deriving $q(1,2)$ may also increase, which in turn may cause faster termination w.r.t. q-atoms. The fact that in this case such an integer j exists allows us in the above equation to replace δ_n with δ. This yields the following system of equations.

$$\begin{aligned}
\sigma_1 &= f_1^p(\alpha_1, \eta) \\
\sigma_{n+1} &= ind(\sigma_1, min(\alpha_2,\ min(\sigma_n,\ \delta))) \quad n \geq j
\end{aligned}$$

assuming that this evaluation requires at least j iterations before the fixpoint is reached. Clearly, when at some iteration n, $\sigma_n \geq min(\delta, \alpha_2)$, this evaluation terminates at iteration $n+1$.

Now, with the simplified system of equations, termination of this evaluation depends on how it proceeds w.r.t. p-atoms. But then, as we know from our analysis in Subsection 4.5.2.1, this evaluation may terminate in arbitrarily large number of iterations or it may not terminate at all, depending on the rule certainties α_1, α_2 in P_1 and μ, η in D. For instance, let $D_1 = D$, with $\mu = \eta = 1$. Let Q_1 be the same as P_1 in which α_2 is any value in $(0, \sigma)$, where σ is the certainty of $p(1,2)$ in the limit. Note that σ can be obtained by using the formula derived in Subsection 4.5.2.1. For the certainties of other p-rules in Q_1, we can take any value in $(0,1]$, the particular choice of which may only affect the iteration at which Q_1's evaluation on D_1 terminates. To define Q_2 from P_1, let $\alpha_i = 1$, for $1 \leq i \leq 4$. This will ensure that when evaluating Q_2 on D_2, the certainty associated with the instance $p(1,2)$ of $p(Z,Y)$ in the body of

r_2 will never be more than the minimum of α_2, the certainty associated with r_2, and the certainty in the limit of the instance $q(1,1)$ of $q(X,Z)$. This in turn implies that the fixpoint evaluation of Q_2 on D does not terminate.

We may thus conclude from the above analysis that for the case at hand, the termination characterization reduces to the corresponding basic case discussed in Subsection 4.5.2.1, in which the propagation and conjunction functions associated with the recursive p-rule were both *min*.

4.5.4.2 Lattice-Theoretic Conjunction but Arbitrary propagation

Let P_{15} be an instance of PM_2, in which $f_2^c = f_4^c = min$ and $f_2^p = f_4^p = *$. Now, consider the bottom-up naive evaluation of P_{15} on $D = \{e(1,1) : \eta\}$, where $\eta \in (0,1]$. The system of recurrence equations we get in this case is as follows, in which σ_n and δ_n denote, respectively, the certainties of $p(1,1)$ and $q(1,1)$ at iteration n.

$$\begin{aligned}
\sigma_1 &= f_1^p(\alpha_1, \eta) \\
\sigma_{n+1} &= ind(\sigma_1,\ \alpha_2 * min(\sigma_n, \delta_n)), \quad n \geq 1 \\
\delta_1 &= f_3^p(\alpha_3, \eta) \\
\delta_{n+1} &= nc(\delta_1,\ \alpha_4 * min(\sigma_n, \delta_n)), \quad n \geq 1
\end{aligned}$$

As in the previous case, we first try to simplify the above system by making some assumptions on the possible certainties which may be used in an instance of P_{15} and the EDB. It then follows from our analysis in Subsections 4.5.2.2 and 4.5.3.2, in which we used the same propagation and conjunction functions, that we can determine instances of the rule certainties in P_{15} and D such that the fixpoint evaluation of the resulting database terminates in finite time. We can also determine instances of these certainties such that the fixpoint evaluation of the resulting p-program on some EDB will not terminate. There are various ways in which this can be accomplished. For instance, we can first determine instances of the rule certainties involved in defining q-atoms, $q(1,1)$ in this case, such that our evaluation saturates w.r.t. q-atoms in a finite number of iterations. Once this is done, we can either determine the rule certainties involved in defining $p(1,1)$ such that this evaluation terminates in finite time, or determine them such that program's evaluation will not terminate. For instance, let $\alpha_1 < 1$, $\alpha_3 = 1$, and $\alpha_4 \in (0,1]$. It can be shown that when $\alpha_2 < 1$, the fixpoint evaluation of P_{15} on $D = \{e(1,1) : 1\}$ terminates in finite time. More

precisely, if we want this evaluation to terminate at iteration step $n+1$, we can use the formula:

$$\sigma_n = \frac{\sigma_1[1-\alpha_2^n(1-\sigma_1)^n]}{1-\alpha_2(1-\sigma_1)}$$

which yields the certainty of $p(1,1)$ at iteration n. This formula is derived in Subsection 4.5.2.2 in which we replaced β by α_2. Now, for this evaluation to terminate at iteration $n+1$, we can first calculate the certainty σ_n from the above formula and then choose the certainty of $q(1,1)$ not less than σ_n. In fact, this is already satisfied by our choice of 1 as the certainties of both η and α_3, since the certainty of $q(1,1)$ in the fixpoint is 1.

To get an instance of P_{15} which does not terminate on some EDB, let $\alpha_2 = 1$ and the other rule certainties be as before. It can then be easily verified that the fixpoint evaluation of the resulting p-program does not terminate on some EDB, e.g. $D = \{e(1,1) : 1\}$. Note that as in the previous case, the certainty of $q(1,1)$ in the fixpoint is 1, obtained in the first step.

4.5.4.3 Lattice-Theoretic Propagation but Arbitrary Conjunction

We next consider p-programs defining recursive predicates of types 2 and 3 in which the propagation and conjunction functions associated with every recursive p-rule defining such predicates are min and $*$, respectively.

Let P_{16} be an instance of PM_2, defined in Example 4.5.6, in which $f_2^p = f_4^p = min$ and $f_2^c = f_4^c = *$. Suppose the EDB contains only $e(1,1) : \eta$, where $\eta \in (0,1]$. Consider the bottom-up naive evaluation of P_{16} on D, captured by the following system of recurrence equations.

$$\begin{aligned}
\sigma_1 &= f_1^p(\alpha_1, \eta) \\
\sigma_{n+1} &= ind(\sigma_1, min(\alpha_2, \sigma_n * \delta_n)), \quad n \geq 1 \\
\delta_1 &= f_3^p(\alpha_3, \eta) \\
\delta_{n+1} &= nc(\delta_1, min(\alpha_4, \sigma_n * \delta_n)), \quad n \geq 1
\end{aligned}$$

The solution to this system of equations yields the certainties σ_n of $p(1,1)$ and δ_n of $q(1,1)$ at iteration n. Suppose $\alpha_2 \leq \alpha_4$. The case where this does not hold is treated in a similar way. Clearly, if at some iteration j, $\alpha_4 \leq \sigma_j\delta_j$, then this evaluation terminates at iteration $j+1$. To let this evaluation proceed, we can assume either

$\sigma_j\delta_j < \alpha_2$ or $\alpha_2 < \sigma_j\delta_j$. Assume the former is the case. Then, we have the following equations for σ_{n+1} and δ_{n+1} obtained from the above.

$$\begin{aligned}\sigma_{n+1} &= \sigma_1 + \sigma_n\delta_n - \sigma_1\sigma_n\delta_n \\ \delta_{n+1} &= \delta_1 + \sigma_n\delta_n\end{aligned}$$

assuming $\delta_1 + \sigma_n\delta_n \leq 1$, by definition of nc. Although these equations are simpler then the previous ones, they still form a system of non-linear simultaneous recurrence relations, an exact solution to which does not appeal to us. As discussed earlier in this section, we may not even need to find the exact solution, as our objective in this study has been to develop an approximate formulation of the termination, characterizing what can be "expected" when we evaluate a given p-program.

It can be easily seen that when at least one of the rule certainties α_2 and α_4 is 1, then this evaluation terminates only at ω, at which the certainty associated with each of $p(1,1)$ and $q(1,1)$ is 1. Similarly, we can see that when $\alpha_2 < 1$ and $\alpha_4 < 2$, the fixpoint evaluation of P_{16} on D always terminates in a finite number of iterations, and the certainty computed for each atom in this case is less than 1. The exact step in which this evaluation terminates is determined by the particular rule certainties used in P_{16} and the EDB.

We may conclude from the arguments above together with our results in Theorems 4.5.4 and 4.5.5 that, given an instance P_{16} of PM_2, there are p-program Q_1 and Q_2, obtained from P_{16}, with possibly different rule certainties, and there are EDBs D_1 and D_2 such that the bottom-up naive evaluation of Q_1 on D_1 terminates in finite time, and that of Q_2 on D_2 will not terminate. For example, to define Q_1, we can take $\alpha_i < 1$, for $1 \leq i \leq 4$. For Q_2, we can take, for instance, $\alpha_1 < 1$, $\alpha_2 = 1$, $\alpha_3 < 1$, and $\alpha_4 = 1$. In both cases, we let η be any certainty in $(0,1]$, the particular choice of which does not affect termination.

4.5.4.4 Arbitrary Propagation and Conjunction

Let P_{17} be an instance of PM_2 in which $*$ is the propagation and conjunction function associated with the recursive p-rules r_2 and r_4. Note that $*$ is chosen since $* < \otimes$ on certainties different from the top and bottom elements. Then, the following recurrence equations capture the fixpoint evaluation of P_{17} on the EDB $D = \{e(1,1) : \eta\}$, where

η is any fixed certainty in $(0, 1]$.

$$\begin{aligned} \sigma_{n+1} &= \sigma_1 + \alpha_2\sigma_n\delta_n - \sigma_1\alpha_2\sigma_n\delta_n, \quad n \geq 1 \\ \delta_{n+1} &= \delta_1 + \alpha_4\sigma_n\delta_n, \quad n \geq 1 \end{aligned}$$

where $\sigma_1 = f_1^p(\alpha_1, \eta)$ and $\delta_1 = f_3^p(\alpha_3, \eta)$. Although the above system of equations is, in general, harder to solve than the one we got in the previous case, the termination property is not expected to be fundamentally different, as there is a close relationship between the two. This follows from the fact that when $\alpha_2 = \alpha_4 = 1$, this case reduces to the previous one, and also from the kinds of combination functions involved in these two cases together with our analysis results in Subsections 4.5.2.4 and 4.5.3.4.

On the basis of our analysis in this section, we conjecture the following which generalizes our observations in the special case analysis using the program template PM_2.

Conjecture 4.5.7 *Let P be a p-program defining at least two recursive predicates, one of type 2 and one of type 3. Then, there are p-programs Q_1 and Q_2, which are the same as P, except possibly for rule certainties, and there are EDBs D_1 and D_2, such that the bottom-up naive evaluation of Q_1 on D_1 terminates in finite number of iterations, and that of Q_2 on D_2 will not terminate.*

4.6 Summary and Concluding Remarks

The termination property of p-programs formulated in this chapter were in the context of programs with uncertainty in the parametric framework. Our results, however, are applicable in the more general context of fixpoint computation with aggregation. Kifer and Li [KL88] also studied the termination property of datalog with uncertainty in the context of their AB framework. It is not settled under what conditions one can expect finite termination when evaluating programs in their framework, unless for a class of programs which use *max* as disjunction function which clearly enjoy PTIME data complexity. The idea of using the recurrence-based evaluation in our work is borrowed from [KL88]. However, unlike us, they used the method as an alternative to the conventional naive method. The recurrence-based evaluation they used, although elegant, is a departure from the conventional query processing method,

and the possibility of efficient implementation of their method is unclear. In our work, we used this method to compute the certainties associated with the derived atoms at different iterations, including the limit, as a by product of studying the termination property of p-programs. The difficulty faced was that in some cases exact iteration at which an evaluation terminates could not be determined. In such cases, we opt for an approximate characterization of termination, which to some extent provided useful insights as to how the choice of combination functions affects termination and complexity of the fixpoint evaluation of p-programs.

