NOTE TO USERS

This reproduction is the best copy available.

An Anytime Deduction Heuristic for First Order Probabilistic Logic

Md. Istiaque Shahriar

A Thesis

in

The Department

of

Computer Science and Software Engineering

Presented in Partial Fulfilment of the Requirements for
the Degree of Master of Computer Science at
Concordia University
Montréal, Québec, Canada

September 2008

Your file Votre référence
ISBN: 978-0-494-63162-1
Our file Notre référence
ISBN: 978-0-494-63162-1

ABSTRACT

An Anytime Deduction Heuristic for First Order Probabilistic Logic

Md. Istiaque Shahriar

This thesis describes an anytime deduction heuristic to address the decision and optimization form of the First Order Probabilistic Logic problem which was revived by Nilsson in 1986. Reasoning under uncertainty is always an important issue for AI applications, e.g., expert systems, automated theorem-provers, etc. Among the proposed models and methods for dealing with uncertainty, some as, e.g., Nilsson's ones, are based on logic and probability. Nilsson revisited the early works of Boole (1854) and Hailperin (1976) and reformulated them in an AI framework. The decision form of the probabilistic logic problem, also known as PSAT, consists of finding, given a set of logical sentences together with their probability value to be true, whether the set of sentences and their probability value is consistent. In the optimization form, assuming that a system of probabilistic formulas is already consistent, the problem is: Given an additional sentence, find the tightest possible probability bounds such that the overall system remains consistent with that additional sentence. Solution schemes, both heuristic and exact, have been proposed within the propositional framework. Even though first order logic is more expressive than the propositional one, more works have been published in the propositional framework. The main objective of this thesis is to propose a solution scheme based on a heuristic approach, i.e., an anytime deduction technique, for the decision and optimization form of first order probabilistic logic problem. Jaumard *et al.* [33] proposed an anytime deduction algorithm for the propositional probabilistic logic which we extended to the first order context.

Acknowledgements

From the bottom of my heart I would like to thank the Almighty God for His blessings and divine guidance that have made this thesis possible and complete.

I am grateful to my supervisor, Dr. Brigitte Jaumard, Concordia University Research Chair (Tier- 1) on the Optimization of Communication Networks. Her wide knowledge in the specific subject and related matters was of great value to me. Her understanding, encouraging, personal guidance, and great efforts to explain things clearly and simply, have provided a good basis for this thesis.

I also wish to thank all the faculty members and staff of the Computer Science Department and Concordia Institute for Information Systems Engineering (CIISE). I am grateful to the Faculty of Engineering and Computer Science, Concordia University for supporting this thesis work.

My special gratitude goes to my loving wife Rezwana Kursia and daughter Rusmia Faiza. They have endured my continuous absence from home during the period of my research. Without their encouragement, patience and support it would have been impossible for me to withstand all the pains and complete this thesis work.

I also would like to convey my special thanks to all my friends, specially Nazmun Nahar Bhuiyan, for their help and support that gave me additional strength to stay on the path.

I would like to dedicate my thesis to my parents, who brought me up in this beautiful world, taught me well and always pray and wish me the best.

Dedicated to My Parents

Table of Contents

List of Tables

List of Figures

Acronyms

ADFOPSAT Anytime Deduction First Order Probabilistic Satisfiability.

ADPSAT Anytime Deduction Probabilistic Satisfiability.

AI Artificial Intelligence.

B&B Branch and Bound.

BLOG Bayesian Logic.

CNF Conjunctive Normal Form.

DNF Disjunctive Normal Form.

FOPL First Order Probabilistic Logic.

FOPSAT First Order Probabilistic Satisfiability.

LP Linear Programing.

PPL Propositional Probabilistic Logic.

PSAT Probabilistic Satisfiability.

Chapter 1

Introduction

1.1 Motivation

Reasoning under uncertainty is a crucial issue in many artificial intelligence applications, e.g., expert systems, automated theorem provers, security systems, circuit testing and several other applications. Uncertainty may lie in the data, or in the data collection method or in the inference rules that are used to answer the queries. Due to its importance in multiple disciplines, uncertainty has been viewed from many different perspectives and consequently, several models have been proposed for modeling it. Some of them belong to the fuzzy logic formalism while others are based on classical logic and probability. The earliest work of classical logic and probability is due to Boole in 1854 and more than a century later revisited by Hailperin in 1976. Almost another decade later, in 1986, Nilsson reshaped them in an artificial intelligence framework, under the name of probabilistic logic problem, for both propositional and first order cases.

In this thesis, the main focus is on the probabilistic logic problem in first order context.

Nilsson, in his seminal publications [50], addressed two forms of probabilistic logic problems, namely: decision and optimization forms. The decision form, also commonly known as Probabilistic satisfiability problem, amounts to finding, given a set of logical sentences together with their probability that these sentences are true, whether the set of sentences together with the probability values is consistent. Upon the assumption that the system is consistent, the optimization form attempts to find, given an additional sentence, the tightest possible probability interval such that the resulting system of sentences and their probability values remain consistent. The optimization form of the probabilistic logic problem is also known as probabilistic entailment problem.

1.2 Thesis Contribution

In this thesis, the first contribution is an extension of the anytime deduction method called AD-SOLPPL, proposed by Jaumard *et al.* [33], for propositional case to first order logic. It is to be noted that an anytime deduction technique is one in which the solution process can be stopped at any time and the information on how the solution value is reached can be yield.

Jaumard *et al.* [33] developed a special set of inference rules for the propositional case which they called *'primitives'*. At first, we made a straightforward extension of these inference rules to first order context. As the second contribution, we make a suggestion for a modified set of inference rules w.r.t. the set of primitives used in Jaumard *et al.* [33] as a simple extension from propositional to first order logic set is not sufficient enough to get satisfactory results. Recall that a set of so-called **primitives** consists of a small set of logical sentences together with probabilities for which the analytical solution is available.

As the third contribution, we have developed an improved version of the first algorithm AD-SOLFOPL, a straightforward extension of the initial AD-SOLPPL of Jaumard *et al.* [33] and named as AD-SOLFOPL+. This improved version can lead to better probability values and reaches tighter probability bounds as compared to the initial algorithm. Indeed, building an efficient algorithm is a trade off between tighter bounds and reasonable (not too large) computation times.

The proposed heuristic is much more scalable compared to the exact algorithm proposed in [63] and developed within the master's thesis of Sultana [63]. In [63] [29] the authors propose an efficient algorithm, SOLFOPL, for solving first order probabilistic logic problems (both optimization and decision form) using column generation techniques. It is an exact algorithm but in most of the cases it sacrifices the computation times in exchange with the tightest bounds, i.e., the optimal probability bounds. While SOLFOPL can only solve instances of size 35 formulas at most, the AD-SOLFOPL+ algorithm is capable of solving instances up to 100 formulas with 50 predicates and 40 different variables in a reasonable amount of time.

1.3 Thesis Organization

In Chapter 2, we provide an introduction to the probabilistic logic problem, its background and mathematical programming formulations. Basic concepts related to first order logic are also discussed there. We start Chapter 3 with a categorization of various models which deals with uncertainty by different means; some of the models consider probability as a means to express uncertainty while the others, e.g., fuzzy logic, portrays uncertainty issue without probability consideration. We present analytical and numerical methods and models for

dealing with uncertainty in terms of probability for both propositional and first order cases. A few exact numerical methods and heuristic approaches for numerical solution are explored in Chapter 3. In the subsequent chapter, Chapter 4, we explain the mathematical model and the newly proposed algorithm for the first order probabilistic logic problem. Chapter 5 presents the experimental setup, numerical results and performance analysis of the proposed algorithm with respect to a previous work [63], an exact algorithm for first order probabilistic logic. Finally, in Chapter 6, we conclude with the findings and a few suggestions made for future works which might be carried out in a similar direction.

Chapter 2

Introduction to Probabilistic Logic

2.1 Propositional Probabilistic Logic PPL

Uncertainty is an important issue that needs to be dealt in many artificial intelligence applications, e.g., expert systems often require the ability to reason with a new information from the problem domain and knowledge base where both the knowledge and inference rules are not known with certainty. To deal with the probability of logical formulas, in either propositional or first-order logic, Nilsson [50] proposed the concept of the probabilistic logic problem in 1986 for both the *decision* and the *optimization* forms.

Given a set of sentences with their associated probabilities, the decision form of the probabilistic logic problem, also known as *probabilistic satisfiability* or P^{PSAT}, answers the question: Are these probabilities consistent? Now, let us assume that, the given set of sentences and their associated probabilities is consistent. The optimization form of probabilistic logic problem, also known as *probabilistic entailment* or P^{PENTAIL} problem finds the best possible probability value or intervals, for an additional logical sentence to be true,

such that the augmented set of sentences remains consistent together with its probability. In other words, *the probabilistic entailment problem* determines the range of values of the probability associated with a new logical sentence such that the overall set of sentences, after adding the new sentence, remains consistent.

2.1.1 Decision form

The probabilistic logic problem in *decision* form, also called P^{PSAT}, is defined in [30] as follows. Let $\mathcal{S} = \{S_1, S_2, \ldots, S_m\}$ be a set of logical sentences defined on a set of n propositional (boolean) variables $X = \{x_1, x_2, \ldots, x_n\}$ with the usual Boolean operators. These variables correspond to a set of elementary facts which are either *true* or *false*. Assume probabilities $\pi_1, \pi_2, \ldots, \pi_m$, for these sentences to be true, are given. Are these probabilities consistent?

In propositional calculus, a *literal* is either a propositional variable or its negation, e.g., y_j and $\overline{y}_j$ are positive and negative literals respectively. Let us consider propositional sentences and also assume, as in most expert systems, that the sentences S_i's are logical implications of the form:

$$S_i : \varphi_i \Rightarrow \psi_i \tag{1}$$

$$\text{where} \begin{cases} \varphi_i & \text{the antecedent or premise of the implication} \\ \psi_i & \text{the consequent} \end{cases}$$

are boolean functions. There are two normal forms of a boolean expression, the *Disjunctive Normal Form* or DNF and the *Conjunctive Normal Form* or CNF. A sentence is in DNF if it is expressed as a set of disjunctions of conjunctions of literals, whereas, a sentence is in

CNF if it is expressed using a set of conjunctions of disjunctions of literals. For instance, $((y_{j1} \wedge \overline{y}_{j2}) \vee (\overline{y}_{j1} \wedge y_{j3}))$ and $((y_{j1} \vee \overline{y}_{j2}) \wedge (\overline{y}_{j3} \vee y_{j1}))$ are examples of DNF and CNF respectively.

In propositional calculus, a *clause* is often expressed as a disjunction of literals, e.g., $y_{j1} \vee \overline{y}_{j2}$. On the contrary, conjunction of literals are expressed as $y_{j1} \wedge \overline{y}_{j2}$.

If the φ_i are written in CNF and the ψ_i are written in DNF, then S_i in (1), is a DNF expression:

$$\left(\bigwedge_{j \in A} y_j \Rightarrow \bigvee_{j \in C} y_j \right) \equiv \left(\bigvee_{j \in A} \overline{y}_j \vee \bigvee_{j \in C} y_j \right). \tag{2}$$

Usually, sentences are represented in clausal form as shown above in (2) whereas a clause is often expressed as a disjunction of literals (positive or negative form).

Nilsson [50] defines a *world* as any truth assignment w over $\mathcal{S}$. A world w is *possible* if there exists a truth assignment over the set of X of variables which leads to w over $\mathcal{S}$, and the world is *impossible* otherwise. From now onward, we will only consider possible worlds.

Let $p = (p_1, p_2, \dots)$ be a *probability distribution* on the set W of possible worlds. Assume we are also given probabilities $\pi_1, \pi_2, \dots, \pi_m$, one for each logical sentence. The probability distribution *satisfies* the set of logical sentences together with the probabilities if: *for each sentence S_i ($i = 1, 2, \dots, m$), the sum of all p_j's over all truth assignments w_j which satisfy S_i equals π_i.*

Let A be an $m. \mid W \mid$ matrix such that:

$$a_{ij} = \begin{cases} 1 & \text{if } w_j \text{ satisfies } S_i \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

The *Probabilistic satisfiability* P^{PSAT} can be defined as follows: Is there a probability distribution p such that the following system admits at least one solution? In a mathematical framework, it amounts to: Is there any $p \in [0,1]^{|W|}$ such that the following mathematical program has a solution?

$$(P^{\text{PSAT}}) \quad \begin{cases} \mathbb{1} \cdot p = 1 \\ Ap = \pi \\ p \geq 0, \end{cases} \tag{4}$$

where $\mathbb{1}$ is a k unit row vector and $(k = |W|) \leq 2^m$. The value of k is 2^m in the worst case but in practice it is smaller than 2^m due to the fact that (i) most often not all worlds are feasible and (ii) two different value assignments on the variables may lead to the same possible world. Let p and π denote the column vectors $(p_1, p_2, \ldots, p_{|W|})^T$ and $(\pi_1, \pi_2, \ldots, \pi_m)^T$ respectively.

Example 1: Consider the set of sentences, with $x_1 \equiv$ *Montreal wins today's match in NHL* and $x_2 \equiv$ *Montreal qualifies next round.*

$$S_1 \equiv x_1 \qquad \pi_1 = 0.8$$

$$S_2 \equiv \overline{x}_1 \vee x_2 \qquad \pi_2 = 0.6$$

$$S_3 \equiv x_2 \qquad \pi_3 = 0.3.$$

Let us denote by $\mathbb{1}$ the 2^2 unit row vector [1 1 1 1] and consider

$$p = \begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \end{bmatrix}, \quad \pi = \begin{bmatrix} \pi_1 \\ \pi_2 \\ \pi_3 \end{bmatrix} = \begin{bmatrix} 0.8 \\ 0.6 \\ 0.3 \end{bmatrix}.$$

Now, let us show how to find the set of possible worlds W from the possible assignments on variables x_1 and x_2. For example, let us assign the values (1, 1) to the variables (x_1, x_2). After computing the truth values of sentences, (S_1, S_2, S_3), we come up with a first possible world, w_1= (1, 1, 1). Similarly, by considering the assignments (1, 0), (0, 1) and (0, 0) for (x_1, x_2) and then making the interpretation for the sentences, we get three more possible worlds (1, 0, 0), (0, 1, 1), (0, 1, 0) respectively. Then the corresponding matrix A becomes:

$$\begin{array}{c} \\ S_1 \\ S_2 \\ S_3 \end{array} \begin{array}{c} \begin{array}{cccc} w_1 & w_2 & w_3 & w_4 \end{array} \\ \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix} \end{array}. \tag{5}$$

The mathematical program for the P^{PSAT} problem associated with this example can be written as follows:

$$\mathbb{1} \cdot p \equiv [1111] \times \begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \end{bmatrix} = p_1 + p_2 + p_3 + p_4 = 1,$$

$$Ap = \pi \equiv \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix} \times \begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \end{bmatrix} = \begin{bmatrix} 0.8 \\ 0.6 \\ 0.3 \end{bmatrix},$$

$$\begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \end{bmatrix} \geq 0;$$

i.e., it reduces to find p such that:

$$\begin{aligned} p_1 + p_2 + p_3 + p_4 &= 1 \\ p_1 + p_2 &= 0.8 \\ p_1 + p_3 + p_4 &= 0.6 \\ p_2 + p_3 &= 0.3 \\ p_j \geq 0, \quad & j = 1, 2, 3, 4, \end{aligned}$$

or show that there is no such p. For example, in the above example there is no such p that

satisfies all those constraints, i.e., this instance is inconsistent.

Constructing a possible world is easy as it requires to interpret the values of a set of sentences, using a given set of value assignments to those variables which constitute the sentences. Checking whether a given world is possible is however NP-complete as it reduces to the well known SAT problem.