The systematic analysis performed in this chapter shows that the distinction between type 2 and type 3 disjunction functions can be best explained when the propagation and conjunction functions associated with every recursive rule coincide with the meet operator in the underlying certainty lattice. We also remark that our results in this chapter on the termination and the complexity of the fixpoint evaluations of p-programs are useful also for the AB frameworks with uncertainty. In particular, these results are applicable to AB programs which do not use annotation constants, since as we will show in the following chapter, the computation of such programs can be simulated by the parametric framework. Next chapter is dedicatd to the relationship between the AB and IB approaches to uncertainty.

In our study in this chapter, we considered p-programs over a complete lattice, i.e., $[0,1]$, assuming that the (arithmetic) computations (over reals) could be carried out with arbitrary precision. In practice, this assumption may not always be valid. That is to say under this assumption, there could be a mismatch between the theoretical results and the actual computations with uncertainty. Before demonstrating this, let us mention some observations with the so-called *finite precision assumption*, made by Graumbach and Su in the context of *constraint databases* [GJ96]. They pont out that some results, which are normally taken for granted, may break down under this assumption. For instance, for all real numbers X, Y, the formula $\exists Y \forall X (X \leq Y)$ would always be true, which says there always exists the greatest real number. Also, the distributive law which holds on reals may not hold anymore under the assumption above, i.e., $a \times (b+c)$ might be different from $(a \times b)+(a \times c)$. Furthermore, unlike with real numbers, arithmetic computations with finite precision are sensitive to the order in which the subexpressions are evaluated in query processing. In our context, we note that arithmetic computations under the finite precision assumption may cause

termination of a fixpoint evaluation of a p-program while theoretically it was expected to run forever. The following example demonstrates this.

Example 4.6.1 *Consider the p-program below.*

$$r_1: \; p(X) \xleftarrow{0.8} e(X); \qquad \langle ind, *, \_ \rangle.$$
$$r_2: \; p(X) \xleftarrow{1} e(X),\ p(X); \qquad \langle ind, *, * \rangle.$$

If we evaluate this program on the EDB $\{e(a) : 1\}$, using our top-down implementation of the parametric framework introduced in Section 6.2, the sequence of certainties obtained for $p(a)$ is 0.8, 0.96, 0.9984, and 1, in the order specified. Theoretically, this should have been an infinite sequence. However, due to finite precision, this sequence converged in a finite number of iterations. The exact iteration number at which this convergence happens is dictated by the run-time environment – both the hardware and software systems. ■

Chapter 5

Expressive Power

In Chapter 1, we introduced a classification of approaches to logic programming and deductive databases with uncertainty into what we called as annotation based (AB) and implication based (IB) approaches. We have also seen the relative pros and cons of these approaches in that chapter. An important metric in a comparison of these two approaches is their relative expressive power: Are there queries one can express in one of the AB (or IB) frameworks which cannot be expressed in the other?

Before we can address this question, we should point out that it refers to two families of frameworks, rather than two specific frameworks. Since we have generalized all known IB frameworks and unified them into our parametric framework, we can take the latter as a representative of all IB frameworks. If there was a similar generic AB framework available, we could compare the expressive power of the two generic frameworks – one for IB and one for AB – and address the above question. Unfortunately, no such generic AB framework is known.

A related point to note is that Kifer and Subrahmanian [KS92] have shown that their GAP framework (which is AB) can simulate the IB framework of van Emden [van86]. No result in the converse direction is known.

In this chapter, we point out the limitations in the expressive power of known IB and AB frameworks and motivate a paradigm of uncertainty programming with (certainty) constraints. Motivated by this need, we propose an extension to the parametric framework, called the *extended parametric (EP) framework*. To facilitate the comparison of expressive power, we develop a generic AB framework, to which

we refer as the *basic GAB framework*. To give an idea, the basic GAB framework developed includes the probabilistic logic programming framework of Ng and Subrahmanian [NS92] as a special case. For the comparison to make sense, we assume that the annotation functions in the basic GAB framework are subject to the same set of postulates as in the parametric framework. Finally, we show that the parametric framework can simulate all programs in the basic GAB framework which do not have annotation constants or repeated annotation variables in rule bodies. We will also extend the basic GAB framework with certainty constraints, to which we refer as the *extended GAB framework*. We will show that the EP framework is equivalent to the extended GAB framework from the point of view of the expressive power. We conclude this chapter with an observation that while the incorporation of certainty constraints strictly increases the expressive power of both parametric and AB frameworks, it comes at a price. While the basic parametric framework has a continuous immediate consequence operator, this continuity is lost when constraints are added.

5.1 Certainty Constraints

A useful operation in deductive databases with uncertainty is to select from a relation all the tuples associated with a certainty not "less" than a specified threshold, e.g., as in the query "find all red objects whose degree of redness is at least 0.75". We call this operation as *selection by certainty*. Another useful operation is *join by certainty*, which amounts to prescribing that a pair of tuples in two relations can be joined provided their associated certainties stand in a certain relationship. We can view the selection/join by certainty as a filtering mechanism, by which we restrict the tuples which can participate in the selection/join operations in the body of a rule to (possibly empty) subsets of the relations in body. Our parametric framework, however, does not support these two operations. To accommodate them, we propose an extension to our framework, and refer to the resulting framework as the *extended parametric (EP) framework*. This is achieved by allowing *certainty constraints* in the rule bodies.

A certainty constraint is of one of the following forms:

- $wt(A)\ \theta\ \sigma$
- $wt(A)\ \theta\ wt(B)$

where A and B are both atoms, σ is an element of the certainty lattice $\mathcal{T}$, and θ is one of $=$, $\prec$, $\preceq$, $\succ$, $\succeq$, and $\neq$. Intuitively, $wt(A)$ can be viewed as a function which returns the certainty (or the weight) associated with atom A.

The EP framework proposed above allows each rule to contain in the body a conjunction of certainty constraints. More precisely, a rule in the EP framework (ep-rule, for short) is an expression of the form:

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

where $Constratint_r$ is a conjunction of certainty constraints such that in every term $wt(A)$ specified in the constraint, A is an atom of the form $q_i(\overline{Y}_i)$, for $1 \leq i \leq n$. We refer to programs in the EP framework as *extended p-programs* or ep-programs, for short. To illustrate how certainty constraints could be useful, let us consider the following example.[1]

Example 5.1.1 Consider a medical application where uncertain knowledge about particular diseases and symptoms is represented as the following ep-rules.

$$\begin{aligned}
r_1:\ & disease(X,D) \stackrel{0.8}{\longleftarrow} has(X,S),\ symptom(D,S),\ wt(has(X,S)) \geq 0.8,\\
& \quad wt(symptom(D,S)) > wt((has(X,S));\ \langle max,*,min\rangle.\\
r_2:\ & disease(X,D) \stackrel{0.9}{\longleftarrow} family_history(X,D),\ hereditary(D),\\
& \quad wt(family_history(X,D)) \geq 0.8,\\
& \quad wt(hereditary(D)) \geq 0.7;\ \langle max,*,min\rangle.
\end{aligned}$$

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

The notion of satisfaction of ep-programs by valuations is very similar to the corresponding notion for p-programs, given in Definition 2.3.1, except that here constraints should be also satisfied. Satisfaction of constraints by valuations is defined as follows. Let P be an ep-program, and v be any valuation of P. Suppose $C_1,\ldots,C_k$ is the conjunction of certainty constraints specified in the body of an ep-rule in P, where C_ℓ,

[1]This example without constraints appeared in [Lak94] and was given in a different context.

$1 \leq \ell \leq k$, is either an expression of the form (1) $wt(B_i)\,\theta\,\sigma$ or (2) $wt(B_i)\,\theta\,wt(B_j)$, where $\sigma \in \mathcal{T}$, and $\theta \in \{=, \neq, \prec, \succ, \preceq, \succeq\}$. We say that v satisfies C_i, denoted $\models_v C_i$, provided $v(B_i)\,\theta\,\sigma$ is true in case 1, and $v(B_i)\,\theta\,v(B_j)$ is true in case 2. We say that v satisfies $C_1, \ldots, C_k$, provided $\models_v C_\ell$, for all ℓ, $1 \leq \ell \leq k$.

5.2 A Generic Annotated Logic Framework

Before comparing the expressive power of EP with the AB approach, we point out the following.

1. The AB framework to which we compare our EP framework does *not* exactly correspond to any existing AB framework, rather it is a generic language which includes the essential features of the AB approach. We refer to this language as the basic *generic AB (GAB)* framework. We will also consider an extension of the basic GAB in which certainty constraints are allowed in the rule bodies. Let us call the resulting language as the *extended GAB*. A rule r in the extended GAB framework is an assertion of the form:

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

where V_i is a *certainty constant* or *certainty variable*, for $1 \leq i \leq n$, and $Constraint_r$ is a conjunction of constraints in r each of which is a relation of the form $V_i\,\theta\,\sigma$ or $V_i\,\theta\,V_j$, where $\sigma \in \mathcal{T}$, and θ is any of the $=$, $\neq$, $\prec$, $\succ$, $\preceq$, and $\succeq$, w.r.t. the certainty lattice $\mathcal{T}$.

Note that certainty constraints are allowed in the extended GAB framework, but not in the basic GAB. Also note that the extended GAB framework so defined is strictly more expressive than any AB framework. To see this, suppose we want to perform a join operation on tuples t_q and t_r in relations q and r, respectively, such that the certainty associated with t_q is not "less" (w.r.t. the ordering $\preceq$ on $\mathcal{T}$) than the certainty of t_r. While this cannot be expressed in any AB framework, including the basic GAB, it can be expressed as the following ep-rule in the extended GAB framework.

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

where f is a desired annotation function.

2. For the comparison between the EP and the extended GAB frameworks to make more sense, we require that the annotation functions allowed in GAB satisfy the corresponding postulates imposed on the combination functions in the parametric framework (Definition 2.1.1). Recall that annotation functions in the AB approach, including the basic and the extended GAB frameworks, play the roles of both conjunction and propagation functions in the parametric framework (and hence the EP).

3. Our motivation for extending the parametric framework with constraints to define the EP framework was to accommodate the operations of selection and join by certainty. However, in terms of expressive power, we can show that the parametric framework (without constraints) can simulate the computation of any program in the basic GAB in which the rules do not contain annotation constants or repeating annotation variables. This is established by the following proposition.