2.1.2 Optimization form

Suppose we are given an instance of the P^{PSAT} problem ($\mathcal{S}$,π) which is said to be consistent. Let S_{m+1} denote an additional logical sentence (which is deduced possibly from $\mathcal{S}$) with an unknown probability value π_{m+1}. The *probabilistic entailment* or P^{PENTAIL} problem determines the best possible lower and upper bound $[\ \underline{\pi}_{m+1},\ \overline{\pi}_{m+1}\ ]$ of the probability π_{m+1} associated with S_{m+1}, such that $(\mathcal{S} \bigcup S_{m+1},\ (\pi,\ \pi_{m+1}))$ remains consistent. In order to solve the P^{PENTAIL} problem, let us consider the objective function $A_{m+1}p$, where $A_{m+1} = (a_{m+1,j}), j = i, 2, \ldots$ and

$$a_{m+1,j} = \begin{cases} 1 & \text{if } S_{m+1} \text{ is true for the possible world } w_j \\ 0 & \text{otherwise.} \end{cases} \tag{6}$$

We next determine $\underline{\pi}_{m+1} = \min A_{m+1}p$ and $\overline{\pi}_{m+1} = \max A_{m+1}p$ subject to the consistency constraints, as described by (4). Note that it is possible that S_{m+1} may contain some variables which do not appear in the logical sentences of $\mathcal{S}$. No matter what, due to the addition of a new sentence, the set of possible worlds might remain the same or be doubled in the worst case. The mathematical formulation of the *probabilistic entailment* problem

can be written as follows:

$$(P^{\text{PENTAIL}}) \qquad \min\ (\max) \quad A_{m+1}p \qquad \text{subject to:} \quad \begin{cases} \mathbb{1} \cdot p = 1 \\ Ap = \pi \\ p \geq 0. \end{cases} \tag{7}$$

Example 2: Let us consider Example 1 again, setting π_3 to 0.5 to obtain consistency. Add a new logical sentence $S_4 \equiv$ *Montreal wins today's match in NHL* $\Rightarrow$ *Montreal celebrates*, i.e., $S_4 \equiv x_1 \rightarrow x_3$ with $x_3 \equiv$ *Montreal celebrates*. As there is a new logical variable x_3 which was not present before in the sentences of $\mathcal{S}$, there are now eight truth assignments on X, and six possible worlds, i.e., (1, 1, 1, 1), (1, 0, 0, 1), (0, 1, 1, 1), (0, 1, 0, 1), (1, 1, 1, 0) and (1, 0, 0, 0). The third one corresponds to the truth assignments (0, 1, 1) and (0, 1, 0) over X and the fourth one to the truth assignments (0, 0, 1) and (0, 0, 0). The matrix A looks like:

$$\begin{array}{c} \\ S_1 \\ S_2 \\ S_3 \\ S_4 \end{array} \begin{array}{c} \begin{array}{cccccc} w_1 & w_2 & w_3 & w_4 & w_5 & w_6 \end{array} \\ \left[\begin{array}{cccccc} 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 \end{array}\right] \end{array}. \tag{8}$$

We want to find the probability range for π_4 associated with S_4 so that the system $\mathcal{S} \bigcup \{S_4\}$ remains consistent. As in the above matrix, there is a probability distribution p_i associated with each possible world w_i. For example, to obtain π_4 associated with S_4, we examine the last row (of all the listed possible worlds) in the above matrix. Finally, the mathematical formulation of the corresponding probabilistic entailment or P^{PENTAIL} problem looks like:

$$\min \text{ (max) } \pi_4 = \quad p_1 + p_2 + p_3 + p_4$$

subject to:

$$\begin{aligned}
& p_1 + p_2 + p_3 + p_4 + p_5 + p_6 = 1 && (s_4) \\
& p_1 + p_2 \qquad\quad + p_5 + p_6 = 0.8 && (s_1) \\
& p_1 \qquad + p_3 + p_4 + p_5 \qquad = 0.6 && (s_2) \\
& p_1 \qquad + p_3 \qquad + p_5 \qquad = 0.5 && (s_3) \\
& p_j \geq 0, \qquad\qquad\quad j = 1, 2, 3, 4, 5, 6.
\end{aligned}$$

Its optimal solution yields $[\underline{\pi}_4, \overline{\pi}_4] = [0.2,\ 1]$.

2.1.3 Extensions

In this section we discuss two extension to the decision and optimization for propositional probabilistic logic. The first one we discuss extends the precision of probability that is assigned to an individual sentence, i.e., it assigns a range of probability values instead of a point probability. In the second extension we introduce the concept of conditional

probability to the already described forms and extensions.

2.1.3.1 Interval probabilities

It is often more realistic to use probability intervals in place of single point probability values, as already observed by Hailperin [19], in order to consider the imprecision on the sentences. The P^{PSAT} problem, with given probability intervals for the sentences to be true, is then defined as *Is there a probability distribution p such that the system* $(\mathcal{S}, \pi)$ admits at least one solution? In terms of a mathematical program, it leads to:

$$(P^{\text{PSAT}}) \qquad \begin{cases} \mathbb{1} \cdot p = 1 \\ \underline{\pi} \leq Ap \leq \overline{\pi} \\ p \geq 0. \end{cases} \tag{9}$$

The optimization form, P^{PENTAIL} problem, can be written as follows:

$$(P^{\text{PENTAIL}}) \qquad \text{subject to:} \quad \begin{cases} \mathbb{1} \cdot p = 1 \\ \underline{\pi} \leq Ap \leq \overline{\pi} \\ p \geq 0. \end{cases} \qquad \begin{matrix} \min(\max) \quad A_{m+1}p \\ \\ \\ \\ \end{matrix} \tag{10}$$

2.1.3.2 Conditional probabilities

A generalized system should be capable of handling conditional probabilities as many of them make use of knowledge which is known with sufficient precision only in some situations.

Usually this kind of situation is expressed with conditional probabilities as illustrated by Jaumard *et al.* in [30].

Let us assume that $\mathcal{F}$ is a set of first order formulas of size m together with their probability value. But not all the probability values are of the same type. Only the first $q < m$ of them are known with simple probabilities while the rest are conditional. Let $Q = \{1, 2, \ldots, q\}$. Assume that the last $m - q$ appear, with possibly some or all of the first $q - 1$ formulas, in conditional probabilities $\pi_{i/j} = \pi(F_i|F_j)_{(i,j)\in T}$ where $T \subseteq M \times M$ and $M = \{1, 2, \ldots, m\}$. Let $C \subseteq M$ be the index of the formulas appearing as condition in the probabilities $\pi_{i/j}$. By definition

$$\pi_{i/j} = \pi(F_i|F_j) = \frac{\pi(F_i \wedge F_j)}{\pi(F_j)} = \frac{(A_i \wedge A_j)p}{\pi_j}$$

where $A_i \wedge A_j = (a_{ik} \wedge a_{jk})_k$.

Then the conditional first order probabilistic logic problem can be expressed as follows: Determine $\underline{\pi}_{m+1}, \overline{\pi}_{m+1}$ such that

$$\underline{\pi}_{m+1} = \min A_{m+1}p$$

subject to:

$$\begin{cases} \mathbb{1} \cdot p = 1 & \\ \underline{\pi}_i \leq A_i p \leq \overline{\pi}_i & i \in Q \\ A_j p - \pi_j = 0 & j \in C \cap (M \setminus Q) \\ (A_i \wedge A_j)p - \overline{\pi}_{i/j}\pi_j \leq 0 & (i,j) \in T \\ 0 \leq (A_i \wedge A_j)p - \underline{\pi}_{i/j}\pi_j & (i,j) \in T \\ \pi_j \geq 0 & j \in C \cap (M \setminus Q) \\ p \geq 0. & \end{cases}$$

and $\overline{\pi}_{m+1} = \max A_{m+1} p$ subject to the same set of constraints.

2.2 First-Order Probabilistic Logic FOPL

So far, we have described the probabilistic logic model in the context of propositional logic. In this section, we will introduce the probabilistic logic model in the context of first order logic.

2.2.1 Decision form

Consider a set of q logical predicates $\mathcal{P} = \{P_1, P_2, \ldots, P_q\}$ defined on a set of n variables $X = \{x_1, x_2, \ldots, x_n\}$. Let $\mathcal{F}$ be a set of m formulas in prenex normal form defined as follows:

$$\mathcal{F} = \{F_i = (Q_1^i x_1)(Q_2^i x_2)\ldots(Q_n^i x_n)(M_i) :$$

$$M_i = S_i(P_1, P_2, \ldots, P_q),\ i = 1, 2, \ldots, m\} \tag{11}$$

where $(Q_j^i x_j), j = 1, 2, \ldots, n$ is either $(\exists x_j)$ or $(\forall x_j)$, and S_i is a logical sentence that contains no quantifier and is built on the set of predicates using the logical connectives $\neg, \wedge$ and $\vee$. $(Q_1^i x_1)(Q_2^i x_2)\ldots(Q_n^i x_n)$ is called the prefix and M_i is called the matrix of the formula F_i, $i = 1, 2, \ldots, m$. For instance, $\forall x \exists y[P_1(x, y) \vee P_2(y)]$ is a first-order formula, where $P(x, y)$ and $Q(y)$ are predicates, x is universally quantified and y is existentially quantified.

The *decision* form of first-order probabilistic logic problem, or P^{FOPSAT} for short, is defined as follows: given a set $\mathcal{F}$ of m formulas in prenex normal form, and a probability vector $\pi = (\pi_1, \pi_2, \ldots, \pi_m)$ associated with these formulas, is the set $(\mathcal{F}, \pi)$ consistent?

We use the definition of a *world* as any truth assignment w over $\mathcal{F}$, as in Nilsson [50]. A *world* is *possible* if there exists a truth assignment over the set of predicates which leads to w over $\mathcal{F}$, and the world is impossible otherwise.

Building a possible world amounts considering possibly several truth assignments on the set of predicates, i.e., one value assignment for an existentially quantified variable, but two value assignments for a universally quantified variable. Checking whether a world is possible amounts to checking whether a set of first-order formulas is satisfiable: it is therefore a NP-complete problem since it is generalized to the NP-complete satisfiability problem for probabilistic logic. [16] [9].

2.2.1.1 Decidability

Before we discuss the solution for the P^{FOPSAT} problem, let us have a closer look to the specific difficulties of first-order logic. One of the major difficulties to be considered is the decidability issue. According to [27] there are problems that cannot be solved by any algorithm that can run on a computer; they are called undecidable problems. In other words, a problem is said to be decidable if there exists an effective algorithm (computer program) that can solve that problem. More general references on decidability and undecidability can be found in [58]. Unlike the propositional calculus, first-order logic is undecidable, provided that the language has at least one predicate having at least 2 variables. Let us recall the following two fundamental results as well.

Theorem 1. *(Church [10], Turing [65]). The satisfiability problem for first-order logic is undecidable.*

Theorem 2. *(Trakhtenbrot [64], Craig [12]). The satisfiability problem for first-order logic on finite structures is undecidable.*

In spite of these negative results, i.e., no sound and complete proof system for validity even on finite structures, several studies were conducted in order to explore decidable set of first-order sentences. A recent survey written by Hustadt *et al.* [28] provides references to the known decidable set of first-order sentences. Indeed, there are several well decidable set of first-order sentences (see [53] for detail), e.g., the AEA class is decidable which consists of all the relational (i.e., without function symbols) first-order sentences with quantifier prefix of the form $\forall\exists\forall$ [17], (see, e.g., Dreben and Goldfarb [13] and Aspvall *et al.* [2]), or the set of sentences with two-variable (see, e.g., Gradel *et al.* [17]). From now onward, we will only

consider sets of formulas belonging to some set of decidable first-order sentences.

2.2.1.2 First-order concepts

In this section, we describe some first-order logic concepts for better understanding.

- **Literal ([40], p. 517):** A *literal* is an atomic expression or the negation of an atomic expression.

- **Clause ([9], p. 48):** A *clause* is a finite disjunctions (or $\vee$) of zero or more literals. For example, $(\neg lily(X) \vee flower(X))$ is a clause with two literals $\neg lily(X)$ and $flower(X)$.

- **Empty clause ([9], p. 48):** An *empty clause* ($\square$) can be attained by creating contradiction between two sets of literals having opposite sign. For instance, $flower(X)$ and $\neg flower(X)$ results in a *null* or *empty clause* ($\square$).

- **Function ([40], p. 53):** A *function* denotes a mapping of one or more elements in a set (called the *domain* of the function) into a unique element of another set (the range of the function) where elements of the domain and range are objects in the world of discourse. Every function has an associated *arity*, indicating the number of elements in the domain mapped onto each element of the range. For instance, *father(salma)* denotes a function of arity 1 that map people onto their (unique) male parent and *plus(2, 3)* is a function of arity 2 that maps two numbers onto their arithmetic sum 5.

- **Term ([40], p. 54):** A *term* is either a constant, variable or function expression which is used to denote objects and properties in a problem domain.

For example, *cat, mother(muller)*, etc.

- **Predicate ([40], p. 55):** A *predicate* names a relationship between zero or more objects in the world where the number of objects so related is the arguments of the predicate. An example of predicate with 2 arguments is *likes(X, jane)* where, X is a variable and jane is a constant. The arguments of a predicate may include terms and also variables or function expressions. For example, *friends(fatherof(salma), fatherof(raika))* is a predicate describing a relationship between two objects in a domain of discourse. If the function expressions (*fatherof(salma)* is *john* and *fatherof(raika)* is *shawn*) are evaluated, the expressions become *friends(john, shawn)*.

- **Interpretation ([9], p. 31):** An *interpretation* for a first-order formula F consists of a nonempty domain D, and an assignment of *values* to each constant, function symbol and predicate symbol occurring in F.

- **Satisfiable ([9], p. 34):** A formula F is *satisfiable* (*consistent*) if and only if there exists an interpretation I such that F is evaluated to T in I. If a formula F is T in an interpretation I, we say that I is a model of F and I satisfies F.

- **Logical consequence ([9], p. 34):** A first-order formula F_{m+1} is a *logical consequence* (*logically follows*) of (from) set of formulas $F_1, F_2, \ldots, F_n$ if and only if for every interpretation I, if $F_1 \wedge F_2 \wedge \cdots \wedge F_n$ is true, F_{m+1} is also true in I.

- **Inference rules ([40], p. 65):** An *inference rule* for both the propositional and first-order logic can be defined as a formal way of generating a new formula which is a logical consequence of a given set of existing formulas.

- **Sound ([40], p. 66):** An inference rule is *sound* if every formula produced by the

inference rule from a set $\mathcal{F}$ of formulas also logically follows from $\mathcal{F}$.

- **Complete ([40], p. 66):** An inference rule is *complete* if, given a set $\mathcal{F}$ of formulas, the inference rule can infer every formulas that logically follows from $\mathcal{F}$.

- **Refutation complete ([40], p. 529):** An inference rule is *refutation complete* if, given an unsatisfiable set of clauses, the unsatisfiability can be established by use of this inference rule alone. All resolution based automated theorem provers use refutation proof procedure for first-order formulas ([40], p. 517).

- **Proof procedure [40], p. 66):** The *proof procedure* is a combination of an *inference rule* and an algorithm for applying the inference rule to a set of formulas. In first-order logic, a formula that logically follows from a given set of formulas can be produced by using *proof procedures.* The *modus ponens* is a *sound* (but not *complete*) inference rule for first-order logic. There are two kinds of proof procedures namely, *direct proof* and *refutation proof* (lecture notes of Dr. Haarslev, Concordia University, winter 2006). Consider a set of formulas $F_1, F_2, \ldots, F_n$. *Direct proof* shows that a new formula F_{m+1} is a logical consequence. That means *direct proof* shows that $(F_1, F_2, \ldots, F_n) \rightarrow F_{m+1}$ leads to consistency. Whereas, *refutation proof* adds the negation of the new formula $\neg F_{m+1}$ and proves that $(F_1 \wedge F_2 \wedge \cdots \wedge F_m \wedge \neg F_{m+1})$ leads to an inconsistency. *Refutation proof* is *sound* and *refutation complete.*

- **Substitution ([9], p. 75):** A *substitution* is a finite set of the form $\{t_1/v_1, \ldots, t_n/v_n\}$, where every v_i is a variable, every t_i is a term different from v_i, and no two elements in the set have the same variable after the stroke symbol. The following two sets are substitutions: $\{f(z)/x, y/z\}$, $\{a/x, g(y)/y, f(g(b))/z\}$, i.e., a substitutes x.