Proposition 5.2.1 *For every program P in the basic GAB framework such that no annotation constants or repeating annotation variables appear in the rules of P, there exists a program P' in the parametric framework such that on every EDB D, P and P' compute the same atom-certainty pairs.*

Proof. We will show how a desired p-program P' could be constructed by showing how an arbitrary rule in P would be transformed into a p-rule in P'. Let $\mathcal{T}$ be the underlying certainty lattice in P. Then, any rule r in P is an assertion of the form:

$$r: \quad p(\overline{X}) : g(V_1, \ldots, V_n) \leftarrow q_1(\overline{Y}_1) : V_1, \ldots, q_n(\overline{Y}_n) : V_n$$

where V_i's are annotation variables and by our assumption, $V_i \neq V_j$, whenever $i \neq j$, for all $1 \leq i, j \leq n$. Suppose f_d is the disjunction function associated with the head predicate, p. Then corresponding to this rule in P, the p-program P' would contain the p-rule:

$$r': \quad p(\overline{X}) \xleftarrow{\top} q_1(\overline{Y}_1), \ldots, q_n(\overline{Y}_n); \quad \langle f_d, g, g \rangle$$

where $\top$ is the top element in $\mathcal{T}$. This completes the transformation process, whenever $n > 0$. (If $n = 0$, then r is of the form $p(\overline{X}) : \alpha$, in which case, P' would contain $p(\overline{X}) \xleftarrow{\alpha}$.) Next, we show that the fixpoint evaluations of P and P' on any EDB

produce the same results. To this end, we use the induction proof technique, where the induction is on the number i of iterations required to compute the fixpoints. Let D be any EDB, and consider the bottom-up naive evaluations of P and P' on D. For every atom-certainty $A : \alpha$ derived at iteration i by r in P, we will show that there is a corresponding derivation of $A : \alpha$ at iteration i obtained by r' in P'.

Basis case: $i = 0$. The result trivially follows upon noting that P and P' are evaluated on the same EDB.

Inductive step: Suppose P and P' define the same atom-certainty pairs at some iteration $i \geq 0$. By our assumption, since there is no annotation constants or repeating annotation variables in the body of r, this rule can be fired exactly when every subgoal in its body is satisfied. That is, every subgoal has a certainty which is greater than $\perp$. It then follows from our construction that in this case, the body of r' is also satisfied and hence r' can be fired. Since the head atoms in r and r' are identical, these rules, when fired, define the same "set" of ground atoms. To show that at iteration $i + 1$, these rules define the same results, considered as multisets, let ρ be any ground instance of r, and v_i be the valuation of P defined at iteration i, which maps every subgoal in the body of ρ to $\mathcal{T}$. Suppose θ is the substitution used to define ρ. Then, the certainty associated with $p(\overline{X})\theta$ defined by r' is $g(\top, g(v_i(q_1(\overline{Y}_1)\theta), \ldots, v_i(q_n(\overline{Y}_n)\theta)))$, which is the same as the certainty of $p(\overline{X})\theta$ defined by r upon noting that $V_j = v_i(q_n(\overline{Y}_j)\theta)$, for $1 \leq j \leq n$, and $g(\top, \alpha) = \alpha$, by Postulate 10 on the propagation functions (Definition 2.1.1). Finally, the certainty γ associated with A at iteration $i + 1$ defined by P is obtained by collecting "all" the derivations of A as a multiset over which we apply the disjunction function f_d. Clearly, γ is also what we obtain at iteration $i + 1$ in the evaluation of P'. The result then follows as A was arbitrary. ■

The key idea in the proof above is that under the given conditions, the annotation variables in the body of every rule r are just place holders for the actual certainties associated with the subgoals of r used when evaluating r. In particular, no relationship between the annotations in r is asserted under the conditions specified.

Our next result establishes that the EP framework and the extended GAB framework have the same expressive power.

Theorem 5.2.2 *For any program in the extended GAB, there exists an ep-program such that on every EDB, they compute the same atom-certainty pairs. Conversely, given any ep-program, there exists a GAB program with certainty constraints such that on every EDB, they compute the same atom-certainty pairs.*

Proof. The proof idea is based on program transformation. We will show that any rule in either of these frameworks can be "equivalently" expressed as a rule in the other, in the sense that on every database, including the EDB as well as the IDB, the rule and the transformed rule define the same *multiset* of atom-certainty pairs. The transformation method is straightforward and is described as follows. Suppose r is any rule in the extended GAB framework. Then, r is of the form:

$$r : p(\overline{X}) : g(V_1, \ldots, V_n) \leftarrow q(\overline{Y}_1) : V_1, \ldots, q_n(\overline{Y}_n) : V_n,\ Constraint_r.$$

In this case, r can be expressed as the following ep-rule:

$$r' :\ \ p(\overline{X}) \xleftarrow{\top} q_1(\overline{Y}_1), \ldots, q_n(\overline{Y}_n),\ Constraint'_r;\ \langle f_d, -, g \rangle$$

where $Constraint'_r$ is obtained from $Constraint_r$ with annotation variable V_j replaced by $wt(q_j(\overline{Y}_j))$, for $1 \leq j \leq n$. Here, f_d is the disjunction function associated with the head predicate p, and g is the conjunction function specified in the annotation function in the rule head. Since the certainty associated with r' is $\top$, it follows from Postulate 10 that it does not matter what propagation function is used in r'. This completes the transformation process which defines the ep-program.

We now show how a given program in the EP framework can be transformed to a program in the extended GAB. Consider an arbitrary ep-rule s, which is of the form:

$$s :\ p(\overline{X}) \xleftarrow{\alpha_s} q_1(\overline{Y}_1), \ldots, q_n(\overline{Y}_n),\ Constraint_s;\ \langle f_d, f_p, f_c \rangle.$$

This rule can be expressed in the extended GAB framework as:

$$s' :\ \ p(\overline{X}) : f_p(\alpha_s, f_c(V_1, \ldots, V_n)) \leftarrow q_1(\overline{Y}_1) : V_1, \ldots, q_n(\overline{Y}_n) : V_n,\ Constraint'_s$$

where $Constraint'_s$ is obtained from $Constraint_s$ with the weight term $wt(q_k(\overline{Y}_k))$ replaced by V_k, for $1 \leq k \leq n$.

It can be easily verified that given any rule in one framework, the transformation method produces an "equivalent" rule in the other framework. It is then straightforward to see that this equivalence at the rule level implies equivalence at the program

level, since, by the transformation method, the disjunction function associated with the head predicates in the corresponding rules in the given program and the transformed program are identical. ■

The following example illustrates the transformation process.

Example 5.2.3 Let P be the following GAB program with certainty constraints.

$$\begin{aligned} r_1: \quad & disease(X,D) : 0.8 * min(V_1, V_2) \leftarrow has(X,S) : V_1,\ symptom(D,S) : V_2, \\ & \qquad V_1 \geq 0.8,\ V_2 > V_1. \\ r_2: \quad & disease(X,D) : 0.9 * min(V_1, V_2) \leftarrow family\_history(X,D) : V_1, \\ & \qquad hereditary(D) : V_2, \\ & \qquad V_1 \geq 0.8,\ V_2 \geq 0.7. \end{aligned}$$

Following the transformation method proposed above, P would be transformed into the following ep-program, assuming that in P, f_d is the disjunction function associated with the predicate *disease*.

$$\begin{aligned} r_1: \quad & disease(X,D) \xleftarrow{0.8} has(X,S),\ symptom(D,S),\ wt(has(X,S)) \geq 0.8, \\ & \qquad wt(symptom(D,S)) > wt(has(X,S));\ \langle f_d, *, min \rangle. \\ r_2: \quad & disease(X,D) \xleftarrow{0.9} family\_history(X,D),\ hereditary(D), \\ & \qquad wt(family\_history(X,D)) \geq 0.8, \\ & \qquad wt(hereditary(D)) \geq 0.7;\ \langle f_d, *, min \rangle. \end{aligned}$$

■

Note that although query processing in the AB framework of Kifer and Subrahmanian [KS92] may in general call for constraint solvers, explicit assertions of relationship among certainty constraints are not allowed in user programs; a restricted form of constraints is implicitly allowed in the basic AB framework, done through associating the same annotation constant/variable with more than one atom in a rule body. Incorporating certainty constraints in the parametric framework or any AB framework strictly increases the expressive power of the framework. Another advantage of allowing certainty constraints in AB frameworks is to providing a common ground so that the comparison between the EP framework and the extended GAB framework would be possible. This sheds more light on the relationships of the AB and IB approaches to uncertainty in logic programming and deductive databases. The above result establishes that when constraints are added, the EP and the extended GAB frameworks are equivalent in terms of expressive power.

5.3 Continuity and Expressiveness Trade Off

The increased expressive power gained by allowing certainty constraints comes at the expense of continuity in the sense that the fixpoint operator T_P would no longer be continuous. The following example illustrates this point.

Example 5.3.1 Consider the ep-program P below, in which the underlying certainty domain is $\mathcal{T} = [0, 1]$.

$$r_1: \ p(X,Y) \xleftarrow{1} e(X,Y); \qquad \langle ind, *, min \rangle.$$
$$r_2: \ p(X,Y) \xleftarrow{1} e(X,Z),\ p(Z,Y); \qquad \langle ind, *, min \rangle.$$
$$r_3: \ r(X,Y) \xleftarrow{1} p(X,Y),\ q(X,Y), wt(p(X,Y)) \geq wt(q(X,Y)); \qquad \langle ind, *, min \rangle.$$

Suppose $D = \{e(1,1) : 0.5,\ e(1,2) : 0.5,\ q(1,2) : 1\}$ *is the EDB. Since* $T_P^{\omega}(r(1,2)) = 0$ *but* $T_P^{\omega+1}(r(1,2)) = 1$, *the fixpoint of* T_P *will not be obtained at* ω, *implying that* T_P *is not continuous.* ■

Another impact of p-programs with constraints is on the termination behavior and complexity of evaluating such programs. We note that there are ep-programs with constraints whose evaluations terminate, while the corresponding p-programs with the constraints dropped may not terminate. For instance, consider a p-program with the p-rules r_1 and r_2 defined in the above example. The fixpoint evaluation of this program on the EDB D does not terminate. Now consider the ep-program $Q = \{r_1, r_2'\}$, where r_2' is the following ep-rule.

$$r_2': \ p(X,Y) \xleftarrow{1} e(X,Z),\ p(Z,Y),\ wt(e(X,Y)) \geq 1,\ wt(p(X,Y)) \geq 1: \ \langle ind, *, min \rangle.$$

It is easy to see that the fixpoint evaluation of Q on D is obtained in two iterations. This is because the first certainty constraint in r_2' is such that it ensures this rule never fires.

5.4 Summary and Concluding Remarks

In Chapter 1, we classified the approaches to uncertainty in logic programming and deductive databases into two – Annotation Based (AB) and Implication Based (IB).

Although for some reasons mentioned in that chapter, we chose the IB approach and developed a powerful framework which unifies and generalizes the IB approach, the relationship between the two approaches was a question which we desired to investigate. The main reason for our quest was because there was a generalized theory of annotated logic programming proposed by Kifer and Subrahmanian [KS92], to which we wanted to compare our parametric framework in terms of expressive power. Our study in this chapter shed some light on this issue and provided insights. We found out that the notion of certainty constraints is a key concept bridging these two approaches. We explained that the annotation constants and variables as introduced in the AB approach provided a restricted form of certainty constraints. Generalizing this, we then showed that allowing certainty constraints in both approaches strictly increases the expressive power of both. Furthermore, we established that the resulting frameworks (the EP and the extended GAB) have the same expressive power.

As illustrated by Example 5.3.1, the increased expressive power obtained by incorporating certainty constraints comes at a price; the immediate consequence operator, T_P, is not continuous any more. On the other hand, we showed that this incorporation is useful in programming with uncertainty, since it lets realizing the operations of selection and join by certainty, as described earlier in this chapter. A question at this point is that: should one go for constraints to benefit from the increased expressive power? If yes, how to deal with the discontinuity?