- **Unification ([9], p. 76):** A substitution $\theta = \{t_1/v_1, \ldots, t_n/v_n\}$ is called a *unifier* for a set of expressions $\{E_1, \ldots, E_k\}$ if and only if $E_1\theta = E_2\theta = \cdots = E_k\theta$. The set $\{E_1, \ldots, E_k\}$ is said to be *unifiable* if there is a unifier for it. For example, the set $\{P(a, y), P(x, f(b))\}$ is unifiable since the substitution $\theta = \{a/x, f(b)/y\}$ is a *unifier* for the set.

2.2.1.3 Resolution refutation

In first-order formulas, we have to deal with quantifiers, predicates and functions, consequently, possible worlds for first-order logic cannot be obtained as easily as in propositional logic. Determining the set of possible worlds from a set of decidable first-order formulas amounts to checking whether a set of first-order formulas is consistent. In practice, it is done using the *resolution refutation method.* This section briefly explains the rules followed by the resolution refutation method for our better understanding. Before going more into the resolution procedure, let us first understand the skolemization technique in order to deal with the quantifiers and reach a standard form also known as a skolem standard formula (see also [9], p. 47) on which the resolution procedure can be applied .

Let $\mathcal{F}$ be a set of m formulas in prenex normal form defined as follows:

$$\mathcal{F} = \{F_i = (Q_1^i x_1)(Q_2^i x_2) \ldots (Q_n^i x_n)(M_i) : \\ M_i = S_i(P_1, P_2, \ldots, P_q),\ i = 1, 2, \ldots, m\} \tag{12}$$

where $(Q_j^i x_j), j = 1, 2, \ldots, n$ is either $(\exists x_j)$ or $(\forall x_j)$, and S_i is a logical sentence that

contains no quantifier and is built on the set of predicates using the logical connectives $\neg, \wedge$ and $\vee$. The first part, i.e., $(Q_1^i x_1)(Q_2^i x_2)\ldots(Q_n^i x_n)$ is called the prefix and M_i is called the matrix of a formula. Now, the basic idea behind the skolemization technique is as follows: If we can remove all the existential quantifiers from a formula with proper replacements such that their effects remain same on that formula, a first order formula is transformed to a standard form which is very similar to a propositional formula and the general rules which can be applied on a propositional formula can also be applied on it. To achieve that form, an existential quantifier is replaced either by a constant or a function (also known as skolem constant). An example is presented shortly which explains the skolemization technique as an essential part of resolution refutation.

The *resolution refutation* ([40], p. 517) proof procedure answers a query or deduces a contradiction out of a set of clauses (i.e., after skolemization in order to deal with the quantifiers) where contradiction is represented by the null clause ($\square$). The contradiction is produced by using the modus ponens rule for resolving pairs of clauses from the database. If the resolution procedure fails to produce a contradiction directly, then the *resolvent* clause produced by the resolution is added to the database of clauses and the process continues. More precisely, in a refutation proof procedure, the new formula is negated and added to the set of formulas (axioms) that are known to be true. Then, it uses the resolution rules of inference to show that this leads to a contradiction. If the theorem prover shows that the negated goal is inconsistent with the given set of formulas, it follows that the original goal was consistent. A strategy is said to be *complete* if it guarantees to find contradiction using *refutation-complete* inference rule whenever a set of formulas is unsatisfiable ([40], p. 529). Resolution refutation proofs involve the following general steps ([40], p. 517):

1. The formulas (after dealing with the quantifiers) or axioms are arranged into *clause form.*

2. The goal is negated in clause form and added to the set of axioms.

3. A set of clauses are *resolved* together, producing new clauses that logically follow from them.

4. A contradiction is reached by generating the empty clause ($\square$).

5. The aim of a substitution is to produce a clause whose truth value is opposite to the negated goal which eventually results in an empty clause ($\square$).

Binary resolution is the most common form of resolution which is applied between two clauses when one contains a literal and the other one its negation ([40], p.517). Unification is needed if these literals contain variable to make them equivalent. Eventually, the resulting new clause comprises of the disjuncts of all the predicates in the two clauses minus the literal and its negative instance, having been *resolved away.* The resulting clause undergoes the necessary unification and substitution which make sure the predicate and its negation are *equivalent.* For example, the knowledge base for the *dead dog* problem ([40], p.518) may be represented in clause form as:

$$(\neg dog(X) \vee animal(X)) \wedge (\neg animal(Y)) \vee die(Y)) \wedge (dog(fido)).$$

To this expression, we add (by conjunction) the negation of our goal which is $\neg die(fido)$. The resolution proof for this is shown in Figure 1.

Transforming formulas into clause form requires 8 steps which are given below ([9], p. 37):

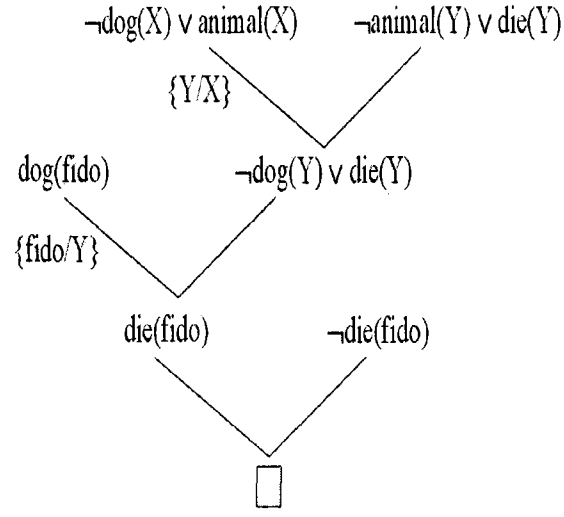

Figure 1: Resolution proof for the *dead dog* problem

1. Use the following laws to eliminate the logical connectives $\leftrightarrow$ (equivalence) and $\rightarrow$ (implication).

$$F \leftrightarrow G \equiv (F \rightarrow G) \wedge (G \rightarrow F). \tag{13}$$

$$F \rightarrow G \equiv \neg F \vee G. \tag{14}$$

2. Repeatedly use the following laws to bring the negation signs immediately before atoms:

$$\neg(\neg F) \equiv F. \tag{15}$$

and De Morgan's laws:

$$\neg(F \vee G) \equiv \neg F \wedge \neg G, \tag{16a}$$

$$\neg(F \wedge G) \equiv \neg F \vee \neg G. \tag{16b}$$

and the laws

$$\neg((\forall X)F(X)) \equiv (\exists X)(\neg F(X)), \tag{17a}$$

$$\neg((\exists X)F(X)) \equiv (\forall X)(\neg F(X)). \tag{17b}$$

3. Rename variables so that no two quantifiers bind the same variable.

4. Use the following laws to move the quantifiers to the left of the entire formula to obtain a prenex normal form:

$$(QX)F(X) \vee G \equiv (QX)(F(X) \vee G). \tag{18a}$$

$$(QX)F(X) \wedge G \equiv (QX)(F(X) \wedge G). \tag{18b}$$

$$(\forall X)F(X) \wedge (\forall X)H(X) \equiv (\forall X)(F(X) \wedge H(X)). \tag{19a}$$

$$(\exists X)F(X) \vee (\exists X)F(X) \equiv (\exists X)(F(X) \vee H(X)). \tag{19b}$$

$$(Q_1X)F(X) \vee (Q_2X)H(X) \equiv (Q_1X)(Q_2Z)(F(X) \vee H(Z)) \tag{20a}$$

$$(Q_3X)F(X) \wedge (Q_4X)H(X) \equiv (Q_3X)(Q_4Z)(F(X) \wedge H(Z)) \tag{20b}$$

For instance let us obtain a *prenex normal form* for the following formula:

$$(\forall X)(\forall Y)((\exists Z)\ (p(X,Z) \wedge p(Y,Z)) \rightarrow (\exists U)q(X,Y,U)).$$

Using (14), we get:

$(\forall X)(\forall Y)(\neg((\exists Z)\ (p(X,Z) \wedge p(Y,Z))) \vee (\exists U)q(X,Y,U))$.

Using (17b) and (16a), we get:

$(\forall X)(\forall Y)((\forall Z)\ (\neg(p(X,Z) \vee \neg p(Y,Z)) \vee (\exists U)q(X,Y,U))$.

Using (18a), we get:

$(\forall X)(\forall Y)(\forall Z)(\exists U)\ (\neg(p(X,Z) \vee \neg p(Y,Z) \vee q(X,Y,U))$.

So, we obtain the last formula as a prenex normal form of the first formula.

5. Replace each existentially quantified variable by a skolem constant or skolem function and remove the quantifier $\exists$.

   case 5a. If x is not universally quantified, replace x by unique, fresh skolem constant.

   For example: $\exists X p(X)$ is skolemized to p(a). Similarly,

   $\exists X \forall Y p(X,Y,X) \Rightarrow \forall Y p(b,Y,b)$ and

   $\exists X \exists Y \forall Z p(X,Y,Z) \Rightarrow \forall Z p(a,b,Z)$.

   case 5b. If the predicate has more than one argument and the existentially quantified variable is within the scope of universally quantified variables, the existential variable must be a function of those other variables. For example:

   $\exists X \forall Y \forall Z \exists U \forall V \exists W\ p(X,Y,Z,U,V,W) \Rightarrow \forall Y \forall Z \forall V\ p(a,Y,Z,f(Y,Z),V,g(Y,Z,V))$.

6. Remove all universal quantifiers

7. Convert the expression to the CNF form. This requires using the associative and distributive properties of $\vee$ and $\wedge$.

8. Rewrite as a set of clauses, where "$\wedge$" is considered to be default between clauses. For instance, $(p(X,Y) \vee r(X)) \wedge q(Y)$ is written as two separate clauses $(p(X,Y) \vee r(X))$ and $q(Y)$.

2.2.1.4 Strategies and simplification methods for resolution

Resolution is a method of theorem proving that proceeds using proof by contradiction. This method can be viewed as a search problem where the solution is the path from the initial set of clauses to the empty clause (contradiction). An exact resolution employs breadth-first search strategy which is exponential. Therefore, we have to apply heuristic strategy for large problems. A *strategy is complete* if by using it with a refutation-complete inference rule, it is guaranteed to find the refutation whenever a set of clauses is unsatisfiable ([40], p. 529). Let us discuss some of the well known strategies used in resolution.

- **Breadth-first strategy ([40], p. 529):** In *breadth-first search* the comparison among the clauses is done in an exhaustive manner. Therefore, it guarantees to find the shortest solution path. It is also a *complete* strategy in the sense that if it continues to search for long enough, it is guaranteed to find a contradiction if one exists.

- **Set of support strategy ([40], p. 530):** *Set of support* of Wos and Robinson [67] is a good strategy for large clause space. In this strategy, a subset T is specified from a set S of input clauses for resolution. It can be proved that, if S is unsatisfiable and $S - T$ is satisfiable, then this strategy is *refutation complete*. It is complete only if

the right (correct) subset is chosen.

- **Unit preference strategy ([40], p. 531):** The *unit preference strategy* usually resolves with shorter clauses specially, clauses with one literal, called unit clause whenever they are available. This strategy guarantees that the resolvent is smaller than the largest parent clause. *Unit resolution* is a related strategy that requires that one of the resolvent must be a unit clause. However, it is an incomplete strategy. The combination of unit preference and set of support strategy can produce a more efficient complete strategy.

- **Linear input from strategy ([40], p. 531):** *Linear input from strategy* directly use negated goal and the original set of formulas. It takes the negated goal and resolves it with one of the formulas at a time. This process repeats until the empty clause is produced. However, this is an incomplete strategy.

We can use some simplification methods in order to reduce the search space and speed up a resolution-based problem solver for finding a solution. Let us see a few of them below.

- **Elimination of tautological clauses ([40], p. 533):** A tautological clause does not need to be considered as it will never be falsified. Therefore, it does not play a useful role in a solution attempt.

- **Subsumption ([40], p. 533):** *Subsumption* is a complete strategy in which more specific clauses are removed, e.g., if S contains predicates FATHER(x, y) then we can remove FATHER(john, salma).

2.2.2 Optimization form

In his paper on probabilistic logic, Nilsson [50] also discussed about the optimization form of first order predicate logic or FOPL which can be stated as follows: Given a set $\mathcal{F}$ of m formulas in prenex normal form, and a probability vector π associated with these formulas, such that the set $(\mathcal{F}, \pi)$ is consistent. Let us consider an additional formula F_{m+1}. The optimization form deals with how to compute bounds on the probability of F_{m+1} so that the system $(\mathcal{F} \cup F_{m+1}, \pi)$ remains consistent.

Chapter 3

Literature Review

The concept of uncertainty is a relation with some form of information deficiency. Uncertainty results whenever any form of incompleteness, impreciseness, fragmentation, unreliability, vagueness, contradiction or information deficiency takes place. Lakhsmanan and Sadri [39] defined uncertainty for an agent's knowledge about a piece of information in the form of a confidence level. A more precise definition came from Smithson [60], who defined uncertainty as a subjective measure of certainty of something (e.g., occurrence of some events) therefore, it can be modelled in terms of a quantitative measure, i.e., a numerical value between 0 and 1 where 0 denotes falsity and 1 denotes truth. Klir and Yuan ([37], p.268) categorized uncertainty into two basic types as fuzziness or vagueness and ambiguity (ambiguity may further be divided into imprecision and discord). The authors ([37], p.267) also claimed, reasoning under uncertainty is an important issue in several theories, e.g., fuzzy set theory [68], possibility theory [69, 14].

In this chapter, we will briefly discuss about few uncertainty based models in AI, for propositional as well as for first-order logic. As we will see, some of them have their main

focus on probability theory while the rest are fuzzy logic or other formalism based. In probabilistic logic, classical propositional and first-order formulas are extended with probability while formulas remain true or false. Even though expressed differently, in practice it is equivalent to a probability logic. Our main focus will be on the *probabilistic logic* model, equivalent to Nilsson's [50] formalism and its extension.

This chapter is organized as follows: In Section 3.1, we briefly describe some mostly studied models, for propositional case, which treat uncertainty by different means. Some of the models consider probability as a means to express uncertainty while the others, e.g., fuzzy logic, portrays uncertainty issue without probability consideration. Section 3.2 presents models (with and without probability) which deal with uncertainty for first order case. In section 3.3 we present analytical methods to deal with propositional probabilistic logic problems while in Section 3.4 we discuss some numerical methods for propositional probabilistic logic problems. We discuss exact numerical methods and heuristic numerical methods in this section. In Section 3.5, we describe the exact numerical method for first order logic. A few first order issues and heuristic strategies for numerical solution are also explored in this section.

3.1 Propositional Models Dealing with Uncertainty

Uncertainty has been studied and treated under many different types of models and perspectives for propositional case. Broadly they can be categorized into two major groups: non-probabilistic and probabilistic formalisms. In the following two subsections we will see some approaches for non probabilistic and probabilistic cases which are taken from the classification suggested by Lakhsmanan and Sadri in [39].

3.1.1 Propositional non-probabilistic models

Fuzzy logic programming: Fuzzy logic programming was introduced by van Emden in his seminal paper on quantitative deduction [66], and further developed by various researchers, including Steger *et al.* [61] and Schmidt *et al.* [59]. Instead of going for absolute value of uncertainty, they go for an approximation on how uncertainty can be expressed and measured.

Annotated logic programming: Annotated logic programming framework was introduced by Subrahmanian [62] and later studied by Blair and Subrahmanian [5] and Kifer and Li [35]. Kifer and Li extended the framework of Subrahmanian [62] to a formal semantics for rule-based systems with uncertainty. Finally, this framework was generalized by Kifer and Subrahmanian into the Generalized Annotated Programming (GAP) framework [36].

Evidence theoretic logic programming: Evidence theoretic logic programming has been mainly studied by Baldwin and Monk [4]. They use only Dempster's evidence theory as the basis for dealing with uncertainty in their logic programming framework. It is to be noted that The Dempster-Shafer theory is a mathematical theory of evidence based on belief functions and plausible reasoning [4], which is used to combine separate pieces of information, known as evidence, to calculate the probability of an event.

3.1.2 Propositional probabilistic models

Logic and Probability based approach: The earliest works of logic and probability are due to Boole in 1854 [7] and more than a century later they were revisited by Hailperin [21]

in 1976. In 1986, Nilsson [50] used a 'possible worlds' approach and reshaped them for Artificial Intelligence under the name of probabilistic logic problem both for propositional logic and first order case. Carnap ([8] also presented a seminal work on probabilistic logic. Some of the probabilistic logic programming works are based on probabilistic logic approaches, such as Ng and Subrahmanian's work on probabilistic logic programming [49].

Annotation based approach: Ng and Subrahmanian [49] were the first to propose a probabilistic basis for logic programming. The idea is that uncertainty is always associated with individual atoms (or their conjunctions and disjunctions), while the rules or clauses are always kept classical. In Ng and Subrahmanian [49], uncertainty in an atom is modeled by associating a probabilistic truth value with it, and by asserting that it lies in an interval. The main interest is in characterizing how precisely we can 'bound' the probabilities associated with various atoms. However, as pointed out in Ng and Subrahmanian [49], even if one starts with precise point probabilities for atomic events, probabilities associated with compound events can only be calculated to within some exact upper and lower bounds, thus naturally necessitating intervals.

Implication based approach: The first implication based framework for probabilistic deductive databases was proposed in Lakshmanan and Sadri [38]. The idea behind implication based approach is to associate uncertainty with the facts as well as rules in a deductive database.

3.2 First Order Models Dealing with Uncertainty

First order logic is much more expressive than propositional logic. But things are more complex due to many reasons. The addition of quantifier and the issue of decidability are

two such points among them.

3.2.1 First order non-probabilistic models

Next we briefly mention about two non-probabilistic approaches which are means to deal with uncertainty without probabilistic approach.

3.2.2 First order probabilistic models

An overview

In this section, we explore some first-order models with probabilities. Several papers have been published on the first-order probabilistic logic [22, 49, 42, 52, 47]. We will discuss a few of those models which are related to our work on models for reasoning uncertainty with probabilities. In the subsequent sections, we will cover Halpern's [22] *degrees of belief* and *chance setup*, Lukasiewicz's [42] *probabilistic logic programming* and Milch and Russell's [45] *Bayesian logic* or BLOG.

Halpern's semantics for probabilistic logic

Halpern [22] provided semantics to first-order logics for two different approaches of probabilistic reasoning. One of them deals with the probability in the domain or at the level of statistical information which is illustrated with the statement "The probability that a randomly chosen bird flies is greater than 0.9". The other one deals with uncertainty at the level of possible outcomes or worlds (Nilsson [50]). Halpern illustrated the second one with the statement "The probability that Tweety (a particular bird) flies is greater than 0.9". The first approach of Halpern is also called *chance setup* [18] while the latter one is called "*degrees of belief*" [3]. A similar type of semantics was also studied by Bacchus [3].

For the probability on domain, Halpern [22] assumed that there is a given first-order language for reasoning about some domain. If a formula φ in logic is given, formulas of the form $w_x(\varphi) \geq 1/2$ are also allowed which can be interpreted as "the probability that a randomly chosen x in the domain satisfies φ is greater than or equal to $1/2$". For instance [22], $w_x(Son(x,y))$ describes the probability that a randomly chosen x is the son of y, whereas $w_y(Son(x,y))$ describes the x is the son of randomly chosen y. Halpern extended this to allow arbitrary sequences of distinct variables in the subscript by providing the syntax and semantics of a two classified languages. The function and predicate symbols, and a limited class of object variables $x^0, y^0, \ldots$, in the φ are considered in the first class which describe the elements of the domain of reasoning. The second class defines syntax for binary function symbols $+$ and $\times$, constant symbols 0 and 1, and limited class of field variables $x^f, y^f, \ldots$, with the intention of ranging over the real numbers. Halpern [22] allowed only two field functions $+$ and $\times$ in his syntax and did not consider Bacchus's [3] *measuring functions* which map object terms into field terms.

In the semantics of chance setup, Halpern [22] gave a probability structure called *type 1*, where probability functions are defined over the domain. These probability functions are standard real-valued and countably additive which make them significantly different from the semantics of Bacchus [3] where non-standard probability functions are considered which take values in arbitrary ordered fields and are only finitely additive.

The syntax and semantics for *degrees of belief* is essentially the same as in the former approach except for few cases. Among them, probability is defined over the set of states or possible worlds [50] instead of taking over domain. The functions or predicates might have different meaning for different states. The probability structure called *type 2* is defined

for reasoning about possible worlds. Some simplifying assumptions were also made for the representation of probability on possible worlds. For example, all functions and predicates can take fixed or flexible structure. Moreover they also assume that there is only one domain and only one predicate measure on the set of states.

Halpern [22] used a deductive mechanism to infer new information for *chance setup* and *degrees of belief*. Deductive mechanism (e.g., modus ponens) usually concludes a new information from a given certain premises.

Halpern [22] used an axiom system which is sound, but is complete only for bounded sized domains. For instance, in order to provide complete axiomatization, problems are restricted to unary predicates for the probability on a domain. However, there is no straightforward way to capture statistical information from degrees of belief or vice-versa. Therefore, in order to simultaneously reason about statistical information and degrees of belief, these two approaches are combined by Halpern [22] in one framework.

Bayesian logic or BLOG

Milch and Russell [45] pointed out that in order to express the real world problem, propositional probabilistic languages such as *Bayesian network* or BN are inadequate. This inadequacy of BN results from the fact that a fixed set of random variables as well as dependencies and probability distributions for each of the variables must be specified individually. Real world problems often involve many objects and their dependent objects which are unknown or uncertain in nature. So, to define uncertainty using a fixed set of random variables and a fixed dependency structure is not sufficient. For example tracking multiple people from a video sequence, is very difficult to define using a fixed set of variables and structures as the number and the types of people are unknown in advance.

Milch and Russell [45] proposed a new probabilistic modeling language, called *Bayesian logic* or BLOG which can model large families of random variables compactly by abstracting over objects and mapping between objects and observations. The authors [45] define probability distribution over relational structures with varying sets of objects. For a particular scenario, BLOG specifies certain non-random aspects which is handled by a typed first-order language and the remaining aspects are specified with a probability model. The probability model describes a generative process for constructing a possible world in two steps. At the first step, boolean functions are assigned values to evaluate some objects. In the second step, new objects are added to their world.

In order to infer, Milch and Russell [45] use a sampling-based inference algorithm which proved to be too slow for many tasks. Therefore, they are still working on improving their inference mechanism.

3.3 Analytical Solution for PPL

In this section, we present studies on analytical solution for *Propositional Probabilistic Logic* problem or PPL in short. At first, we will discuss Boole's analytical solution [7] in algebraic form to PPL problem. Later on, we will discuss about Hailperin's extensions of Boole's method. Hailperin was one of the first authors who investigated Boole's analytical solution. He pointed out that there is a way to find an analytical expression for the solution using the enumeration of the extreme points of a particular polyhedron. Finally, Hansen *et al.* [24] formulated and provided the proof of the consistency condition for the analytical expression. Let us start with Boole's algebraic method.

3.3.1 Boole's algebraic method

To solve the decision and optimization forms of probabilistic logic problem analytically, Boole outlined some algebraic manipulations in his book of 1854 [7], and in several of his contemporary and subsequent papers which are similar for both cases. The simplest and most efficient method carries on as follows [23] :

Step 1. All m logical sentences are expressed as sum of complete products, i.e., products of all variables in direct or complemented form.

Step 2. Associate each of these products with an unknown probability p_j, write linear equations such that for each associated logical sentence $\sum_j p_j = \pi_i$ for S_i to be true. Noted that $j \leq 2^m$, because not all the products are valid and some of the products may be duplicated. Add constraints stating that $\sum_j p_j = 1$ and each p_j is non-negative.

Step 3. Eliminate as many p_j as possible from the so obtained equalities and inequalities (we will see in an example later).

Step 4. From the inequalities obtained in the previous steps eliminate the remaining probabilities p_j as well as π_{m+1} by considering all upper and lower bounds on one of them, stating that each lower bound < upper bound, removing redundant constraints and iterating.

Thus we obtain relations involving $\pi_1, \pi_2, \ldots, \pi_m$ which are called *conditions for feasible experience* by Boole. Moreover, these relations involve π_{m+1} which gives the best possible bounds on the probability of the additional logical sentence.

This is known as the solution to Boole's *general problem.* Let us see an example to solve a problem by using the above steps 1 to 4.

Example 1 (Boole [6]) Find the best possible bounds on the probability of $S_6 = x_3$ considering the following:

$$\begin{aligned} prob(S_1 \equiv x_1) = &\quad \pi_1 \\ prob(S_2 \equiv x_2) = &\quad \pi_2 \\ prob(S_3 \equiv x_1x_3) = &\quad \pi_3 \\ prob(S_4 \equiv x_2x_3) = &\quad \pi_4 \\ prob(S_5 \equiv \overline{x}_1\overline{x}_2x_3) = 0. & \end{aligned}$$

Step 1, gives

$$\begin{aligned} x_1 = &\quad x_1x_2x_3+x_1x_2\overline{x}_3+x_1\overline{x}_2x_3+x_1\overline{x}_2\overline{x}_3 \\ x_2 = &\quad x_1x_2x_3+x_1x_2\overline{x}_3+\overline{x}_1x_2x_3+\overline{x}_1x_2\overline{x}_3 \\ x_1x_3 = &x_1x_2x_3+x_1\overline{x}_2x_3 \\ x_2x_3 = &x_1x_2x_3+\overline{x}_1x_2x_3. \end{aligned}$$

Set $p_1 = prob(x_1x_2x_3)$, $p_2 = prob(x_1x_2\overline{x}_3)$, $p_3 = prob(x_1\overline{x}_2x_3)$, $p_4 = prob(x_1\overline{x}_2\overline{x}_3)$, $p_5 = prob(\overline{x}_1x_2x_3)$, $p_6 = prob(\overline{x}_1x_2\overline{x}_3)$, $p_7 = prob(\overline{x}_1\overline{x}_2x_3)$, $p_8 = prob(\overline{x}_1\overline{x}_2\overline{x}_3)$. Step 2 yields the following equalities and inequalities:

$$
\begin{array}{llll}
p_1 + p_2 & +p_3 + p_4 & & = \pi_1 \\
p_1 + p_2 & & +p_5 + p_6 & = \pi_2 \\
p_1 & +p_3 & & = \pi_3 \\
p_1 & & +p_5 & = \pi_4 \\
 & & & +p_7 \quad = 0 \\
p_1 + p_2 & +p_3 + p_4 & +p_5 + p_6 & +p_7 + p_8 = 1 \\
p_1, p_2, \ldots, p_8 & \geq 1 & &
\end{array}
$$

Eliminating successively the variables $p_7, p_4, p_3, p_6, p_5, p_1$ and p_2 yields, at the end of Step 3, the bounds

$$\max\{\pi_3, \pi_4\} \leq \pi_6 \leq \min\{1-\pi_1+\pi_3, \pi_3+\pi_4, 1-\pi_2+\pi_4\}$$

and the consistency conditions

$$\pi_1 \leq \pi_3 \qquad \pi_2 \geq \pi_4.$$

Eliminating π_6 yields the additional condition

$$\pi_1 - \pi_3 + \pi_4 \leq 1.$$

Hailperin [20] extended and provided a systematic treatment for Boole's algebraic method to deal with conditional probabilities. Hailperin showed [20] that, the extensions can be

done along two directions, either using conditional probability in the objective function or using conditional probability in the constraints.

3.3.2 Polyhedral method

Polyhedral method has been contrived for obtaining an analytical solution of probabilistic logic problem which is different from Fourier-Motzkin elimination. It is based on the study of dual polyhedra of (7) as expressed by the following two linear programs (LP^1, LP^2):

$$LP^1 = \begin{cases} \min\ y_0 + \pi y \\ \text{subject to: } \mathbb{1}y_0 + A^t y \geq A^t_{m+1}. \end{cases}$$

$$LP^2 = \begin{cases} \max\ y_0 + \pi y \\ \text{subject to: } \mathbb{1}y_0 + A^t y \leq A^t_{m+1}. \end{cases} \tag{21}$$

It can be observed that, the polyhedra defined by (21) includes the vector $(1, 0)\big((0,0)\big)$ respectively, therefore, the corresponding polyhedra are non empty. In order to obtain the condition under which the probabilities are consistent, consider the dual of the probabilistic satisfiability problem in decision form (4), with a dummy objective function, $0p$, to be maximized:

$$\begin{aligned} &\min \quad y_0 + \pi y \\ \text{subject to:} \quad &\mathbb{1}y_0 + A^t y \leq 0. \end{aligned} \tag{22}$$

Hansen, Jaumard and Poggi de Arago [24] showed that:

Theorem 1. *The probabilistic satisfiability problem (4) is consistent if and only if*

$$(1, \pi)^t r \leq 0 \tag{23}$$

for all extreme rays r of (22).

Using the duality theorem of linear programming, Hailperin [19] showed that:

Theorem 2. *The best lower bound for π_{m+1} is given by the following convex piecewise linear function of the probability assignment:*

$$\underline{\pi}_{m+1}(\pi) \quad = \quad \max_{j=1,2,\ldots,k_{max}} \quad (1, \pi)^t y^j_{max} \tag{24}$$

where $y^j_{\max}$ for all j represent the k_{max} extreme points of (21).

The best upper bound for π_{m+1} is given by the following concave piecewise linear function of the probability assignment:

$$\overline{\pi}_{m+1}(\pi) \quad = \quad \min_{j=1,2,\ldots,k_{min}} \quad (1, \pi)^t y^j_{min} \tag{25}$$

where $y^j_{\min}$ for all j represent the k_{min} extreme points of (21).

This result provides best possible bounds on $\underline{\pi}_{m+1}$ and $\overline{\pi}_{m+1}$. Both Theorems 1 and 2 readily extend to the case of probability intervals (10) but not to the case of conditional probabilities [23]. Here we present examples for the theorem 1 and theorem 2 on small logical systems.

$$\begin{array}{ll} x_1 & [\underline{\pi}_1, \overline{\pi}_1] \\ S : x_1 \vee x_2 & [\underline{\pi}_s, \overline{\pi}_s] \\ x_2 & \pi_2? \end{array}$$

The upper and lower bound is given by the analytical expressions:

$\overline{\pi}_2 = \min\{1, \overline{\pi}_s\}$ and $\underline{\pi}_2 = \max\{0, (\underline{\pi}_s - \underline{\pi}_1)\}$.

The consistency condition is enforced by

$$0 \leq \underline{\pi}_i \leq 1; \quad \underline{\pi}_i \leq \overline{\pi}_i \quad i = 1, S,$$

and

$$\overline{\pi}_s \geq \underline{\pi}_1.$$

3.4 Numerical Solution for PPL

Numerical methods are needed to assess the solution of the *Propositional Probabilistic Logic* problem (or PPL for short) in practice as analytical solution can be used only for very small instances. Numerical solution methods for P^{PSAT} and P^{PENTAIL} can be categorized into two types namely, *exact* and *heuristic* methods. Let us briefly describe what are these exact and heuristic methods.

Table 1: Examples of analytical solutions and consistency condition for small logical systems.