We remark that not every use of certainty constraints would be a problem w.r.t. discontinuity of T_P. For instance, if a program in the EP or extended GAB framework uses certainty constraints of the form $wt(A)\ \theta\ wt(B)$ only, then the fixpoint semantics would be computed in at most ω steps. The same is true when every certainty constraint in a program is of the form $wt(A) = \sigma$, where $\sigma \in \mathcal{T}$ is a certainty value. The discontinuity may be an issue when the program contains both kinds of certainty constraints. A similar remark by Kifer and Subrahmanian in the context of the GAP framework [KS92] is that evaluating a GAP program may exhibit undesirable behavior when annotation constants and variables are intermixed in the programs.

It is interesting to note that certainty constraints in which the relationship θ could be $=$, $\neq$, $\prec$, or $\preceq$ is related to negation. The following example illustrates that in this case, monotonicity may be lost.

Example 5.4.1 Consider the following example of an ep-program.

r_1 : $A \xleftarrow{0.5}$.

r_2 : $A \xleftarrow{0.5} A,\ wt(A) = 0.5.$

Suppose $\langle ind, min, min \rangle$ is the triple of combination functions associated with r_1 and r_2. A fixpoint evaluation of this program is as follows. At iteration 1, we derive A with certainty 0.5. In the second iteration, there are two derivations of A each of which is of certainty 0.5. The overall certainty of A at this iteration is thus $ind(0.5, 0.5) = 0.75$. In the third step, only r_1 can be fired and hence we derive A with certainty 0.5, which is less than A's certainty in the previous step. The euality relation "=" in rule r_2 of this example seems to play the role of negation in the standard framework. Similar remarks hold for the other relations mentioned above as θ. This issue worths further investigation.

Chapter 6

Implementations

In Chapter 3, we studied the containment of parametric conjunctive queries which is at the heart of query optimization. The results obtained would help decide, for instance, when a subgoal can be removed from the body of a p-rule in a given p-program without changing the semantics of the program. This could result in a significant increase in the efficiency of an evaluation of the program.

Efficient query processing in standard logic programming and deductive databases has been studied extensively for over a decade, and numerous techniques have been proposed and implemented in many existing systems. See [CGT89] for an overview of these techniques. It is well-known that alternate derivations of the same atom are not important in the standard framework. One source of inefficiency in the bottom-up naive evaluation of datalog programs is that some atoms might be derived multiple times during the iteration process. More precisely, atoms derived at some iteration will continue to be derived at all the subsequent iterations. The conventional semi-naive evaluation methods proposed improve the situation by partially eliminating such inefficiencies, and hence such methods are preferred over the naive method.

However, as shown before, alternate derivations may not be ignored in general when uncertainty is present. What we need to do for evaluating queries in this case is to collect "all" the answer tuples as a *multiset* to which we then apply a desired disjunction function. We propose a semi-naive method suitable in our context obtained by extending the idea from the corresponding method in the classical case. It also takes into account the certainties mentioned in a p-program as well as the

combination functions used.

The rest of this chapter is organized as follows. In the following section we develop a semi-naive method for evaluating programs with uncertainty. This method is used in both of our top-down (TD) and bottom-up (BU) implementations of the parametric framework which are introduced in Section 6.2. Finally, in Section 6.3, we give examples demonstrating how query programs in the parametric framework would be evaluated using our TD and BU implementations. Our experience in various implementations together with experimenting with the TD and BU systems increase our belief that the ideas in this thesis lend themselves to an efficient and easy-to-use environment for deduction with uncertain knowledge.

6.1 A Semi-Naive Evaluation Method

In this section, we study efficient bottom-up evaluations of p-programs. Let us quickly describe the bottom-up naive evaluations of p-programs. Initially, every atom is assigned the least certainty value, $\perp \in \mathcal{T}$. Then, at iteration i, we apply *every* rule in which every subgoal in the rule body has a certainty greater than $\perp$, and plug for each subgoal, its "best" certainty derived in the previous iteration. Then, alternate derivations of the same atom are combined into a single certainty, using the desired disjunction function. This process terminates at some iteration, possibly ω, at which no atom is derived with a "better" certainty.

The naive method just described is a straightforward extension of the conventional naive method, which uses the disjunction functions defined in the program for aggregating at each iteration the certainties obtained. Following the same idea, we develop a semi-naive method for evaluating programs with uncertainty, described as follows. At each iteration, we maintain a multiset of certainties derived so far for each atom, as well as the best combined certainty obtained for it in that iteration. At iteration 0, the multiset is empty for all atoms, and the best certainty is $\perp$. (In the classical case, every ground atom is initially associated with $false$). At iteration $i+1$, we apply every rule for which there is at least some subgoal whose best certainty changed at iteration i. (In the classical case, we fire every rule for which we found a proof for a subgoal in the rule body at the previous iteration.) The certainty used for

each subgoal in a rule applied is the best certainty of that subgoal from iteration i, obtained from its associated multiset. For each atom A derived at iteration $i+1$, the multiset of certainties obtained from all possible derivations up to this iteration are collected and combined using the disjunction function associated with the predicate of A. If no atom had its best certainty changed from iteration i to $i+1$, then the evaluation terminates. (In the classical case, this means if no new atom is derived from iteration i to $i+1$, then the evaluation stops.)

The above steps are formalized in the algorithm shown in Figure 1. Recall that $\dot{\emptyset}$ is the empty multiset, and $\dot{\cup}$ is the multiset union which retains the duplicates. For every atom A at iteration i, we associate a pair, $M_i(A)$ and $\sigma_i(A)$, where $M_i(A)$ is a multiset which contains "all" certainties obtained for A up to and including iteration i from all possible derivations of A, and $\sigma_i(A) = f_d(M_i(A))$, where f_d is the disjunction function associated with the predicate of A. In this algorithm, we use New_i to denote the set of all atoms derived at iteration i ($i > 0$) with a better certainty than before; $New_0 = \emptyset$. In the classical case, New_i would contain every atom derived for the first time at iteration i. We also use v_i to denote the valuation of the program obtained at iteration i, where $v_0 = v_\perp$ is the least valuation. Recall that P^* is the set of ground instances of all the p-rules in P. The following example illustrates the steps of this algorithm.

Example 6.1.1 Consider P be the following p-program.

$$r_1: \quad p(X,Y) \xleftarrow{1} e(X,Y): \qquad \langle ind, \_, \_ \rangle.$$
$$r_2: \quad p(X,Y) \xleftarrow{1} e(X,Z),\ p(Z,Y): \qquad \langle ind, \_, * \rangle.$$

Any propagation and conjunction function can be used in r_1 and r_2 where there is a don't care. Let D be the EDB which includes the tuples $e(0,1)$, $e(0,2)$, $e(1,2)$, $e(1,3)$, and $e(3,2)$, each of which is associated with certainty 0.5. Let us now consider the bottom-up semi-naive evaluation of P on D, for which we show the multiset-certainty pair $\langle M, \sigma \rangle$ associated with each atom, whenever the pair is updated in an iteration. At iteration 0, every ground atom in the Herbrand base of $P \cup D$ is associated with the pair $\langle \dot{\emptyset}, \perp \rangle$. At iteration 1, the pair associated with every atom in D is $\langle \{\!\{0.5\}\!\}, 0.5 \rangle$, and $New_1 = D$. We can apply r_1 at iteration 2 and derive $p(0,1)$, $p(0,2)$, $p(1,2)$, $p(1,3)$, and $p(3,2)$, and the pair associated with each of these atoms is $\langle \{\!\{0.5\}\!\}, 0.5 \rangle$. In

procedure SemiNaive (P, D; $lfp(T_{P\cup D})$)

$\forall A \in B_P$ **begin** $M_0(A) := \dot{\emptyset}$; $\sigma_0(A) := \perp$; **end**;

$\forall (A : \alpha) \in D$ **begin** $M_1(A) := \{\!\{\alpha\}\!\}$; $\sigma_1(A) := \alpha$; **end**;

$i := 1$; $v_1 := D$; $New_1 := \{A \mid A : \alpha \in D\}$;

repeat

$\forall A \in B_P$ **do**

begin

$M_{i+1}(A) := M_i(A)$;

if $\exists (A \xleftarrow{\alpha_r} B_1, \ldots, B_n;\ \langle f^d, f^p, f^c\rangle) \in P^*$

such that $B_j \in New_i$, for some $j \in \{1, \ldots, n\}$,

then begin

$\forall (A \xleftarrow{\alpha_r} B_1, \ldots, B_n;\ \langle f^d, f^p, f^c\rangle) \in P^*$

such that $B_j \in New_i$, for some $j \in \{1, \ldots, n\}$, **do**

begin

$M_{i+1}(A) := M_{i+1}(A) \dot{\cup} \{\!\{\sigma\}\!\}$,

where $\sigma := f^p(\alpha_r, f^c(\{\!\{v_i(B_1), \ldots, v_i(B_n)\}\!\}))$;

end;

$\sigma_{i+1}(A) := f_d(M_{i+1}(A))$;

$v_{i+1}(A) := \sigma_{i+1}(A)$;

end;

else $v_{i+1}(A) := v_i(A)$;

end forall;

$New_{i+1} := \{A \mid A \in B_P, \sigma_{i+1}(A) \succ \sigma_i(A)\}$;

$i := i + 1$;

until $New_i = \emptyset$;

$lfp(T_{P\cup D}) := v_i$;

end

Figure 1: A semi-naive evaluation algorithm for p-programs

this case, New_2 includes only these p-atoms. In the third iteration, we can only apply r_2 by which we derive $p(0,2)$. $p(0,3)$, and $p(1,2)$, each of which is associated with the pair $\langle\{\!|0.5, 0.25|\!\}, 0.625\rangle$. We thus have $New_3 = \{p(0,2),\ p(0,3),\ p(1,2)\}$. At iteration 4, we can apply rule r_2 and derive only $p(0,2)$ using $p(1,2)$ and $e(0,1)$. In this case, $\langle\{\!|0.5, 0.25, 0.3125|\!\}, 0.65625\rangle$ is the pair associated with $p(0,2)$, and $New_4 = \{p(0,2)\}$. The evaluation process proceeds to the next iteration, in which no rule can be applied, and hence terminates. The atom-certainty pairs we get in v_5 are those in D as well as $p(0,1)$:0.5, $p(1,3)$: 0.5, and $p(3,2)$: 05, $p(0,3)$: 0.625, $p(1,2)$: 0.625, and $p(0,2)$: 0.65625. ■

The following result shows that the semi-naive algorithm described above produces the same atom-certainty pairs as the naive method.

Theorem 6.1.2 *For every p-program P and every EDB D, the bottom-up naive and semi-naive evaluations are equivalent.*

Proof. The proof follows from the definitions of the two methods upon noting that at every iteration i, under both methods, we combine exactly the multiset of all certainties for a given atom A coming from all derivations considered so far. The only difference between the two methods is that under the semi-naive evaluation proposed above, at iteration $i+1$, we do not recreate derivations previously considered, since their results are "cached" in the multiset $M_i(A)$ associated with A. ■

Before closing this section, let us point out a similarity we observe between the role of disjunction functions in programs and the role of function symbols in logic programming framework, although function symbols in the conventional sense are not allowed in the parametric framework. Since in a p-program P, there are only finitely many certainty values, we may conclude that the non-termination behavior or termination in an arbitrarily large number of iterations discussed in Chapter 4 is due to *generation* of new certainties derived by disjunction functions in P, such as *ind* in the above example. In a logic program Q with function symbols, such a function *generates* infinitely many ground atoms in the Herbrand base of Q. The bottom-up evaluations of both P and Q may terminate only at ω. The similarity mentioned above between the two frameworks is in the reason for having non-terminating evaluations, which is "generation of new things" at every iteration, where the new things in the

case of program P are atoms derived with better certainties, and in the case of Q, they are atoms derived with new functional terms. Note also a difference between the two cases from the viewpoint of the output size, which is finite in the case of P, and infinite in the case of Q.