Logical Systems	Probability Assigned	Consistency Conditions	Probability Bounds
$S: x_1 \vee x_2$ x_1 x_2	$[\underline{\pi}_S, \overline{\pi}_S]$ $[\underline{\pi}_1, \overline{\pi}_1]$ π_2 ?	$\underline{\pi}_i \leq 1 \quad i = 1, S$ $\overline{\pi}_i \geq 0 \quad i = 1, S$ $\underline{\pi}_i \leq \overline{\pi}_i \quad i = 1, S$ $\overline{\pi}_S \geq \underline{\pi}_1$	$\underline{\pi}_2 = \max\{0, \underline{\pi}_S - \overline{\pi}_1\}$ $\overline{\pi}_2 = \min\{1, \overline{\pi}_S\}$
$S: x_1 \vee x_2 \vee x_3$ x_1 x_2 x_3	$[\underline{\pi}_S, \overline{\pi}_S]$ $[\underline{\pi}_1, \overline{\pi}_1]$ $[\underline{\pi}_2, \overline{\pi}_2]$ π_3 ?	$\underline{\pi}_i \leq 1 \quad i = 1, 2, S$ $\overline{\pi}_i \geq 0 \quad i = 1, 2, S$ $\underline{\pi}_i \leq \overline{\pi}_i \quad i = 1, 2, S$ $\overline{\pi}_S \geq \underline{\pi}_2$ $\overline{\pi}_S \geq \underline{\pi}_1$	$\underline{\pi}_3 = \max\{0, \underline{\pi}_S - \overline{\pi}_1 - \overline{\pi}_2\}$ $\overline{\pi}_3 = \min\{1, \overline{\pi}_S\}$
$S: x_1 \vee x_2 \vee x_3 \vee x_4$ x_1 x_2 x_3 x_4	$[\underline{\pi}_S, \overline{\pi}_S]$ $[\underline{\pi}_1, \overline{\pi}_1]$ $[\underline{\pi}_2, \overline{\pi}_2]$ $[\underline{\pi}_3, \overline{\pi}_3]$ π_4?	$\underline{\pi}_i \leq 1 \quad i = 1, 2, 3, S$ $\overline{\pi}_i \geq 0 \quad i = 1, 2, 3, S$ $\underline{\pi}_i \leq \overline{\pi}_i \quad i = 1, 2, 3, S$ $\overline{\pi}_S \geq \underline{\pi}_1, \overline{\pi}_S \geq \underline{\pi}_2$ $\overline{\pi}_S \geq \underline{\pi}_3$	$\underline{\pi}_4 = \max\{0, \underline{\pi}_S - \overline{\pi}_1 - \overline{\pi}_2 - \overline{\pi}_3\}$ $\overline{\pi}_4 = \min\{1, \overline{\pi}_S\}$
$S_1: \overline{x}_1 \vee x_2$ $S_2: x_1 \vee \overline{x}_2$ x_1 x_2	$[\underline{\pi}_{S_1}, \overline{\pi}_{S_1}]$ $[\underline{\pi}_{S_2}, \overline{\pi}_{S_2}]$ $[\underline{\pi}_1, \overline{\pi}_1]$ π_2?	$\overline{\pi}_i \geq 0 \quad i = 1, S_1, S_2$ $\underline{\pi}_i \leq \overline{\pi}_i \quad i = 1, S_1, S_2$ $\underline{\pi}_i \leq 1 \quad i = 1, S_1, S_2$ $\underline{\pi}_1 \leq \overline{\pi}_{S_2}$ $\overline{\pi}_1 + \overline{\pi}_{S_2} \geq 1$ $\overline{\pi}_{S_1} + \overline{\pi}_{S_2} \geq 1$	$\underline{\pi}_2 = \max\{1 - \overline{\pi}_{S_2}, \underline{\pi}_1 + \underline{\pi}_{S_1} - \overline{\pi}_{S_2}, \underline{\pi}_1 + \underline{\pi}_{S_1} - 1\}$ $\overline{\pi}_2 = \min\{\overline{\pi}_{S_1}, \overline{\pi}_1 + \overline{\pi}_{S_1} - \underline{\pi}_{S_2}, 1 + \overline{\pi}_1 - \underline{\pi}_{S_2}\}$

3.4.1 Exact numerical methods for PPL

An *exact* algorithm guarantees to find the best optimal solution if there exists one whereas, a *heuristic* algorithm usually only finds a good or near optimal solution without necessarily any information on how far it is from the optimal solution. However, it may take too much time to solve large sized PPL problems using an exact method. Therefore, one might need to use a heuristic method in order to find a solution in a reasonable amount of computing time.

In the subsequent sections, we will discuss about the *linear programming* solution with simplex or revised simplex algorithm ([11], p.97) and the linear programming solution with the *column generation* technique ([11], p.195), both of them correspond to an exact method. Next, AD-SOLPPL or AD-SOLFOPL (based on anytime deduction) and *variable neighborhood*

search which fit into the category of heuristic solution methods are also discussed. Linear programming and column generation methods can solve PPL problem comprised of instances of conditional and unconditional probabilities.

Linear programming modeling

It has been already shown in Chapter 2 that the probabilistic satisfiability or the probabilistic entailment problem can be reformulated as linear programs. The simple method for solving (4), (7) and (10) problems include firstly searching for the overall set of possible worlds (see Chapter 2), and secondly, solving the resulting linear program. We can solve the linear program by using simplex or revised simplex algorithm [11]. Although simple, this method can be used only for small instances since the size of the input grows exponentially with the increase of the number of logical variables $x_1, x_2, \ldots, x_n$. In view of the enormous size of these programs (about 10^{18} columns for $\min\{m, n\} = 60$, where m is the number of sentences and n is the number of logical variables in these sentences), it is impossible to solve large sized probabilistic satisfiability or probabilistic entailment problem using this method. This led Nilsson [51] to suggest looking for heuristic solution methods only. Such a view is overly pessimistic as we will see in the next section.

Column generation solution

When linear programs cannot be solved using the simplex or revised simplex algorithm, tools from large scale optimization come to rescue. In operations research, there exist some tools, namely the column generation technique which deals with solving linear programs with a huge number of variables. In this section, we will see how column generation technique works using a description from [31].

Column generation starts by solving a small, manageable part of a problem (few of the possible worlds). By analyzing this partial solution it determines an additional small subset of the worlds (one or more possible worlds) to be added in the model, and then solves the enlarged model. This column-wise modeling repeats this process until it satisfies an optimality condition for the problem. More formally, column generation technique extends the revised simplex algorithm (see e.g., Chvatal [11], p.97), in which only a small number of columns are kept explicit, by determining the entering column through the solution of an auxiliary subproblem.

To describe the general principle of column generation technique, let us assume, without the loss of generality, a linear program (26) for a minimization problem in order to get the lower bound in the optimal solution:

$$\min \{cx : Ax \geq b,\ x \geq 0\} \tag{26}$$

The coefficients of variables x of the equations of (26) form the columns of the matrix A. Therefore, variables and columns are used with the same meaning. By using either the simplex algorithm or the column generation technique, the optimal solution (which provides a lower bound) of equation (26) can be obtained. However, when the number of variables is very large, it is better to use the column generation methods.

In column generation methods, the initial linear program is decomposed into a restricted linear program and a pricing problem. The restricted linear program is called the *restricted master* problem which corresponds to a linear program associated with a restricted matrix A' such that, A' is a sub-matrix of A. In principle, if there exists an optimal solution for the

restricted linear program, it may also be an optimal solution for the initial linear program. However, it depends on the signs of the reduced costs of the missing columns, i.e., columns which are not explicitly considered in (26).

Column generation methods proceed as follows: the linear program corresponding to a sub-matrix $A^1(= A')$ of dimension $m \times n_1$ of the original matrix is solved by using the simplex algorithm, and a feasible solution x^1_{LP} is obtained. Note that, the cost of the columns of A^1 are denoted by c^1 which is a sub-vector of c.

$$\begin{cases} \min\ c^1 x \\ \\ A^1 x \geq b \\ \text{subject to: } 0 \leq x \leq 1 \\ \\ x \in \mathbb{R}^{n_1}. \end{cases} \tag{27}$$

Does there exist a column $a^j \in A \backslash A^1$ such that $\overline{c}_j < 0$? i.e., can we find a column of the matrix $A \backslash A^1$ for which the reduced cost is negative? If the answer is *no*, then the feasible solution x^1_{LP} is optimal for (26). If the answer is *yes*, we must add one or more columns a^js to the matrix A^1 in order to find the optimal solution. Therefore, we need to solve the following system

$$\begin{cases} \min\ c^2 x \\ \\ A^2 x \geq b \\ \text{subject to: } 0 \leq x \leq 1 \\ \\ x \in \mathbb{R}^{n_2}. \end{cases} \tag{28}$$

with $A^2 = A^1 \cup \{a^j\}$ such that $\overline{c}_j < 0$.

This process is repeated until a column j with a negative reduced cost is found, i.e., $\overline{c}_j \leq 0$. However, if no more iteration is possible, we conclude that we have obtained an optimal solution for equation (26).

Now in order to find if there exists a column with a negative reduced cost one has to solve the so-called *pricing problem.* Considering the master problem, e.g., (27), let us assume that we want to find if there exists a column a^j with a negative reduced cost. So, we must solve the pricing problem according to:

$$\begin{cases} \min\ \overline{c}(a^j) \\ \text{with constraints on the components of } (a^j) \\ \text{in order to guarantee that} A^2 \subseteq A, \end{cases} \tag{29}$$

where A^2 is the concatenation of A^1 and $a^j : A^2 = (A^1 \mid a^j)$.

For the master problem, the reduced cost is defined in the matrix form as follows:

$$\overline{c} = c - vA$$

where v is the vector of the dual variables associated with equation (26). For the master problem, we obtain:

$$\overline{c}_j = c_j - v^1 a^j = c_j(a^j) - v^1 \cdot a^j$$

for the column j where $\overline{c}_j$ is the reduced cost of the column j, v^1 is the optimal dual vector obtained when solving (27).

Solving the pricing problem is equivalent to solving the following problem:

$$\overline{c}^* = \min\{\overline{c}(a) = c(a) - va : a \in \mathcal{A}\},$$

where $\mathcal{A} = \{a \in \mathbb{R}^m : (A^1 \mid a)$ is a sub-matrix of $A\}$ and $\overline{c}^*$ is the best known reduced cost.

The column generation process is summarized in Table 2.

Table 2: Column generation process

Identify an initial set of (artificial) columns
Solve the restricted master problem
WHILE (there exists a column a_j with a negative reduced cost as a solution of the pricing problem
  Include a^j to the restricted master problem
  Solve the restricted master problem
  Solve the pricing problem

The pricing problem (29) is often a NP-complete problem which is very difficult to solve. However, it is not mandatory to solve the pricing problem exactly at each iteration. In order to ensure an iteration of the revised simplex algorithm to take place, it is enough to find a column with a negative reduced cost. Therefore, a heuristic algorithm can be designed for finding such a column. If a feasible solution (i.e., a column with negative reduced cost) is obtained heuristically, the decision form of the probabilistic logic problem is solved. However, finding no feasible solution by choosing the entering column in a heuristic way cannot guarantee that none exists. Therefore, when no more column with a negative reduced cost is obtained heuristically, it is necessary to turn to an exact algorithm either to prove that, there is no feasible solution for the decision form of the probabilistic logic problem or to prove that, there exists no feasible solution which gives better bounds than the incumbent one for the optimization form of probabilistic logic problem.

In order to start the column generation technique either a set of artificial columns or a feasible set of columns is needed. Usually, it starts with an artificial solution which can be generated in an insignificant time. The set of artificial columns corresponds to a set of columns (tiny compare to the number of variables), which constitute a square matrix as large as the number of constraints. It is to be noted that, in order to minimize the objective function, the algorithm needs a negative reduced cost, but if the aim is to maximize the objective function, then a positive reduced cost is required.

3.4.2 Heuristic numerical methods for PPL

Exact methods are accurate but at the cost of possibly long computation times. That is why alternative solution methods have been explored and suggested by several authors.

Anytime deduction technique.

One of the pioneer works in this ground (for the probabilistic logic problem) for both unconditional and conditional cases of probabilities (point and interval cases) is due to Frisch and Haddawy [15]. Their procedure takes a set of propositional sentences associated with their probability values and a query (either an atomic propositional sentence or conjunction of them) as input. Propositional sentences of the form $(S_j|S_i)[\overline{\pi}_{S_j}, \underline{\pi}_{S_j}]$ are expressed with conditional probability while for unconditional probability it reduces to $(S_i|T)[\overline{\pi}_{S_i}, \underline{\pi}_{S_i}]$ taking an ever-true value for S_j. The anytime deduction approach proceeds by computing increasingly narrow probability intervals for the query (i.e., the additional sentence) that contain the tightest entailed probability interval. They call this approach "anytime deduction" as the process can be stopped at any time and approximate information on the tightest probability interval of an additional sentence can be yielded. The principal focus of

their work is not necessarily on finding the optimal solution, instead it tries to find a near optimal solution (which gives an estimation of the complete solution) very quickly. This is good specially for very large instances to stop anytime and get the information how the probability bound is achieved through the deductive process.

The authors claim that the heuristic solution developed by their procedure is valid but may be not very precise (i.e., not the tightest probability bounds) even on very small instances. They have used a set of sound (not complete) inference rules to reach the solution. Still the authors do not provide an explicit and well-defined procedure to perform the probability interval tightening. Moreover, the consistency issue is not considered, i.e., the instances are not checked for consistency before proceeding with their anytime deduction method. For instance, Hansen *et al.* [25] show that one of the examples considered by Frisch and Haddawy is indeed inconsistent although their procedure does not detect it.

Yet this gives a good approximation on the bound of probability in a small amount of time. They also suggested for an extension in first order logic but did not provide any clear direction.

The procedure starts with a probability interval [0,1] which they called current derived interval and proceeds by applying the inference rules in order to find a better probability bound. Suppose the procedure has obtained a current probability bound $[\underline{\pi}_i, \overline{\pi}_i]$ for sentence S_i and it finds a new interval $[\underline{\pi}_i^{new}, \overline{\pi}_i^{new}]$ where at least one of the the new probability bounds is better than the current one, the value for the current probability bound will be updated applying the rule for intersection of the bound intervals. The main focus of their work is on the anytime nature, which says that at anytime the process can be stopped and a correct value can be obtained together with how the result is obtained. The authors also

states that the aim of the process is to direct the interval tightening in such a direction such that it always keeps track of the tightest entailed probability intervals. This concept is similar to that of the inference rules presented in ADPSAT [32] by Jaumard *et al.*.

Constraint propagation with imprecise conditional probability .

Though Frisch and Haddawy [15] discussed both conditional and unconditional probabilities, a stronger focus on conditional probabilities is due to Amarger, Dubois and Prade [1]. They propose a constraint propagation algorithm, similar to anytime deduction type. Similar to Frisch and Haddawy they proposed two local inference rules. But they have considered the consistency issue and checked it at the beginning with help of linear programming as similar to that of [50], see also Section 2.1. Then they try to gather as much information as possible about the given system before applying the inference rules for the entailment problem. They call it information gathering process. As compared to the linear programming tools, they are considered more flexible and scalable. But they are somewhat restricted to particular types of systems in order to apply the information gathering of the network successfully.

Their process works as follows: The knowledge base, which they call network, is a collection of general statements involving a group of general objects while these objects are set of logical terms. The network is presented only in terms of conditional probabilities. At this point, it is worth mentioning that the input is a set of sentences such that for some of the sentences both $\pi_{(S_i|S_j)}$ and $\pi_{(S_j|S_i)}$ forms exist. In order to work with the unconditional probability the statements are represented in the form $\pi_{(S_i|X)}$ where X is the set of logical terms which corresponds to ever-true propositions or tautologies. Next, their procedure checks the consistency of the network globally by linear programming. If the network is

inconsistent, it quits the procedure, otherwise it uses the first set of inference rule on the network which the authors named Quantified Syllogism, QS for short. The purpose of this rule is to gather as much information as possible about a given query. This gathering process is done by repeatedly applying the Quantified Syllogism inference rule to get the missing information. For example assuming that the probability bounds for $\pi_{(S_i|S_j)}$, $\pi_{(S_j|S_i)}$, $\pi_{(S_k|S_j)}$ and $\pi_{(S_j|S_k)}$ are given, applying the Quantified Syllogism inference rule in order to try to get either first probability bounds on $\pi_{(S_k|S_i)}$, or tighter bounds.

The lower bound given by the QS inference rule is as follows :

$$\underline{\pi}(S_k|S_i) = \underline{\pi}(S_j|S_i)\max\{0, 1 - \frac{1 - \underline{\pi}(S_k|S_j)}{\underline{\pi}(S_i|S_j)}\}$$

whereas the upper bound is calculated as:

$$\overline{\pi}(S_k|S_i) = \min\{1, 1 - \underline{\pi}(S_j|S_i) + \frac{\underline{\pi}(S_j|S_i).\overline{\pi}(S_k|S_j)}{\underline{\pi}(S_i|S_j)},$$
$$\frac{\overline{\pi}(S_j|S_i).\overline{\pi}(S_k|S_j)}{\underline{\pi}(S_i|S_j)\underline{\pi}(S_j|S_k)}, \frac{\overline{\pi}(S_j|S_i).\overline{\pi}(S_k|S_j)}{\underline{\pi}(S_i|S_j)\underline{\pi}(S_j|S_k)}[1 - \underline{\pi}(S_j|S_k)] + \overline{\pi}(S_j|S_i)\ \}.$$

Next the inference rule named Generalized Bayes theorem is applied on all the gathered information (probability values) which assumes that unconditional probabilities like $\pi(S_i)$ or $\pi(S_j)$ are not known in advance. In order to calculate the final probability bound for the query, maximum value of the lower bound and the minimum value of the upper bound is taken into consideration. Their proposed inference procedure is polynomial in the number of nodes in a network but it takes most of its time to gather the information about the query from the network. Finally, the process stops when there is no more improvement in the probability bounds otherwise it repeats from the step of applying Quantified Syllogism rule. Although they mention extension to first order, the method proposed by Amarger *et al.* [1] does not provide any (explicit) guideline for first order probabilistic logic.

Probabilistic logic programming and local probabilistic deduction techniques over events of different types.

Similar to Halpern's [22] semantics to formulas that describe *degrees of belief*, Lukasiewicz [42] proposed an approach called *probabilistic logic programming* where uncertainty is handled by assigning probability distribution over the set of possible worlds (i.e., same as [50], in Section 2.1). In order to deduce a tight probability bounds for an additional clause (i.e., similar to an additional sentence), they proposed two solutions. As a first solution, Lukasiewicz *et al.* [42] showed that it is possible to compute the tight bounds by using straightforward linear program (i.e., same as [50] discussed in Section 2.1). However, they also proved that when the number of variables (or possible worlds) increases, the solution grows exponentially. Therefore, in order to handle this difficulty, Lukasiewicz *et al.* [42] proposed another solution technique to generate linear programs that generally have a much lower number of sentences as well as variables. For this purpose (i.e., to lower the number of sentences), they partitioned the probabilistic logic program ($\mathcal{P}$) into a set of *logical program* clauses (pure propositional sentences) ($\mathcal{L}$) and a set of purely *probabilistic program* clauses (propositional sentences having conditional probabilities) ($\mathcal{P}\backslash\mathcal{L}$). As for example, $(S_j|S_i)[\underline{\pi}_{S_j}, \overline{\pi}_{S_j}]$; $[\underline{\pi}_{S_j}, \overline{\pi}_{S_j}] \in [0,1]$; $[\underline{\pi}_{S_j} > 0, \overline{\pi}_{S_j} < 1]$; $S_i \neq true$, belongs to the later group whereas propositional sentences are of the form $S_i \rightarrow S_j$ without associated probability interval.

Now the number of generated possible worlds is reduced in size and finally the probabilistic deduction problem is solved using two linear programs (for calculating the upper and lower bounds). The authors claim that this linear programming approach for probabilistic deduction shows efficient result only in restricted cases.

Lukasiewicz *et al.* proposed a set of inference rules which they call magic inference rules for probabilistic deduction in [41]. Here, they presented four groups of inference rules namely: Sharpening, Chaining, Fusion and Combination which works in a similar fashion (but not same) as we already found in [1]. They also claimed that those inference rules are, what they called, locally complete and sound. In contrast to the approaches like linear programming (e.g., of Nilsson's approach [50] see also in Section 2.1), which are globally complete (i.e., able to find the tightest probability bound if there is one), their approach is able to compute tightest bounds but only for a maximum of four probabilistic formulas. The main focus of this work is on the proposed inference rules and the procedure to apply them for restricted cases, there is no algorithm presented here. They did not discuss about any extension of their work of [41] to first order logic.

Two more works due to Lukasiewicz in this sequence are published in [44] [43]. In [44], the author proposes a polynomial time probabilistic deduction procedure for a particular case, i.e., conditional constraint trees which are indirected trees with basic events as nodes and with bidirectional conditional constraints over basic events as edges between the nodes. The input is a form of graph consisting of propositional sentences of the form $(S_j|S_i)[\overline{\pi}_{S_j}, \underline{\pi}_{S_j}]$ and $(S_i|S_j)[\overline{\pi}_{S_i}, \underline{\pi}_{S_i}]$ having edges $[S_j \rightarrow S_i,\ S_j \leftarrow S_i]$ and from this they form a conditional constraint tree by taking a particular direction. But Lukasiewicz did not provide any algorithm for this procedure in [44]. Initially, the conditional constraint tree is represented by a set of basic propositional events and conditional probabilities of the form $(S_j|S_i)[\overline{\pi}_{S_j}, \underline{\pi}_{S_j}]$ with real numbers $[\overline{\pi}_{S_j}, \underline{\pi}_{S_j}] \in [0, 1]$ and basic propositional sentences S_i and S_j. The author has provided result only for the local deduction method to compute bounds on the query.

In [43], Lukasiewicz presents another linear time probabilistic deduction scheme for one more particular case, namely, taxonomic and probabilistic knowledge bases over conjunctive events. In this work he proposed a general algorithm which integrates a whole set of inferences rules (with conditional probabilities) and which may detect inconsistency. The inference rules are applied on taxonomic and probabilistic formulas over propositional events. Here taxonomic formulas are expressions of the kind $S_i \rightarrow S_j$ with propositional events S_i and S_j and probabilistic formulas are expressions of the form $(S_j|S_i)[\overline{\pi}_{S_j}, \underline{\pi}_{S_j}]$ with real numbers $[\overline{\pi}_{S_j}, \underline{\pi}_{S_j}] \in [0,1]$ and propositional events S_i, S_j. The inference rules applied are very complex for scalability as the author claims. In [43] Lukasiewicz mentions that it improves on previous works but he did not provide proof with any numerical results comparing his algorithm with global approaches using linear programming tools.

Variable neighborhood search.

Variable Neighborhood Search, or VNS in short, was introduced by Mladenović and Hansen [46]. VNS is a meta-heuristic which helps escaping when the search process is trapped in a local optimum by changing the neighborhood structures systematically. Initially, a set of neighborhood structures is preselected, a stopping condition is determined and an initial local solution is found. In the main loop of the method, it searches for a better solution by changing the neighborhood structures using three process (i) shaking, (ii) local search and (iii) move or not. It moves to a new solution from the current best known solution if the new one is better or it may accept a worse solution with certain probability in order to escape the local optima and move toward the global optima.

For solving the probabilistic logic problem, Jovanović *et al.* [34] suggested using VNS based heuristic. Hansen and Perron [26] use VNS to generate multiple columns (or possible

worlds [50]) with negative reduced cost simultaneously at each iteration of the column generation method. Adding multiple columns at each iteration instead of only one column reduces the number of iteration and speeds up the algorithm [26].

ADPSAT

A more generalized approach of explicit deductive nature has been proposed by Jaumard *et al.* [32] called ADPSAT, using their previous work [24] on the analytical solution of Nilsson's [50] probabilistic logic problem. In ADPSAT, only a small subset of logical sentences (typically one or two) together with the interval probability values of a small subset of variables (typically one to four) are considered at each step to determine the probability interval values of either a selected variable or a selected sentence. Afterwords, another subset of sentences is considered to compute the probability interval values and the newly computed probability value is compared with the previously computed probability values. By repeating this deductive mechanism, the final tight probability interval value is determined.

ADPSAT is capable of checking inconsistency of a given set of sentences very quickly but it does not always guarantees to do so. However, in practice, ADPSAT is able to find very often the tightest probability bounds, for instance, usually when the sentences contain two variables.The entailment is achieved using a sequential deductive approach. In their approach [32], the authors have used an ordered set of well thought combinations of small number of sentences and variables which they call *primitives*. Actually these are a set of sound inference rules to be applied in order to find the probability bounds. Finally, ADPSAT has solved, by far, some of the largest instances with 1000 variables and 2500 sentences for a reasoning under uncertainty model based on probability theory.

3.5 Numerical Solution for FOPL

3.5.1 Exact numerical methods for FOPL

A numerical solution scheme for First Order Probabilistic Logic using column generation methods has been proposed by Sultana [63] in her Master's thesis, see also [29]. In this work, a mathematical modeling for the first-order probabilistic logic problem or FOPL is provided. This is originally arisen from the mathematical model for propositional probabilistic logic [30] or PPL, based on a column generation formulation. While in propositional calculus, the underlying considerations are only with propositional variables and symbols, in first-order logic, it is mandatory to handle quantifiers, predicates and functions. Therefore, [63] claimed it to be more difficult to provide a mathematical model for first-order probabilistic logic than for propositional probabilistic logic.

Based on their mathematical model, they have developed a solution scheme for the FOPL mathematical model. While the model could be solved by linear programming tools using the simplex or revised simplex algorithms [11], the approach becomes limited to very small sized instances as the number of variables grows exponentially with the number of predicates. However, as for PPL, by using column generation techniques (see Chvatal [11], p. 195), these limitations can be overcome. Moreover, it is much faster than the classical simplex algorithm for solving linear programs since it is not required to consider explicitly all the possible worlds to guarantee an optimal solution. Just to recall that, by a *possible world* they mean a truth assignment on the set of sentences such that the set of sentences is satisfiable for that truth assignment. In the column generation model a possible world corresponds to a column in the matrix. Column generation technique begins with taking a

small, manageable set of columns (a few of the possible worlds), solves the associated LP, then analyzes its partial solution to determine an additional small subset of worlds (one or more possible worlds) to be added to the model. Each time there is an addition of a column or set of columns, then the procedure again solves the expanded model. The column-wise modeling repeats this process until it satisfies an optimality condition for the problem.

From the above discussion it's obvious that column generation technique does not solve the whole problem at a time as does the straight forward LP solution scheme. Instead the column generation method relies on a decomposition of the initial linear program into a *master* and a *pricing* problem. At each iteration, we consider only a *restricted master problem*, i.e., the master problem with only a small number of variables, i.e., a small subset of possible worlds. The *pricing problem* corresponds to the mechanism of generating possible worlds which improve the value of the current solution and which are added to the restricted master problem. Therefore, in order to solve the FOPL problem, an algorithm has been provided for both the restricted master problem as well as for the pricing one.

Usually solving the *restricted master problem* is easy but solving the *pricing problem* is more difficult because of its nature. The pricing problem is an optimization problem. Therefore, it is defined by an *objective* and a *set of constraints*. The *objective* of the pricing problem is to find a world with an appropriate (negative or positive) sign for the reduced cost such that it will improve the current solution(provided by their restricted master problem). The *reduced cost* is an indication (metric) that is used to check the optimality criteria of a solution in LP [11]. The *set of constraints*, in the pricing problem, corresponds to the set of rules associated with the definition of a possible world. In PPL, a world is a possible or valid truth assignment over the sentences, i.e., it is possible if there exists a truth assignment on

the variables that leads to that truth assignment on the sentences. Finding a possible world is easy in PPL, indeed, the truth assignment or number of interpretations of a propositional sentence is finite. Whereas, in FOPL, it is more difficult to obtain a possible world (see Chang and Lee ([9], p. 31) for more information).

The basic definition of the *satisfiability* problem for first-order logic and propositional calculus is same whereas, the definition of *interpretation* is different for them. However, [63] has made use of the idea that finding a feasible solution of the pricing problem can be reduced to the well known *satisfiability* problem of first-order logic in AI.

In order to solve this satisfiability issue, Jaumard *et al.* [29] and Sultana [63] have used a theorem prover package named *Theo-2006*. This package reaches a decision by searching for contradiction or inconsistency rather than satisfiability using the *resolution refutation* principle (see [48] for more information). However, addressing the satisfiability issue is just only a part of the pricing problem, corresponding to the search of a feasible solution. Each time the pricing problem runs with the objective of finding a negative reduced cost in order to reach the optimal solution. The issue of finding the negative reduced cost (i.e., for minimization problem) has been addressed by employing the well-known branch-and-bound technique. Although the branch-and-bound technique is an exact method (i.e., the solution is found if there exist one), this can be used within a heuristic by stopping it as soon as a node with negative reduced cost is identified.

Finally they have devised an algorithm for solving the *pricing problem* which is an *exact* algorithm. It is to be noted that, an exact algorithm is one which guarantee to find the optimal solution if there exists one. They claim their algorithm is scalable as it is comfortable to solve medium sized first-order instances. By far it is the first exact algorithm

for solving the first-order probabilistic logic problem.

3.5.2 First order strategies

In this section we describe some essential strategies for first order probabilistic logic. In the first part we discuss some practical and scalable procedures, i.e., regarding satisfiability issues used in some well known theorem provers. In the second part, we discuss their special strategies behind success.

3.5.2.1 Practical and scalable procedures for satisfiability issues in first order logic

Although the satisfiability problem, i.e., SAT problem, is associated with PPL, in first-order logic, automated theorem provers are used to prove the satisfiability of a set of formulas. In order to serve the purpose of checking satisfiability of a set of given first-order formulas, we were searching for a suitable and efficient theorem-prover. We decide to focus on the theorem provers which participated in the CADE ATP System Competition. Among them, Vampire, SPASS, Theo are the most well reputed theorem provers in the international competitions. Next, we go through their underlying assumptions and strategies.

Vampire uses saturation with resolution and paramodulation; SPASS combines saturation with superposition (a variant of demodulation), conventional splitting with branching and backtracking; and Theo uses resolution refutation.

We have already discussed *refutation* proving, some strategies and simplification methods for *resolution* proving in Section 2.2.1.4. In order to make resolution more efficient, several sophisticated *inference rules* have also been developed and applied in the efficient

theorem provers. In this section, we will discuss two of such techniques: *paramodulation* and *saturation*. Moreover, some recent adaptive strategies that are used to improve the performance of resolution based first-order theorem provers are also discussed in the last section.

3.5.2.2 Adaptive strategies

Several strategies are used to enhance the performance of first-order theorem provers. Among them **Limited Resource Strategy** (LRS) of Riazanov and Voronkov [54, 56] greatly improves the effectiveness of saturation algorithm. This strategy addresses the problem of reasoning in limited time. According to the authors, LRS is an adaptive strategy as it can dynamically adjusts the limit on some weight of clauses. This dynamic adjustment is based on the collected statistics on the earlier stages of proof searching.

The cost of backtracking is usually very high as it may contain several hundreds or thousands of clauses, literals. In order to prevent this problem, Riazanov and Voronkov suggested another strategy **Splitting Without Backtracking** (SWB) in [55]. The SWB is an intelligent backtracking that contains the splitting history. Therefore, by analysing the history, it is easy to identify the path of splitting to reach a contradiction. For instance, consider, a set S of clauses where C_1 and C_2 are two new clauses. Now to refute the set $S \cup \{C_1 \vee C_2\}$ we can split it as $S \cup \{C_1\}$ and $S \cup \{C_2\}$ and check refutation.

Riazanov and Voronkov also showed in [57], the use of demodulation in two different modes, *forward* demodulation and *backward* demodulation. In forward demodulation, according to the unit equalities that already exist in the current clause set, newly derived clauses are re-written. While in the backward demodulation, old set of clauses is rewritten

based on the positive unit equalities of the newly derived clauses. One needs a technique to retrieve instances in order to re-write the subset of clauses with unit equalities. The authors [57] have also introduced a technique called **Path Indexing** for retrieval instances smoothly.

Chapter 4

An Anytime Deduction Algorithm for First Order Logic

In Chapter 2, we defined the model for probabilistic logic problem in the first order case. We recall from Chapter 3 that a numerical solution for this model can be achieved in two ways, namely: Exact methods and anytime deduction methods. An exact method is one which solves the problem in such a way that it guarantees to find a solution if there exists one, and guarantees to find one with the best possible objective value. On the other hand, an anytime deduction method can be stopped anytime during its execution and we get a partial result (i.e., a near optimal solution) at that point along with the explanation, (i.e., how the solution is reached as opposed to the exact methods).