6.2 Implementation Modules

In this section, we introduce our *top-down* and *bottom-up* implementations of the parametric framework. We will use TD to refer to the top-down implementation, and BU to the latter one. For convenience, we will use *system* to refer to either one. The TD system consists of four modules, described below. Since there are parallels between the two systems, we will describe all the four kinds of modules in the TD system and the inference engine module of the BU system. Since our top-down implementation TD is on top of the XSB system [SSW94], we will use the XSB notations and formalism to describe TD. A quick remark about XSB itself is that it is a powerful logic programming system based on PSB-PROLOG [Xu 89] which itself is based on version 2.0 of SB-PROLOG [Deb88]. In addition to providing all the functionality of full PROLOG, XSB has several interesting features discussed below which we used in TD.

At a very high level, TD is a parametric "programming environment" in which a user can set the parameters to define a desired IB framework for evaluating p-programs according to the semantics of that framework defined by the parameters. For instance, if P is a p-program in van Emden's language [van86], and D is an EDB on which we want to evaluate P, then we can easily set the parameters in TD to set the environment so that the atom-certainty pairs computed conforms to the semantics proposed in [van86]. The system has the following four modules, which we call *system modules.*

- Inference Engine (IE)
- Parameters' Definitions (PD)
- User Interface (UI)
- User Program (UP)

A general description of the system modules is as follows. Each module is a logic programs written in the underlying database language, that is, XSB for the TD system, and CORAL for the BU. The IE module is the heart of the system, responsible for interpreting user programs in a UP module. We have taken a meta-level programming approach to develop IE and UI modules. A PD module contains definitions of the parameters (that is, the certainty lattice, top and bottom elements, and the combination functions) of an IB framework. Thus, PD is in fact a collection of modules, one for each IB framework. By consulting a PD module at the TD prompt, a user effectively instructs the system to evaluate a p-program in a UP module according to the parameters defined in the PD module. In fact, this is how TD allows a user to set the parameters for defining a desired environment so that the computation conforms to the semantics captured in the consulted PD. The UI module is a collection of rules providing the users with some high-level operations that make the programming environment user friendly and efficient.

A detailed description of the system modules is as follows, presented in the reverse order from UP to IE, for ease of presentation.

6.2.1 User Program

The *user program module* (UP, for short) is collection of non-ground facts representing the rules and facts in a user program, to be evaluated by the system. This module contains two kinds of atoms, `rule(R,H,B,C)` and `fact(F,H,C)`, where `R`, `H`, `B`, `C`, and `F` denote, respectively, the Rule number, the rule Head, the rule Body, the rule Certainty, and the Fact number. The meanings of these atoms should be clear. The rule body `B` is a list of atoms in `[...]`. Each rule and fact in a user program has an id, `R` and `F`, used for two purposes. The first is to facilitate asserting what combination function is associated with what rule. For instance,

```
:- assert(conj_mode(r2,product)).
```

in a UP module says the conjunction function associated with rule `r2` is $*$. The second use of a rule-id is to facilitate tracing, explained later in the UI module.

As can be seen in Figure 2, a UP module *may* also contain the following (non-ground) atoms.

```
:- assert(rule(r1,p(X,Y),[e(X,Y)], 1)).
:- assert(rule(r2,p(X,Y),[e(X,Z),p(Z,Y)], 1)).
:- assert(fact(1,e(0,2),0.5)).
:- assert(fact(2,e(0,1),0.5)).
:- assert(fact(3,e(1,2),0.5)).
:- assert(fact(4,e(1,3),0.5)).
:- assert(fact(5,e(3,2),0.5)).
:- assert(conj_mode(r1,_)).
:- assert(conj_mode(r2,product)).
:- assert(prop_mode(_Rule,_)).
:- assert(disj_mode(e,max)).
:- assert(disj_mode(p,ind)).
```

Figure 2: An instance of a user program module in TD

```
disj_mode(p, dmode).
conj_mode(r, cmode).
prop_mode(r, pmode).
```

The first assertion means that the disjunction function associated with the predicate `p` is `dmode`. The second (third) assertion means that the conjunction (propagation) function used in rule `r` is `cmode` (`pmode`). In some frameworks, e.g., [van86], [DLP91], and [LS94b], there is only one combination function allowed of each kind. In this case, a user need not include the above atoms in the UP module; the system will find the default combination functions defined in the PD module. (See Figure 4, for instance). As an instance of UP module, consider the p-program of Example 6.1.1 and the given EDB D, assuming *max* is the disjunction function associated with the EDB predicate e. This could be represented as the UP module shown in Figure 2.

6.2.2 User Interface

The user interface module (UI, for short) is a collection of definitions of high-level programming predicates available to users to interact with the system in a more convenient and efficient way. A list of these predicates is shown in Figure 3. In fact, the list itself is the result of applying such a predicate, that is, `help`.

Each command in UI is defined by a collection of rules. Let us explain some of

```
Command        Description
===========    ====================================
help           get this menu;
[P]            consult module P;
eval           compute all atom-certainty pairs;
eval(A,C)      compute the certainty C of atom A;
filter(A,C,D)  compute atoms with certainty at least D;
purge          remove the user program;
rmf            remove the facts in user program;
del(N)         delete rule/fact no. N in user program;
reset          clear the current work space;
list           list the rules/facts in the user program;
combos         list the combination functions defined;
all            list the rules/facts/combos defined;
explain        show the deduction sequence;
clear          clear the screen;
halt/quit      exit the system;
click exit     exit, if running in window.
```

Figure 3: High-level commands available to users in TD

these commands. A user can use the command `Del(N)` at the system prompt to delete a fact or a rule whose id is `N`. Using `rmf`, a user can remove the EDB facts. This is useful, for instance, when the user wants to evaluate the same program on different EDBs. This is done, of course, with no side effects from the previous computations. Using the command `purge`, a user can remove all the rules, facts, and assertions of the various combination functions in the UP module consulted. This allows a user to evaluate various programs in the same framework defined, without leaving the run-time system. In particular, this command does not affect the IE and PD modules consulted. What a user can do after a purge command is to consult another UP module, say `P`, through the command `[P]`, and evaluate it.

The UI module developed also allows a user to evaluate the same program according to more than one semantics, whenever meaningful, without having to leave the run-time environment. This means, removing the PD and UP modules from the run-time system, while the IE and UI modules continue to exist. This is done using the `reset` command, which "resets" the environment. The user can then create a new environment by consulting the PD module defining a desired framework.

```
:- assert(bottom(0)).
:- assert(top(1)).
:- assert(disj_mode(_Predicate,max)).
:- assert(prop_mode(_RuleNo,product)).
:- assert(conj_mode(_RuleNo,min)).
:- hilog max. max(X,Y,Z) :- X<Y -> Z=Y; Z=X.
conjOp(min,CertA,CertB,MinAB) :- min(CertA,CertB,MinAB).
propOp(product,CertA,CertB,CertAB) :- CertAB is CertA * CertB.
```

Figure 4: An instance PD module defining the framework in [van86]

The command `explain` lists the sequence of rules and facts applied to answer the query processed. We will illustrate this through an example in Section 6.3. The command `filter(V)` instructs the system to discard on the fly those atoms derived with certainties less than a threshold value `V` supplied as the argument. This lets the system prune useless search graphs, whenever desired, determined by the user. This in general results in a more efficient computation and should be used only when the disjunction function associated with the recursive predicates in the program coincides with the lattice join.

6.2.3 Parameters Definitions

The *parameters' definitions module* (PD. for short) is a collection of assertions. rules and facts. defining a particular IB framework. In fact. PD is not one module. but a collection of modules. each of which contains definitions of the parameters relevant to an IB framework. This includes the certainty lattice, the top and bottom elements. and the definitions of disjunction, propagation, and conjunction functions allowed in the framework. Note that a UP module uses combination functions, while the PD module defines them. To give an idea, Figure 4 shows the content of this module for the van Emden framework [van86], whose underlying certainty lattice is $\langle [0,1], min, max \rangle$, and which uses max as the disjunction, $*$ as the propagation, and min as the conjunction function.

In XSB, HiLog [CKW93] and tables can be used to support aggregation. The HiLog rule:

```
:- hilog max. max(X,Y,Z) :- X<Y -> Z=Y; Z=X.
```

used in a PD module defines `max` as a disjunction function, in our terminology. More precisely, `max` is the name of a multiset to which the system applies the aggregation operation `max`.

As a more elaborate example, consider a subset of rules in a PD module defining the parameters of the probabilistic framework of Lakshmanan and Sadri [LS94a] shown in Figure 5. This module contains 32 rules in about 200 lines of XSB code. The first pair of assertions in this figure define the top and bottom elements of the certainty lattice used in this framework. A list `[V1,V2,V3,V4]` of values in $[0, 1]$ represents the pair of intervals as in the confidence level `<[V1,V2],[V3,V4]>`. Since this framework does not consider propagation functions, we define in the first rule in the figure that the propagation function in a rule to be the same as the conjunction function associated with that rule. The figure also includes definitions of two conjunction functions, in the *positive correlation (pc)* and *independence (ind)* modes. The last two rules are HiLog rules defining the disjunction functions in the *pc* and *ind* modes.

6.2.4 Inference Engine of the TD System

The *inference engine module* (IE, for short) is the heart of the system. It is responsible for evaluating user programs, defined in a UP module, according to a desired semantics, defined in the PD module. As in the UI module in TD, we have used a meta-level programming technique to implement the IE module in XSB, which essentially takes into account the various parameters defined in the PD modules for evaluating a program in a UP module.

XSB includes several general predicates which allow a programmer to collect "all" solutions to a query as a multiset (or a list, to be more precise) and then computes a desired (disjunction) function over the multiset. This is useful in our context to perform aggregations using disjunction functions. This is done by using the general predicate `bagReduce/4`, as follows.

```
bagReduce(interpMgoal(Atom),CertaintyAtom,DisjOp,Bottom).
```

This predicate takes a multiset name, `Atom`, an operator, `DisOp`, the identity of the operator, `Bottom`, and uses `DisjOp` to combine the elements in the multiset. The result is returned as the second argument, `CertaintyAtom`. Intuitively, this works as

```
:- assert(top([1,1,0,0])).
:- assert(bottom([0,0,1,1])).
propOp(Mode,X,Y,Z) :- conjOp(Mode,X,Y,Z).
conjOp(pc,[V0,V1,V2,V3],[V4,V5,V6,V7],[V8,V9,V10,V11]) :-
    !,
    min(V0,V4,V8),
    min(V1,V5,V9),
    max(V2,V6,V10),
    max(V3,V7,V11).
conjOp(ind,[V0,V1,V2,V3],[V4,V5,V6,V7],[V8,V9,V10,V11]) :-
    !,
    V8 is V0 * V4,
    V9 is V1 * V5,
    V10 is V2 + V6 - V2 * V6,
    V11 is V3 + V7 - V3 * V7.
:- hilog pc.
pc([V0,V1,V2,V3],[V4,V5,V6,V7],[V8,V9,V10,V11]) :-
    max(V0,V4,V8),
    max(V1,V5,V9),
    min(V2,V6,V10),
    min(V3,V7,V11).
:- hilog ind.
ind([V0,V1,V2,V3],[V4,V5,V6,V7],[V8,V9,V10,V11]) :-
    V8 is V0 + V4 - V0 * V4,
    V9 is V1 + V5 - V1 * V5,
    V10 is V2 * V6,
    V11 is V3 * V7.
```

Figure 5: Part of the PD module defining the framework in [LS94a]

follows. It finds the first element of the multiset, applies the operator to the identity and that element and stores the result in the table, and continues this way: for each new element the operator is applied to the current value and the new element and the result replaces the current value. The final value in the table is returned as the result [War96].