4.1 AD-SOLFOPL: An Anytime Deduction Algorithm

In this chapter we describe an anytime deductive approach, called AD-SOLFOPL, for solving the first order probabilistic logic and entailment problem. Before going any further into the detail of the deductive procedure, let us recall the two forms of FOPL problem in brief. The first-order probabilistic logic or FOPL problem has been formally defined in Section 2.2.1. The *decision* form of FOPL (P^{FOSAT} for short), is defined as follows: Given a set $\mathcal{F}$ of m formulas in prenex normal form, and a probability vector $\pi = (\pi_1, \pi_2, \ldots, \pi_m)$ associated with these formulas, is the set $(\mathcal{F}, \pi)$ consistent? The *optimization* form of FOPL (P^{FOENTAIL} for short), formally defined in Section 2.2.2, can be stated as follows: Given a set $\mathcal{F}$ of m formulas in prenex normal form, and a probability vector π associated with these formulas, such that the set $(\mathcal{F}, \pi)$ is consistent, considering an additional formula F_{m+1}, what are the probability values for F_{m+1} to be true so that the overall system $(\mathcal{F} \cup F_{m+1}, \pi)$ remains consistent.

Now, the AD-SOLFOPL procedure is a sequential procedure in which, at each iteration, we examine the impact of a set of inference rules, i.e., a small subset of logical sentences (typically one to three) together with the interval probability values of a small subset of predicates, on the probability interval values of either a selected predicate or a selected sentence. In other words, at each iteration, it tries to tighten the probability interval of a predicate or of a sentence, and consequently defines a deductive approach corresponding to a sequential tightening procedure. We explain the outline of the AD-SOLFOPL procedure in the subsequent paragraphs but before moving to that part let us have a quick look into a special set of *inference rules*, a small but an essential concept for the overall AD-SOLFOPL procedure.

4.1.1 Set of inference rules

In the next two subsections we state a set of inference rules which plays a very important role for the anytime deduction algorithm. First, we explain the set of inference rules which were proposed by Jaumard *et al.* [33] for propositional case with the AD-PPL procedure. In the succeeding subsection we present the proposed set of inference rules adapted and extended for first order case from the one proposed by [33].

4.1.1.1 Set of inference rules for AD-PPL

Jaumard *et al.* [33] has defined a particular set of **inference rules**, which consists of a small set of logical sentences together with their probabilities for which the analytical solution is available. For example, let S_i be a logical sentence with two or three literals. Let x_k be one of the variables appearing in S_i, either as a positive literal (x_k) or a negative one ($\overline{x}_k$). The formation of the first *inference rule* is as follows: an analytical expression of the tightest possible probability interval for x_k or $\overline{x}_k$, given probability intervals for S_i and for all other literals involved in S_i.

The general idea behind this proposed set is that: At most two sentences are given with their associated probability bounds such that these sentences are true. Each sentence is represented as the combination of positive or negative literals of at most four distinguishable variables. In an inference rule, the variables which make up a sentence are either given explicit probability bounds or assume probability bounds of [0,1]. We have to find the probability intervals of an additional sentence having at most two literals either in positive or negated form. Now, let us consider the following example: A probabilistic sentence S, described in terms of the variables x_1 and x_2, is given with its probability bounds. One of

the literals of the sentence (i.e., x_1) is also provided with probability bounds; the inference rule is able to calculate the probability bounds on the literal (i.e., x_2) and check consistency of the calculated bounds.

$$\begin{array}{ll} x_1 & [\underline{\pi}_1, \overline{\pi}_1] \\ S : x_1 \vee x_2 & [\underline{\pi}_s, \overline{\pi}_s] \\ x_2 & \pi_2? \end{array}$$

The upper and lower bound is given by the analytical expressions:

$\overline{\pi}_2 = \min\{1, \overline{\pi}_s\}$ and $\underline{\pi}_2 = \max\{0, (\underline{\pi}_s - \underline{\pi}_1)\}$.

The consistency condition is enforced by:

$$0 \leq \underline{\pi}_i \leq 1; \quad \underline{\pi}_i \leq \overline{\pi}_i \quad i = 1, S,$$

and

$$\overline{\pi}_s \geq \underline{\pi}_1.$$

Other simple examples of inference rules from Jaumard *et al.* [33] are shown in the appendix. They are varied based on several criteria. Some of them have only one sentence defined in terms of positive literals only while others have both negative and positive literals in them. The size of a sentence varies from two to a maximum of four literals (both positive or negated form). A set of more complex inference rules include two logical sentences and the literals having sign of both type. The most complex set of inference rules contains query having at most two literals while simple queries have only one literal.

4.1.1.2 Set of inference rules for AD-SOLFOPL

Initially, we used a straightforward extension for the inference rules described in the preceding section. For instance, an extension (only considering the existential quantification) of the above mentioned inference rule from propositional probabilistic logic to first order case can be:

$$\begin{array}{lll} & \exists x\, \exists y \quad [P(x,y)] & [\underline{\pi}_1, \overline{\pi}_1] \\ S: & \exists x\, \exists y \quad [P(x,y) \vee Q(y)] & [\underline{\pi}_s, \overline{\pi}_s] \\ & \exists y \quad [Q(y)] & [\pi_2?] \end{array}$$

The consistency condition in this case is different from those in AD-PPL because of the introduction of the quantifiers in first order logic. More particularly, they are similar for the upper and lower bounds of individual sentences and predicates but the relation between quantified predicates and quantified sentences is yet to be explored (in future work). For the numerical values of the bounds, we used the exact algorithm proposed in [63] and [29] (presented in Section 3.5.1).

In most of the cases, the straightforward extension technique does not look sufficient to help finding the tightest probability bounds for the additional sentence in the *optimization* form of FOPL (i.e., P^{FOENTAIL}). Having a closer look, it reveals that the issue of the quantifiers is not fully explored in the straightforward extension scheme (first algorithm). Here, we propose another extension of inference rules from the first set to an enhanced one based on the exploration of the quantifiers of the formulas. We present this idea with the following example. Let us consider the previous example (the original inference rule and its naive

extension) once more. Suppose, we have a different sentence this time and the rule looks like:

$$\begin{array}{lll} \exists x\, \exists y & [P(x,y)] & [\underline{\pi}_1, \overline{\pi}_1] \\ S : \forall x\, \exists y & [P(x,y) \vee Q(y)] & [\underline{\pi}_s, \overline{\pi}_s] \\ \exists y & [Q(y)] & [\pi_2 ?] \end{array}$$

Having a closer look will reveal that we have considered only one particular case of quantification (existential) for both the constituent predicates of sentence S while there are others are yet to be considered. This seems to have incomplete information to evaluate the bounds of $Q(y)$. To be more clear on what we are missing, let us check the following instance.

$$\begin{array}{lll} S : \forall x\, \exists y & [P(x,y) \vee Q(y)] & [\underline{\pi}_s, \overline{\pi}_s] \\ \forall x\, \forall y & [P(x,y)] & [\underline{\pi}_{11}, \overline{\pi}_{11}] \\ \forall x\, \exists y & [P(x,y)] & [\underline{\pi}_{12}, \overline{\pi}_{12}] \\ \exists x\, \forall y & [P(x,y)] & [\underline{\pi}_{13}, \overline{\pi}_{13}] \\ \exists x\, \exists y & [P(x,y)] & [\underline{\pi}_{14}, \overline{\pi}_{14}] \\ \forall y & [Q(y)] & [\underline{\pi}_{21}, \overline{\pi}_{21}] \\ \exists y & [Q(y)] & [\pi_{22} ?] \end{array}$$

Now, we have more information about the probability bounds for both the predicates, $P(x,y)$ and $Q(y)$. If a predicate has no given probability bound, the bound for the predicate

is initialized with [0.0, 1.0]. Although this extension scheme helps to find the tightest bound on $Q((y)$ in most of the cases, it looks like an exhaustive enumeration of all possible quantifications on the predicates of the sentence S. More importantly, it may take a long time to compute the bounds when we have too many constraints in a single inference rule. The benefit of this scheme seems outweighed by the possibly high computation time overhead. Again, we have following combinations of the sentence S for each of which we have to consider all the combinations of the quantified predicates as shown in the above example.

$$S^0 : \forall x\ \exists y \quad [P(x,y) \vee Q(y)] \quad [\underline{\pi}_s, \overline{\pi}_s]$$

$$S^1 : \forall x\ \forall y \quad [P(x,y) \vee Q(y)] \quad [\underline{\pi}_s, \overline{\pi}_s]$$

$$S^2 : \exists x\ \forall y \quad [P(x,y) \vee Q(y)] \quad [\underline{\pi}_s, \overline{\pi}_s]$$

$$S^3 : \exists x\ \exists y \quad [P(x,y) \vee Q(y)] \quad [\underline{\pi}_s, \overline{\pi}_s]$$

Clearly, the inference rules can be generalized to the following form:

$$\Delta_1 x\ \Delta_2 y \quad [P(x,y)] \qquad\qquad [\underline{\pi}_1, \overline{\pi}_1]$$

$$S : \Delta_1 x\ \Delta_2 y \quad [P(x,y) \vee Q(y)] \quad [\underline{\pi}_s, \overline{\pi}_s]$$

$$\Delta_1 y \quad [Q(y)] \qquad\qquad [\pi_2?]$$

where Δ_2 and Δ_1 assume values from all combinations of the quantifiers (i.e., $\forall x$ and $\exists y$).

This is a complete set of inference rules for a particular combination of sentences and predicates that can be applied but we apply only a subset of these rules based on some heuristics. It is to be noted that the order of application of the inference rules is very

important to get a improved bound for a particular predicate or a sentence.

Suggestions

Instead of taking all the combinations of the predicates we can take a reduced subset which are effective to produce a good result (i.e, a tight probability bound). Initially, we try to improve the probability bound of each quantified combination of the predicates. Some values are improved, some are not (i.e., remains with [0, 1] probability bounds). Next, from the inference rule, we eliminate some of the quantified combination of the predicates which still have the [0, 1] bounds after the first attempt to improve their bounds. But in this process we do not exclude those quantified combination of the predicates which have the same sequence as is in the original sentence. For example, if the sentence S has the value $\forall\exists$ for the generalized quantifiers Δ_1 and Δ_2, we do not exclude any predicate having the same sequence as S for quantifiers (e.g., $\forall\exists$ [P(x,y)] even with [0, 1] probability bounds will not be excluded). This suggestion is very effective because applying the modified inference rule (of reduced size) we do compute the same probability interval but the computation (i.e., CPU) time is greatly reduced. This is one of the main issues considered on the improved version of the algorithm called AD-SOLFOPL+.

4.1.2 Outline of the AD-SOLFOPL procedure

Suppose, we are given a set of logical sentences, $\mathcal{S} = \{S_1, S_2, \ldots, S_m\}$, associated with their probability intervals $[\underline{\pi}_{S_k}, \overline{\pi}_{S_k}]$ for $k = 1, 2, \ldots, m$, and defined on a set of boolean variables $X = \{x_1, x_2, \ldots, x_n\}$ by a set of predicates $\mathcal{P} = \{P_1, P_2, \ldots, P_r\}$ associated with the probability intervals $[\underline{\pi}_i, \overline{\pi}_i]$ for $i = 1, 2, \ldots, r$. The AD-SOLFOPL procedure attempts to

produce a tight probability interval $[\underline{\pi}_{m+1}, \overline{\pi}_{m+1}]$ for an additional sentence S_{m+1}, in such a way that the overall system $(\mathcal{S} \cup S_{m+1}, [\underline{\pi}, \overline{\pi}])$ where $\underline{\pi}, \overline{\pi} \in [0,1]^{m+1}$, remains consistent. If some of the predicates are given without prior probability, at the initialization, all predicates for which no information is given on their uncertainty are assigned the trivial probability interval $[0, 1]$.

At the beginning, each individual predicate $P_i(x)$, for $i = 1, 2, \ldots, r$ is marked as .false. which means none of the probability intervals of the predicates has been attempted to be tightened yet. Each of the sentences S_k, for $k = 1, 2, \ldots, m$, is also marked .false. which means none of them has been searched for updating the probability intervals. Updated bound, which means being successful at updating one probability bound after solving a probabilistic entailment problem is also set to .false. at the beginning for all the predicates. The AD-SOLFOPL procedure starts with the examination of the additional sentence S_{m+1} and focuses on improving the probability interval of one of its predicates, $P_j(x)$, one at a time. Then, it searches for an occurrence of the predicate $P_j(x)$ (either in negated or positive form) in the set of given sentences. Whenever a sentence, which contains $P_j(x)$, is found it tries to update the bounds of the remaining predicates (i.e., which are still marked as .false.) of the newly identified sentence by following the same procedure. The AD-SOLFOPL procedure solves a probabilistic entailment problem, following the predefined set of inference rules, whenever all the predicates in a sentence are marked as .true. (i.e., their probability intervals are either tightened or attempted to be done so). It is a recursive procedure which continues until all the predicates are either updated or, more precisely, an attempt to improve the bounds has been made.

Let, S be an arbitrary sentence. Each time the algorithm AD-SOLFOPL solves a probabilistic entailment problem $\mathcal{P}_e\big((S_k)_{k\in K}, S\big)$ using the exact method of [63], [29], not only it generates the probability interval of the target sentence S, but also checks whether the problem $\mathcal{P}_e\big((S_k)_{k\in K}, S\big)$ is consistent. If the problem is not consistent, it implies that the original probabilistic entailment problem ($\mathcal{P}$) is not consistent either, and the algorithm AD-SOLFOPL stops at that point.

As the AD-SOLFOPL algorithm is iterating, probability intervals are tightened for a small set of sentences and predicates and the current set of probability intervals assigned to the predicates and sentences. This small set of sentences and predicates are selected according to the set of inference rules described in Section 4.1.1.2. The AD-SOLFOPL procedure defines an order in which the inference rules are applied in order to find the tightest possible probability intervals of an additional sentence with a minimum number of iterations. AD-SOLFOPL starts with the investigation of the additional sentence, i.e., it examines each of the predicates it contains and looks for the logical sentences those contain them. Next, it makes recursive attempts to improve the probability intervals of those predicates.

At first, the algorithm tries to improve the bounds of a predicate by involving inference rules where only one formula and its constituent predicates are involved. In the second stage, it tries to improve the bounds of a predicate by involving two of the sentences where both the sentences contain the predicate for which we try to improve the bound.

Now, we will see an illustration of the AD-SOLFOPL procedure on a small example. Let us consider the following example with 5 formulas defined as follows. We assume that the subset composed of the first four sentences is assumed to be consistent and that we are interested in the tightest possible range of the probability values that can be assigned to

the additional sentence, S_5, such that the overall set of formulas (all five) remains consistent.

$$
\begin{array}{lll}
S_1 \equiv \forall x\, \exists y & [P(x,y) \vee Q(y)] & [0.85, 0.90] \\
S_2 \equiv \exists x & [\neg R(x)] & [0.55, 0.60] \\
S_3 \equiv \exists y & [\neg Q(y)] & [0.55, 0.60] \\
S_4 \equiv \forall x\, \forall y & [\neg P(x,y) \vee R(x) \vee Q(y)] & [0.65, 0.70] \\
S_5 \equiv \exists x\, \exists y & [\neg P(x,y)] & [\underline{\pi}, \overline{\pi}]?.
\end{array}
$$

As we have stated, we are interested in the largest possible range of the probability values of $S_5 \equiv \exists x\, \exists y \quad [\neg P(x,y)]$ such that the overall set of sentences and probability intervals are consistent.

Let us apply the sequential deductive approach of the AD-SOLFOPL algorithm starting with the additional sentence, $S_5 \equiv \exists x\, \exists y \quad [\neg P(x,y)]$. Now we will try to find the sentence that contains the predicate $P(x,y)$ from the additional sentence. The procedure examines all the sentences that contain the predicate $P(x,y)$ and in the successive steps try to improve the probability bound of the remaining predicates of those sentences in order to improve the bound of S_5.