The HiLog predicate `interpMgoal(Atom)` is responsible for deriving atom-certainty pairs, which are then fed into the multiset `bagReduce`. As mentioned earlier, aggregation operation on this multiset is done on the fly as a new certainty arrives. The following rule in the IE module shows how this predicate is used.

```
derive(Atom,CertaintyAtom) :-
    functor(Atom,Predicate,_Arity),
    disj_mode(Predicate,DisjOp),
    bottom(Bottom),
    bagReduce(interpMgoal(Atom),CertaintyAtom,DisjOp,Bottom).
```

Processing a query posed to the TD system begins by invoking the above rule, which on a goal G looks for every atom which unifies with G and derives its associated certainty. The first predicate in the rule body determines the head predicate, which would be the name of the multiset, and the second one determines the disjunction function associated with this predicate. Then, to obtain all derivations of this goal and its associated certainties, we invoke another predicate, `interpMgoal(Atom)`, responsible for deriving `Atom`. The following rules in IE define this latter predicate. This rule invokes the predicate `interpM(Body,CertaintyBody,ConjOp)` which computes the certainty of the conjunction `Body` of atoms in a rule body, presented as a list. The meanings of other predicates should be clear.

```
interpMgoal(Atom)(CertaintyAtom) :-
    fact(FactNo,Atom,CertaintyAtom).
interpMgoal(Atom)(CertaintyAtom) :-
    rule(RuleNo,Atom,Body,CertaintyRule),
    conj_mode(RuleNo,ConjOp),
    interpM(Body,CertaintyBody,ConjOp),
    prop_mode(RuleNo,PropOp),
    propOp(PropOp,CertaintyRule,CertaintyBody,CertaintyAtom).
```

The predicate `interpM` "interprets" a rule body by computing its certainty. This is done by combining the certainties of all atoms in the body into a single certainty, using the conjunction function `ConjOp` supplied. In TD system, we have done this by the following rules. The first rule defines the certainty of an empty conjunct to be the top certainty element (Postulate 9).

```
interpM([],Certainty,_ConjOp) :- !, top(Certainty).
interpM([Atom|Atoms],Certainty,ConjOp) :-
```

```
derive(Atom,CertaintyAtom),
interpM(Atoms,CertaintyAtoms,ConjOp),
conjOp(ConjOp,CertaintyAtom,CertaintyAtoms,Certainty).
```

The collection of the above rules forms the core of the IE module of the TD system, which has about 50 rules in over 200 lines of XSB code.

6.2.5 Inference Engine of the BU System

In this section, we introduce the inference engine of BU, the bottom-up implementation of the parametric framework on top of the CORAL database language [RSS92]. A few words on CORAL; it is a powerful system with many attractive features such as *annotations* through which, a user can "influence" the run-time environment. In particular, the annotations `@multiset` and `@aggregation\_selection` are needed in our context, where the former allows collecting the derivations as multisets, and the latter allows aggregation per iteration, but only for *max*, *min*, and *any*[1].

Our implementation project started with developing, in CORAL, an interpreter for programs in the probabilistic framework in [LS94a]. In the course of that project, we noticed that CORAL has to be extended to support user-defined aggregations, which we will get back shortly. We remark that any IB framework which uses (fuzzy) sets as the basis for the structure of the semantics could be implemented in the existing CORAL efficiently by taking advantage of the aggregation_selection *max*, which instructs CORAL to discard the unwanted tuples on the fly in a bottom-up evaluation. In fact, we have developed such an efficient implementation of the IB framework proposed by van Emden [van86].

We have obtained an extended running of CORAL which supports user-defined aggregations per iterations, thanks to Dr. Raghu Ramakrishnan at Wisconsin university for supporting the idea and to Shaun Flisakowski for the design and implementing it. Aside from formulating the problem, we contributed to the design of the module supporting user-defined aggregations, and to the testing of the system. We believe this increased capability of CORAL is important for deductive databases in the more general context of fixpoint computation with arbitrary (user-defined) aggregations.

[1] *any* returns any one of its argument values, determined by an implementation of CORAL.

```
e(1,2,0.75).
e(2,3,0.8).

module test.
export p[bbf,fff].
@aggsel_per_iteration p(X,Y,Z) (X,Y) ind(Z).
@multiset +.

p(X,Y,Z) :- e(X,Y,Z1), min(1,Z1,Z).
p(X,Y,Z) :- e(X,M,Z1), p(M,Y,Z2), Z3=Z1*Z2, Z=0.5 * Z3.

min(V1,V2,V1) :- V1 <= V2.
min(V1,V2,V2) :- V1 >  V2.
end_module.
```

Figure 6: Example of a p-program in the BU system

To see how the BU system works, consider the p-program shown in Figure 2. This program could be represented in the BU system as a CORAL module shown in Figure 6. Note that in this module, the propagation and conjunction functions are "hard-coded", in the sense that the user program, part of the inference engine, and part of the definitions of the parameters are all defined in one module. This should be contrasted with the modules in the TD system, where we had a "clean" separation of responsibilities of the modules. The problem here is that assertion of non-ground facts is not supported any more in the extended CORAL, and this lack of support does not support meta-level programming. Therefore, the user has to produce the module shown in Figure 6 directly, or develop a translator to create this module automatically from something more convenient for the user to provide, such as the one given in Figure 2.

Annotation `@multiset` in this module instructs the run-time of CORAL to collect derivations of the atoms as a multiset, and

`@aggsel_per_iteration p(X,Y,Z) (X,Y) ind(Z).`

instructs CORAL that, for every p-atom in relation `p` having the same `X,Y` values, group as a multiset all values in the third argument, `Z`, over which it performs the aggregation using `ind`. Recall that $ind(X,Y) = X + Y - XY$ is the probabilistic independence mode. If `C` is the result of this function, then all tuples contributed to

```
module independence.
export ind[bbf].
ind([ [X,Y,OldCertain] | Rest], OldCertains, [[X,Y,ComboCertain]])
        :-  combine_certain(OldCertains, ComboCertain).
combine_certain([], 0.0).
combine_certain([[C1]], C1).
combine_certain([[C1],[C2]], Ans) :- Ans = C1+C2 - C1*C2.
combine_certain([[C1]|Rest], Ans) :-
            length(Rest,Len), Len >= 2,
            combine_certain(Rest,C2), Ans = C1+C2 - C1*C2.
end_module.
```

Figure 7: A module in BU exporting a user-defined disjunction function

the group will be replaced by one tuple, `p(X,Y,C)`, before the next iteration begins.

It remains to define the module exporting `ind`. This is shown in Figure 7. The first line in this module says that this module exports `ind[bbf]` to any module who calls it, where `b` stands for bound, and `f` for free. Thus, given two values, this function returns the combined value. Actually, every group of values that should be combined using this function are passed to it as a list. The assertion:

```
combine_certain([], 0.0).
```

in the module defines 0 as the certainty of the empty list: 0 is the identity for `ind`. The other rules in this module define how the certainty values in the list are combined. When the list has one element, the result returned is that element. For a list of two elements, `C1` and `C2`, the fourth rule in this module returns `C1+C2 - C1*C2`. When there are more than two elements in the list passed to this function, the last rule in the module is used to compute the combined value returned. Each application of this rule reduces the number of elements in the list by 1.

One final remark about the BU system. Our extended version of CORAL implements the semi-naive algorithm for evaluating query programs in the parametric framework. Although it is possible to extend CORAL so that its computation would correspond to the naive method, we take advantage of the BU system to compare its results with what TD computes. Based on our understanding of how BU and TD work, and also based on our experimenting in evaluating many query programs using the two systems, we believe that "*these systems produce the same results.*" No

formal arguments would be provided. Also, TD is easier and more convenient to use compared to BU, although using a BU implementation with meta-level programming would be as easy and convenient as the TD system.

6.3 Experimenting with the TD System

In this section, we demonstrate query evaluations using our implementations on some sample p-programs.

Consider the p-program shown in Figure 2. First, note that according to the semantics of this program defined through the parameters used, this program is not in a well-known IB framework. Recall that the parametric framework allows defining a new IB framework provided the combination functions satisfy the relevant postulates. Now, if we evaluate this program in the TD system, we get the following three atom-certainty pairs on the query `?- eval.` Note that `>>` is the system prompt.

```
>> ?- eval.
   p(1,3) : 0.5
   p(1,2) : 0.75
   p(2,3) : 0.8
   (Number of answers = 3)
```

Using the explain command, we get the following result showing the sequence of rules/facts the system used to process the query. The symbol * under the rule number means the result in that row was obtained by applying the aggregation operation. The repetition of some atom-certainty pairs indicates that the system derived the pairs multiple times, e.g., for proving different goals.

```
>> ?- explain.
  Sequence of the derivations:
  Rule#    Atom    Certainty
    3      e(1,2)      0.75
    4      e(2,3)      0.8
    *      e(2,3)      0.8
```

```
r1      p(2,3)      0.8
*       e(1,2)      0.75
r1      p(1,2)      0.75
*       e(2,3)      0.8
*       e(1,2)      0.75
4       e(2,3)      0.8
*       e(2,3)      0.8
r1      p(2,3)      0.8
*       e(2,3)      0.8
*       p(2,3)      0.8
r2      p(1,3)      0.5
*       p(1,3)      0.5
*       p(1,2)      0.75
*       p(2,3)      0.8
```

On query `?- eval(p(1,3),C)`, we get the following result.

```
>> ?- eval(p(1,3),C).
   p(1,3) : 0.5
   (Number of answers = 1)
```

Continuing with the session, suppose we now want to evaluate the p-program of Example 1.2.7 in the probabilistic framework proposed in [LS94a]. Also suppose this program is in UP module pdds_p2 and the definitions of the parameters are in DP pdds. Then, the following is the result of TD on the sequence of the queries `reset. [pdds,pdds_p2]. eval(b,C). eval(a,C). explain.` Note that the reset command clears the definitions of the environment, and [pdds] defines the environment so that programs, such as pdds_p2 in this framework can be evaluated.

```
>> ?- reset. [pdds,pdds_p2].  eval(b,C). eval(a,C). explain.
>> ?- [pdds loaded]
[pdds_p2 loaded]
>> ?-   b :  <[0.9,0.95], [0,0.15]>
  (Number of answers = 1)
```

```
>> ?-   a :  <[0.91,0.96], [0.01,0.04]>
  (Number of answers = 1)
>> ?-   Sequence of the derivations:
  Rule#    Atom          Certainty
    *       b        <[0.9,0.95], [0,0.15]>
    1       a        <[0.7,0.8], [0.1,0.2]>
    4       c        <[0.7,0.8], [0.1,0.2]>
    *       c        <[0.7,0.8], [0.1,0.2]>
    2       a        <[0.7,0.8], [0.1,0.2]>
    *       a        <[0.91,0.96], [0.01,0.04]>
```

Finally, consider the p-program of Example 4.6.1. Suppose `mine_p2` is the UP module containing this program and `mine` is the DP module containing the definitions of the parameters in `mine_p2`. Continuing from the previous execution, we get the following on query `eval(p(a),C)`. An explanation is also provided, as requested.

```
>> ?- reset. [mine,mine_p2]. eval(p(a),C). explain.
>> ?- [mine loaded]
[mine_p2 loaded]
>> ?-   p(a) : 1
  (Number of answers = 1)
>> ?-   Sequence of the derivations:
  Rule#    Atom    Certainty
    1       e(a)       1
    *       e(a)       1
    2       p(a)       0.8
    *       e(a)       1
    *       p(a)       0.8
    3       p(a)       0.8
    *       p(a)       0.96
    3       p(a)       0.96
    *       p(a)       0.9984
    3       p(a)       0.9984
    *       p(a)       1
```

6.4 Summary and Concluding Remarks

We proposed a semi-naive method for evaluating programs with uncertainty in the parametric framework. The proposed method extends the corresponding method in the standard framework, and was shown to be equivalent to the naive method. We presented our top-down and bottom-up implementations of the parametric framework, both of which implement the proposed semi-naive method. The top-down implementation is on top of the XSB system and the bottom-up implementation is on top of the CORAL system. These implementations together with our experimenting with them strengthen our belief that the ideas in this thesis lend themselves to an easy-to-use and efficient environment for deduction with uncertainty. Although we did not measure the efficiency of our implementations, our belief for them being efficient is on the basis of the fact that in both cases, the underlying XSB and CORAL systems use various advanced query processing/optimization techniques which would surely result in efficient computations.

In order to develop a bottom-up implementation of the parametric framework using a meta-programming approach, we need to extend CORAL to allow assertions of non-ground facts. Without this, the BU system requires many changes to a user program before it can be evaluated. With this extension, we would be able to use a meta-programming approach for the BU system so that a user will not be required to do much coding for preparing the input program, which in the current implementation of BU is integrated with the other modules, including the IE. This extension makes the BU system more convenient to use, as is the TD system. Also, we plan to enhance the user interface of our implementations. For this, we are working on developing a graphical user interface using the UIM/X system [Vis93]. This will help users create the program modules and evaluate them more readily while the unnecessary details of the implementations are hidden.

Chapter 7

Conclusion and Future Research

Research in *uncertainty* is essential, as we must learn not only to cope with data of limited reliability, but do so efficiently, with massive amounts of data [SSU91]. Our goal in this thesis was primarily to develop an *efficient* environment for *declarative* manipulation of *large* amount of *uncertain* knowledge. In this chapter, the work done towards this goal and the contributions of the thesis is summarized. We will also briefly discuss future research issues.

To reach our goal, we first studied the language aspect of the goal and proposed a powerful framework, called the *the parametric framework*, which unifies and generalizes the existing IB frameworks to uncertainty in logic programming and deductive databases. We observed that the existing frameworks differ in the underlying certainty lattice, and in the the conjunction, propagation, and disjunction functions allowed, which form our parameters. We identified a collection of reasonable properties which these combination functions should satisfy. Having done so, we then developed the semantics of the parametric framework based on *multisets*, as opposed to sets which are conventionally used. We defined the declarative, fixpoint, and proof-theoretic semantics of programs in our framework, and established the equivalence of these semantics.

The parametric language proposed forms a basis for us to study other related issues such as (1) query optimization, (2) termination and complexity of the bottom-up naive evaluation of p-programs, (3) expressive power of the language, and (4) its efficient implementation. Each of these issues was discussed in a chapter of this

thesis, in that order. As a byproduct of studying these aspects in the context of the parametric framework, the results obtained and the conclusions drawn are useful for a number of frameworks. This is an advantage of our "axiomatic" approach in studying all the above issues, making our results applicable to several frameworks.

Containment of conjunctive queries is central to query optimization in the traditional relational and deductive databases. In Chapter 3, we extended this idea to parametric conjunctive queries and developed various results, which yield tools for optimizing logic programs with uncertainty. An interesting direction for extending this work is on characterizing exactly when the "no teaming up" property exhibited by unions of classical conjunctive queries carries over to the parametric framework. Future work should address the central questions of containment and equivalence of programs. For instance, how can we lift the chase technique (e.g. see Sagiv [Sag88]) to IB frameworks? Can we characterize exactly when program equivalence in IB frameworks reduces to classical equivalence?

In terms of complexity of evaluating logic programs, it is well known that when function symbols are not allowed, this evaluation can be done in PTIME, in the size of database. In Chapter 4, we studied this in the presence of uncertainty. We showed that in this case the bottom-up fixpoint evaluation of p-programs (1) may terminate in PTIME, (2) may terminate in arbitrarily large number of iterations, independent of the EDB size, or (3) may not terminate at all. To our knowledge, case 2 has not been discussed any previous work. In our attempt to address the termination behaviors of p-programs, we illustrated that how sensitive the termination behavior becomes w.r.t. rule certainties or the presence/absence of certain "innocuous" looking rule, e.g., r_0 in Example 4.1.1. This state of affairs complicates a general analysis of termination for a substantial class of programs. One way to distill the termination behavior of a p-program from the rule certainties we found was to look at a *strong termination property*, namely at the question of whether the program terminates on all EDBs, regardless of the certainties associated with rules. With this as our approach, we classified the collection of disjunction functions allow in the parametric framework into three types — types 1 to 3. This in turn induces a classification of predicates into types 1 to 3; a predicate is of type i, if its associated disjunction function is of type i, for $i = 1, 2, 3$.

Our study highlighted the role of the various combination functions in the termination property of p-programs. We established PTIME data complexity for p-programs in which every recursive predicate is of type 1. When the certainty lattice $\mathcal{T}$ is finite, the complexity of the bottom-up evaluation is PTIME in the EDB size and is exponential in the size of $\mathcal{T}$. When $\mathcal{T}$ is infinite, we considered several cases, depending on the types of recursive predicates defined in the program and the combination functions employed. We conjectured that under certain conditions, given a p-program P, we can identify a p-program Q, which is the same as P except possibly for the rule certainties, and there are EDBs D_1 and D_2 such that the bottom-up naive evaluation of Q on D_1 terminates in finite time and that of Q on D_2 does not terminate. The conjectures are based on our observations in the special case analysis performed. Future work will address proving those conjectures. Our approach in this study was to analyze simultaneous (non-linear) recurrence equations induced by the input p-programs, which essentially capture the mechanics of the bottom-up naive evaluation of the programs.

Next, we studied the expressive power of the parametric framework, in particular, and the expressive power of the IB approach to uncertainty compared with the AB approach, in general. In Chapter 1, on the basis on how uncertainty is associated with the facts and rules of a program, we classified the approaches to uncertainty in logic programming and deductive databases into the annotation based (AB) and implication based (IB). We also saw the relative pros and cons of these approaches in that chapter. An important metric in a comparison of these two approaches is their relative expressive power: are there queries one can express in one of the AB (or IB) frameworks which cannot be expressed in the other? In Chapter 5, we pointed out the limitations in the expressive power of known IB and AB frameworks and motivated a paradigm of uncertainty programming with (certainty) constraints. Motivated by this need, we proposed an extension to the parametric framework, which we called the *extended parametric (EP) framework.*

To facilitate the comparison of the two approaches in terms of of expressive power, we developed a generic AB framework, which we called the *basic GAB framework.* We showed that the parametric framework can simulate the computation of any AB program P in the basic GAB framework when no annotation constant is used and in no rule body annotation variables occur more than once. We also extended the

basic GAB framework with certainty constraints, and defined the *extended GAB framework*. We established that the EP framework is equivalent to the extended GAB framework from the point of view of the expressive power, when the annotation functions in the latter satisfy similar postulates imposed on the combination functions in the parametric framework. This shows that the concept of certainty constraints unifies the AB and IB approaches. Finally, we showed that while the incorporation of certainty constraints strictly increases the expressive power of both parametric and AB frameworks, it comes at a price. While the basic parametric framework has a continuous immediate consequence operator, this continuity is lost when constraints are added.

To show that the ideas in this thesis lend themselves in developing an easy-to-use and efficient system for deduction with uncertainty, we developed two implementations of the parametric framework, a top-down and a bottom-up. We developed a semi-naive method for evaluating programs in the parametric framework by extending the idea of the corresponding method in the classical case. We established that the semi-naive method proposed is equivalent to the naive method.

To implement the bottom-up evaluation of p-programs on top of the CORAL system, we noted that CORAL needs to be extended with user-defined aggregation functions per iteration, before it can be used for this purpose. Our contribution in this direction was in the design and the testing phases of extending CORAL with this feature implemented by the CORAL database programmer at Wisconsin. The current extended CORAL does not allow assertion of non-ground facts, which is needed for meta-level programming. This extension will help separate the various system modules in our BU system, which in the current implementation are integrated in one module. Although we did not measure, but on the basis of the fact that the XSB and CORAL systems employ advanced processing and optimization techniques, we believe both of our implementations would be very efficient in their computations.

In our work, we considered negation-free clauses. What if negative literals are allowed in the body of p-rules? Studying the semantics of general p-programs which allow negation would be a useful direction of research to pursue. The new issue, in addition to those introduced by the incorporation of negation in the classical framework, is how uncertainties associated with negative literals should be interpreted and manipulated? To give an idea of what it means, consider the following cases. The

classical negation of $A : \alpha$ (where α could be any certainty element, not necessarily a number in $[0, 1]$) is simply $\neg A : \alpha$, which is read as "it is not the case that $A : \alpha$". The negation as failure of this is simply failing to prove $A : \alpha$. The epistemic negation, which we denote by *not*, can be parameterized. This negation is supposed to change the annotation α directly. A simple example where α is a point probability is that $not(A : \alpha)$ is equivalent to $A : 1 - \alpha$. This is how it is done for instance in [Sub87]. More generally, epistemic negation is based on a symmetric function on the lattice from which the certainties are drawn. The point here is that *any* symmetric function can be used to define epistemic negation, explaining why it is parameterizable.

Bibliography

[AIS93] Agrawal R., Imielinski T., and Swami A. Mining association rules between sets of items in large databases. In *19th ACM SIGMOD Conf. on the Management of Data, Washington, DC*, May 1993.

[Apt86] Apt, Krzystof R. Introduction to logic programming. In van Leeuwen, editor, *Handbook of Theoretical Computer Science*. North Holland, 1986.

[Bal87] Baldwin, J.F. Evidential support logic programming. *Journal of Fuzzy Sets and Systems*, 24:1–26, 1987.

[BGP92] Barbara D., Garcia-Molina H., and Porter D. The management of probabilistic data. *IEEE Transactions on Knowledge and Data Engineering*, 4(5):487–502, October 1992.

[Bir67] Birkhoff, Garrett. *Lattice Theory*. Providence. American Mathematical Society, 3rd edition, 1967.

[BS75] Buchanan B.G. and Shortliffe E.D. A model of inexact reasoning in medicine. *Mathematical Biosciences*, 23:351–379, 1975.

[BS89] Blair H.A. and Subrahmanian V.S. Paraconsistent logic programming. *Theoretical Computer Science*, 68:135–154, 1989.

[CGT89] Ceri S., Gottlob G., and Tanca L. What you always wanted to know about datalog (and never dared to ask). *IEEE Transactions on Knowledge and Data Engineering*, 1(1), March 1989.

[CHS+95] Carey M.J., Hass L.M., Schwarz P.M., Arya M., Cody W.F., Fagin R., Flickner A., Luniewski W., Niblack W., Petkovic D., Thomas J., Williams

J.H., and Wimmers E.L. Towards Heterogeneous Multimedia Information Systems: the Garlic Approach. In *5th Intl. Workshop on Research Issues in Data Engineering (RIDE-DOM'95): Distributed Object Management*, pages 124–131, 1995.

[CKW93] Chen W., Kifer M., and Warren D.S. Hilog: A foundation for higher-order logic programming. *Journal of Logic Programming*, 15(3):187–230, 1993.

[CM77] Chandra A.K. and Merlin P.M. Optimal implementation of conjunctive queries in relational databases. In *Proc. 9th Annual ACM Symp. on the Theory of Computing*, pages 77–90, 1977.

[Cod79] Codd, E.F. Extending the database relational to capture more meaning. *ACM TODS*, 4:560–575, 1979.

[CRSS95] Chellappa R., Rajput A., Sirohey S., and Subrahmanian V.S. The FIST face database system. manuscript, 1995.

[CV93] Chaudhuri Surajit and Vardi Moshe Y. Optimization of *real* conjunctive queries. In *Proc. ACM Symp. PODS*, pages 59–70, Washington, D.C., 1993.

[Deb88] Debray Saumya. *SB-Prolog System, Version 3.0. A User Manual.* Dept. of Computer Science, University of Arizona, September 1988.

[DLP91] Dubois Didier, Lang Jérôme, and Prade Henri. Towards possibilistic logic programming. In *Proc. 8th Intl. Conf. on Logic Programming*, pages 581–596, 1991.

[DR94] Debray S. and Ramakrishnan R. Generalized Horn clause programs. manuscript, January 1994.

[EM95] Escalada-Imaz, Gonzalo and Manyà, Felip. Efficient interpretation of propositional multi-valued logic programs. In B. Bouchon-Meunier, R. R. Yager, and L. A. Zadeh, editors, *Advances in Intelligent Computing. Proc. of the 5th Intl. Conf. on Information Processing and Management of Uncertainty in Knowledge-Based Systems (IPMU '94)*, pages 428–439, Paris, France, 1995. Springer-Verlag, LNCS 945.

[End72] Enderton Herbert B. *A Mathematical Introduction to Logic.* Academic Press, 1972.

[Fag96] Fagin, Ronald. Combining fuzzy information from multiple systems. In *Proc. ACM Symp. PODS*, pages 216–226, June 1996.

[Fit88] Fitting M.C. Logic programming on a topological bilattice. *Fundamenta Informaticae*, 11:209–218, 1988.

[Fit91] Fitting M.C. Bilattices and the semantics of logic programming. *Journal of Logic Programming*, 11:91–116, 1991.

[GJ96] Grumbach Stéphane and Jianwen Su. Towards practical constraint databases. In *Proc. ACM Symp. PODS*, pages 28–39, June 1996.

[Gra91] Grahne Gösta. *The problem of Incomplete Information in Relational Databases.* Springer-Verlag, LNCS-554, 1991.

[GS90] Gordon J. and Shortliffe E.H. The dempster-shafer theory of evidence. In G. Shafer and J. Pearl, editors, *Readings in Uncertain Reasoning.* Morgan Kaufmann, San Mateo, California, 1990.

[Hh96] Hähnle Reiner. Exploiting data dependencies in many-valued logics. *Journal of Applied Non-Classical Logics.* 6(1):49–69. 1996.

[HCC93] Han J., Cai Y., and Cercone N. Data-driven discovery of quantitative rules in relational databases. *IEEE Trans. on Knowledge and Data Eng.*, 5(1):29. Feb. 1993.

[IR95] Ioannidis Y.E. and Ramakrishnan R. Containment of conjunctive queries: Beyond relations as sets. *ACM Transactions on Database Systems*, 20, 3:288–324, September 1995.

[Joh94] Johnstone, Heather J. A relational data model for manipulating probabilistic knowledge. Master's thesis, Dept. of Computer Science, Concordia University, Montreal, Canada, September 1994.

[KKTG93] Köstler, G., Kießling, W., Thöne, H., and Güntzer, U. The differential fixpoint operator with subsumption. In *Proc. 3rd Intl. Conf. on Deductive and Object-Oriented Databases (DOOD '93)*, pages 35–48. Springer-Verlag, LNCS-760, December 1993.

[KL88] Kifer M. and Li A. On the semantics of rule-based expert systems with uncertainty. In M. Gyssens, J. Paradaens, and D. van Gucht, editors, *2nd Intl. Conf. on Database Theory*, pages 102–117, Bruges, Belgium, August 31-September 2 1988. Springer-Verlag LNCS-326.

[KL89] Kifer M. and Lozinskii E.L. A logic for reasoning with inconsistency. In *Proc. 4th IEEE Symp. on Logic in Computer Science (LICS)*, pages 253–262, Asilomar, CA, 1989. IEEE Computer Press.

[KS92] Kifer Michael and Subrahmanian V.S. Theory of generalized annotated logic programming and its applications. *Journal of Logic Programming*, 12:335–367, 1992.

[Lak94] Lakshmanan, Laks V.S. An epistemic foundation for logic programming with uncertainty. In *Proc. 14th Conf. on the Foundations of Software Technology and Theoretical Computer Science (FST and TCS'94)*. Springer-Verlag. LNCS-880. December 1994.

[LL90] Li Deyi and Liu Dongbo. *A Fuzzy Prolog Database System*. Research Studies Press, Taunton, England, 1990.

[LL96] Leach Sonia M. and Lu James J. Query processing in annotated logic programming: Theory and implementation. *Journal of Intelligent Information Systems*, 6(1):33–58, January 1996.

[Llo87] Lloyd J. W. *Foundations of Logic Programming*. Springer-Verlag, second edition, 1987.

[LS94a] Lakshmanan Laks V.S. and Sadri F. Probabilistic deductive databases. In *Proc. Intl. Logic Programming Symposium*, pages 254–268, Ithaca, NY, November 1994. MIT Press.

[LS94b] Lakshmanan Laks V.S. and Sadri F. Modeling uncertainty in deductive databases. In *Proc. Intl. Conf. on Database Expert Systems and Applications (DEXA '94)*, Athens, Greece, September 1994. Springer-Verlag, LNCS-856.

[LS96a] Lakshmanan Laks V.S. and Sadri F. Uncertain deductive databases: A hybrid approach, March 1996. Tech. Report TR-DB-96-02. Submitted for Publication.

[LS96b] Lakshmanan Laks V.S. and Shiri Nematollaah. A generic framework for deduction with uncertainty. In *Proc. 6th Intl. Conf. on Information Processing and Management of Uncertainty in Knowledge-Based Systems (IPMU '96)*, Granada, Spain, July 1-5, 1996.

[LS96c] Lakshmanan Laks V.S. and Shiri Nematollaah. A parametric approach to deductive databases with uncertainty. In *Proc. Intl. Workshop on Logic in Databases (LID'96)*, pages 61–81, San Miniato, Italy, July 1996. Springer-Verlag, LNCS-1154.

[LS97] Lakshmanan Laks V.S. and Shiri Nematollaah. A parametric approach to deductive databases with uncertainty. *Accepted to the IEEE Transactions on Knowledge and Data Engineering*. 1997. (A preliminary version appeared in Proc. Intl. Workshop on Logic in Databases (LID'96). Springer-Verlag. LNCS-1154. San Miniato. Italy).

[NBEY93] Niblack W., Barber R., Equitz W., and Yanker M.P. The qbic project: Querying images by content using color, texture and shape. In *SPIE Conf. on Storage and Retrieval for Image and Video Databases*, volume 1908, pages 173–187, June 1993.

[NS91] Ng R.T. and Subrahmanian V.S. Relating Dempster-Shafer theory to stable semantics. In Vijay Saraswat and Kazunori Ueda, editors, *Proc. Intl. Logic Programming Symposium*, pages 551–565, San Diego, California, USA, Oct. 28 – Nov. 1, 1991. MIT Press.

[NS92] Ng R.T. and Subrahmanian V.S. Probabilistic logic programming. *Information and Computation*, 101(2):150–201, December 1992.

[NS93] Ng R.T. and Subrahmanian V.S. A semantical framework for supporting subjective and conditional probabilities in deductive databases. *Automated Reasoning*, 10(2):191–235, 1993.

[RSS92] Ramakrishnan R., Srivastava D., and Sudarshan S. CORAL: Control, relations, and logic. In *Proc. Intl. Conf. on Very Large Databases*, 1992.

[Sad91] Sadri Fereidoon. Modeling uncertainty in databases. In *Proc. 7th IEEE Intl. Conf. on Data Eng.*, pages 122–131, April 1991.

[Sag88] Sagiv Y. Optimizing datalog programs. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 659–698. Morgan-Kaufmann, 1988. Extended abstract of this paper appears in Proc. ACM Symp. PODS, 1987, pp 237–249.

[Sha83] Shapiro E. Logic programs with uncertainties: a tool for implementing expert systems. In *Proc. IJCAI'83*, pages 529–532. William Kaufmann, 1983.

[Shi93] Shiri Nematollaah. Complexity analysis of datalog with uncertainty. Doctoral seminar, Dept. of Computer Science, Concordia University, Montreal, Canada, Dec. 1993.

[SSU91] Silberschatz Avi, Stonebraker Michael, and Ullman J.D. Database systems: Achievements and opportunities. *Communications of the ACM*. 34:110–120. October 1991.

[SSW94] Sagonas Konstantinos, Swift Terrance, and Warren David S. XSB as an efficient deductive database engine. In *Proc. of the ACM SIGMOD Intl. Conf. on the Management of Data*, pages 442–453, Minneapolis, Minnesota, May 1994.

[Sub87] Subrahmanian V.S. On the semantics of quantitative logic programs. In *Proc. 4th IEEE Symposium on Logic Programming*, pages 173–182, Computer Society Press, Washington DC, 1987.

[Sub94] Subrahmanian V.S. Amalgamating knowledge bases. *ACM Transactions on Database Systems*, 19, 2:291–331, 1994.

[Ull89] Ullman, J.D. *Principles of Database and Knowledge-Base Systems*, volume II. Computer Science Press, Maryland, 1989.

[van86] van Emden M.H. Quantitative deduction and its fixpoint theory. *Journal of Logic Programming*, 4(1):37–53, 1986.

[Vis93] Visual Edge Software Ltd., St-Laurent, Quebec, Canada. *The UIM/X Developer's Guide*, 1993.

[vK76] van Emden, M.H. and Kowalski, R.A. The semantics of predicate logic as a programming language. *JACM*, 23(4):733–742, October 1976.

[War96] Warren David. Aggregation in XSB. Unpublished manuscript, 1996.

[Xu 89] Xu Jiyang. *PSB-Prolog System, Version 1.40, User Manual.* ECRC, 1989.

[Zad65] Zadeh L.A. Fuzzy sets. *Information and Control* **8**, pages 338–353, 1965.

[Zad78] Zadeh L.A. Fuzzy sets as a basis for a theory of possibility. *Fuzzy Sets and Systems*, 1(1):3–28, 1978.

[Zvi87] Zvieli Arie. A fuzzy relational calculus. In Benjamin and Cummings, editors, *Expert Database Systems.* Larry Kerschberg, 1987.