For this particular example, let us consider S_1, which contains $P(x,y)$ in positive form. In addition to $P(x,y)$, S_1 contains predicate $Q(y)$. Now $Q(y)$ becomes the predicate for which we try to tighten the probability interval. Again, we start from the beginning (i.e., from S_1) but this time searching for $Q(y)$. In S_1, both predicates ($P(x,y)$ and $Q(y)$) are already marked .true. and S_1 does not contain any unmarked predicate. We therefore attempt to improve the probability interval of $Q(y)$. The inference rule that comes in to

play to solve the entailment problem looks like:

$$
\begin{array}{lll}
S : \forall x\ \exists y & [P(x,y) \vee Q(y)] & [0.85, 0.90] \\
\exists x\ \exists y & [P(x,y)] & [0.0, 1.0] \\
\forall y & [Q(y)] & [\underline{\pi}, \overline{\pi}]?
\end{array}
$$

Note that we only considered the positive literal for both the predicates and when there is no prior value of probability is given, we assume $[0, 1]$ for predicate $P(x, y)$ which is originated from S_5. In order to get the best possible values for $\underline{\pi}$ and $\overline{\pi}$, we use the exact AD-SOLFOPL procedure developed by [63] based on [29]. Just to recall from Chapter 3, this implementation used column generation technique instead of straightforward LP solution. We get the probability interval of $[0.0, 0.9]$.

We compare it with the incumbent interval, i.e., the initialized value, $[0, 1]$, update and keep the tighter bound, $[0.0, 0.9]$ as current bounds. Next, S_3 is encountered with $Q(y)$ and from there we get a better bound of $[0.4, 0.45]$ for $Q(y)$ as compared to the current bounds.

Note that, a bound is updated only when either the new upper bound is smaller than the previous upper bound or the new lower bound is greater than the previous lower bound. But in no case the lower bound is allowed to be greater than the upper bound. In this case AD-SOLFOPL procedure claims the original problem instance to be inconsistent.

In the continuing way of AD-SOLFOPL procedure, S_4 is encountered and $R(x)$ becomes the new predicate to be evaluated. The inference rules that gets considered next is the following:

$$\forall x\,\forall y \quad [\neg P(x,y) \vee R(x) \vee Q(y)] \quad [0.65, 0.70]$$

$$\exists x\,\exists y \quad [P(x,y)] \quad [0.0, 1.0]$$

$$\exists y \quad [Q(y)] \quad [0.55, 0.60]$$

$$\forall x \quad [\neg R(x)] \quad [\underline{\pi}, \overline{\pi}]?$$

leading to the probability intervals $[0.0, 0.7]$ for $R(x)$ but the bound from S_2 (i.e., $[0.4, 0.45]$) is already tighter and we keep the tighter one as current value. Again, at this point the recursive evaluation for $R(x)$ ends and the procedure backtracks to S_4 for evaluation of $Q(y)$.

$$\forall x\,\forall y \quad [\neg P(x,y) \vee R(x) \vee Q(y)] \quad [0.65, 0.70]$$

$$\forall x\,\forall y \quad [P(x,y)] \quad [0.0, 1.0]$$

$$\exists x \quad [\neg R(x)] \quad [0.55, 0.60]$$

$$\forall y \quad [Q(y)] \quad [\underline{\pi}, \overline{\pi}]?$$

resulting in $[0.0, 0.7]$ for $Q(y)$ but we already have a tighter bound. Now calling for $Q(y)$ ends and the recursive evaluation for $P(x,y)$ resumes at S_1 with the following inference rule:

$$\forall x\, \exists y \quad [P(x,y) \vee Q(y)] \quad [0.85, 0.90]$$

$$\exists y \quad [Q(y)] \quad [0.0, 1.0]$$

$$\forall x\, \forall y \quad [P(x,y)] \quad [\underline{\pi}, \overline{\pi}]?$$

Bounds for $P(x,y)$ is changed to the newly found bounds $[0.0, 0.9]$.

Next, $P(x,y)$ is found in S_4, all the predicates are marked .true., calling the following inference rule:

$$\forall x\, \forall y \quad [\neg P(x,y) \vee R(x) \vee Q(y)] \quad [0.65, 0.70]$$

$$\forall x \quad [R(x)] \quad [0.55, 0.60]$$

$$\forall y \quad [Q(y)] \quad [0.55, 0.60]$$

$$\forall x\, \forall y \quad [P(x,y)] \quad [\underline{\pi}, \overline{\pi}]?$$

where we find the result $[0, 1]$, we keep the previously found tighter one, $[0.0, 0.9]$. Finally, the bounds for the additional sentence $F_5 \equiv S_5 \equiv \exists x\, \exists y \quad [\neg P(x,y)]$ becomes $[0.1, 1.0]$ from $\forall x\, \forall y \quad [\neg P(x,y)]$.

To add to the performance to this algorithm, it is to be mentioned that the optimal solution obtained with the application of the exact algorithm of [63] based on [29] on the whole set of sentences is also $[0.1, 1.0]$, which implies the AD-SOLFOPL algorithm is able to find the tightest probability bound (i.e., the optimal solution) in this case.

4.1.3 Algorithm AD-SOLFOPL

We now describe, in detail, the AD-SOLFOPL algorithm. As we already mentioned, it uses the implementation of the exact algorithm of [63], [29], detects whether a given instance $(S, \underline{\pi}, \overline{\pi})$ is consistent and, in case of consistency, attempts to provide tight bounds on the probability interval of the given logical sentence S_{m+1}.

The AD-SOLFOPL algorithm starts with the investigation of the predicates of the additional sentence in order to tighten the probability intervals, through a call to EVAL_SENTENCE $(S_{m+1}, [\underline{\pi}_{m+1}, \overline{\pi}_{m+1}])$. Once this tightening step is completed, AD-SOLFOPL attempts to tighten the probability interval of S_{m+1} using the improved probability intervals of its predicates. If, during an iteration, the probability interval of a predicate $P_j(x)$ is tightened, UPDATED_BOUNDS is set to .true.. When AD-SOLFOPL investigates the probability interval of a predicate $P_j(x)$, a flag MARK_PRED is set to .true., similarly for the sentences that contains $P_j(x)$ or $\overline{P}_j(x)$. Once a predicate or a sentence is marked, it is no more selected for bound tightening by AD-SOLFOPL during the same iteration. The algorithm is presented below.

Algorithm 4.1: Algorithm AD-SOLFOPL.

Initialization

MARK_PRED$(P_i(x)) \leftarrow$ .false. for all $P_i(x) \in \mathcal{P}$;
MARK_SENTENCE$(S_k) \leftarrow$ .false. for all $S_k \in \mathcal{S}$;
UPDATED_BOUNDS$(P_i(x)) \leftarrow$ .false. for all $P_i(x) \in P$;

Main iteration

EVAL_SENTENCE$(S_{m+1}, [\underline{\pi}_{m+1}, \overline{\pi}_{m+1}])$.

There are two possibilities for tightening the probability interval of a given logical sentence. Either, we first tighten the probability intervals of the predicates defining this sentence using [63], [29]; or we directly attempt to tighten the interval probability of the sentence.

Algorithm 4.2: EVAL_SENTENCE $(S_k, [\underline{\pi}_{S_k}, \overline{\pi}_{S_k}])$

{* Tightening the probability interval of a logical sentence *}
for all $P_j(x) \in S_k$ **do**
  EVAL_PRED$(P_j(x), [\underline{\pi}_j, \overline{\pi}_j]$
end for

The EVAL_PRED procedure not only tighten the probability intervals of some predicates, but along its search for improved bounds, updates the probability intervals of either other predicates and possibly some sentences.

Next, we use the procedure EVAL_PRED_2SENTENCE$(P_j(x))$ for tightening the bounds for the predicate with inference rules involving two sentences. This procedure is able to tighten the bounds further as it has more information with the sentences. In practice it increases the computation time slightly (which is quite understandable) but improves the bounds a lot.

Once the bounds for the predicates are tightened, we try to tighten the bounds of the sentences. If any improvement is achieved for sentences' bound, the procedure for improving the bounds of the predicates is re-computed. As one can observe it, the AD-SOLFOPL algorithm is an anytime algorithm in the sense that, if one stops the algorithm before the stopping conditions are met, we still get a solution. However, it is not necessary the best possible one.

4.2 AD-SOLFOPL+ Procedure

We found reasonable computation time and a satisfactory level of scalability with the application of the first version of our proposed algorithm, called AD-SOLFOPL. But the calculated bounds were not the tightest ones in most of the cases. That is why we identified a few drawbacks of our scheme which was directly followed from the AD-PPL proposed by Jaumard *et. al.* in [33]. We propose a heuristic procedure which follows from our first proposed one with some major modification in its approach. They are discussed below.

As we already mentioned, the first version of the AD-SOLFOPL algorithm follows somewhat similar set of *inference rules* as those are used in AD-PPL proposed by Jaumard *et. al.* in [33]. But we found that by using those inference rules, it is not possible to reach the optimal solution very often. This is because *inference rules* in the first version, we are restricted to only a particular case of quantification, existential one. For instance, for the above case, we were missing except the first quantification for predicate $P(x, y)$. Taking this point into consideration, AD-SOLFOPL heuristic solves for all four variations of $P(x, y)$ listed in the extended inference rules structure. This action gives rise to a better bounds in many cases.

When we extend the set of inference rules, at the beginning somehow it was an exhaustive enumeration of all possible quantification of a predicate. Later on we tried to reduce the number of possible enumeration while keeping the process effective to get the tightest bound. As we already explained in Section 4.1.1.2 some of the predicates with probability bounds $[0, 1]$ are not considered. But in the inference rules we do not exclude those quantified predicates which have the same quantifier sequence as are present in the original sentence.

While sending the set of predicates and sentences following a particular inference rule, we do not send any predicate having trivial bound of $[0, 1]$. This reduces the computational time in small scale for each iteration and in turn lowers the overall computation time in larger scale.

When all the constituent predicates of the additional sentence (objective function) have been updated, we tighten the bound of the additional sentence. We make a small suggestion for the inference rules; to include the sentences which contains the predicates of the objective function. In the first version of the proposed algorithm, the inference rule is used only once to tighten the probability bound of the additional sentence as a whole. In the improved version the additional sentence is sent for tightening as many times as we have occurrence of the predicate(s) of the objective function.

$$F_1 \equiv \forall w\, \forall y \quad [B(w) \vee \neg A(y)] \qquad [0.61, 0.66]$$

$$F_2 \equiv \exists y\, \exists x\, \forall z \quad [A(y) \vee \neg C(x, z)] \qquad [0.71, 0.76]$$

$$F_3 \equiv \exists y \qquad [\neg A(y)] \qquad [0.64, 0.69]$$

$$F_4 \equiv \forall x\, \exists z\, \forall y \quad [C(x, z) \vee A(y)] \qquad [0.71, 0.76]$$

$$F_5 \equiv \exists y\, \forall w\, \forall v\, \exists x \quad [\neg A(y) \vee \neg B(w) \vee \neg D(v, x)] \qquad [0.91, 0.96]$$

$$F_6 \equiv \exists y\, \exists v\, \exists x \quad [\neg A(y) \vee \neg D(v, x)] \qquad [0.85, 0.90]$$

$$F_7 \equiv \exists v\, \exists x \quad [D(v, x)] \qquad [0.55, 0.60]$$

$$F_8 \equiv \exists w\, \exists x\, \exists z \quad [B(w) \vee \neg C(x, z)] \qquad [\underline{\pi}, \overline{\pi}]$$

For example, in the instance with eight sentences, the first version, AD-SOLFOPL solves for

the bound of F_8 **only once** after the bounds of its constituent predicates $B(w)$ and $C(x,z)$ are updated. But in the improved version, AD-SOLFOPL+, F_8 is solved twice for $B(w)$.

$$
\begin{array}{lll}
F_1 \equiv \forall w\, \forall y \quad [B(w) \vee \neg A(y)] & & [0.61, 0.66] \\
\forall w \quad [Q(w)] & & [\underline{\pi}_{11}, \overline{\pi}_{11}] \\
\exists w \quad [Q(w)] & & [\underline{\pi}_{12}, \overline{\pi}_{12}] \\
\forall x\, \forall z \quad [C(x,z)] & & [\underline{\pi}_{21}, \overline{\pi}_{21}] \\
\exists x\, \exists z \quad [C(x,z)] & & [\underline{\pi}_{24}, \overline{\pi}_{24}] \\
F_8 \equiv \exists w\, \exists x\, \exists z \quad [B(w) \vee \neg C(x,z)] & & [\underline{\pi}, \overline{\pi}]
\end{array}
$$

$$
\begin{array}{lll}
F_5 \equiv \exists y\, \forall w\, \forall v\, \exists x \quad [\neg A(y) \vee \neg B(w) \vee \neg D(v,x)] & & [0.91, 0.96] \\
\forall w \quad [Q(w)] & & [\underline{\pi}_{11}, \overline{\pi}_{11}] \\
\exists w \quad [Q(w)] & & [\underline{\pi}_{12}, \overline{\pi}_{12}] \\
\forall x\, \forall z \quad [C(x,z)] & & [\underline{\pi}_{21}, \overline{\pi}_{21}] \\
\exists x\, \exists z \quad [C(x,z)] & & [\underline{\pi}_{24}, \overline{\pi}_{24}] \\
F_8 \equiv \exists w\, \exists x\, \exists z \quad [B(w) \vee \neg C(x,z)] & & [\underline{\pi}, \overline{\pi}]
\end{array}
$$

Then it is solved twice more with F_2 and F_4 as well in the same manner. Now we take the best result (i.e., the smallest upper bound and largest lower bound) as for the final interval.

Therefore, we find that the proposed heuristic is able to obtain tight probability bounds (i.e., optimal solution) in most of the cases with small computation time. When this heuristic is applied on large instances, the benefits of the heuristic becomes more obvious. Again,

the AD-SOLFOPL+ algorithm is an anytime one.

Algorithm 4.3: EVAL_PRED $(P_j(x), [\underline{\pi}_j, \overline{\pi}_j])$

{* Tightening the probability interval of a predicate by taking one logical sentence into account *}

{* Initialization *}
MARK_PRED $(P_j(x)) \leftarrow$ .true.

{* Main Iteration *}
for each $S_k \in \mathcal{S}$ such that $P_j(y) \in \mathcal{P}_k$ and MARK_SENTENCE(S_k) = .false. **do**
  if for all $P_i(x) \in \mathcal{P}_k \setminus \{P_j(x)\}$, MARK_PRED$(P_i(x))$ =.true. **then**
    MARK_SENTENCE$(S_k) \leftarrow$ .true.;
    for all $P_i(x) \in P_k \setminus \{P_i(x)\}$ **do**
      if UPDATED_BOUNDS$(P_i(x))$ = .false. **then**
        EVAL_PRED$(P_i(x), [\underline{\pi}_i, \overline{\pi}_i])$;
      end if
    end for
    Consider all the probabilistic logic problems $\mathcal{P}_e$ defined by $(S_k, [\underline{\pi}_{S_k}, \overline{\pi}_{S_k}])$ and $(P_i(x), [\underline{\pi}_i, \overline{\pi}_i])$ for all combinations of $P_i(x) \in P_k \setminus \{P_j(x)\}$ and quantifiers $(\exists and \forall)$;
    if $\mathcal{P}_e$ is inconsistent **then**
      STOP: the initial system is inconsistent
    else
      Solve $\mathcal{P}_e$ probabilistic logic problem using the appropriate inference rules and LP solution in order to deduce a new interval probability $[\underline{\pi}_j^{\text{NEW}}, \overline{\pi}_j^{\text{NEW}}]$ for $P_j(x)$;
      if $\underline{\pi}_j > \overline{\pi}_j^{\text{NEW}}$ or $\overline{\pi}_j < \underline{\pi}_j^{\text{NEW}}$ **then**
        an inconsistency has been detected, STOP: the initial system is inconsistent ;
        IF$\underline{\pi} j < \underline{\pi}_j^{\text{NEW}}$
        set $\underline{\pi}_j \leftarrow \underline{\pi}_j^{\text{NEW}}$ and UPDATED_BOUNDS$(P_j(x)) \leftarrow$ .true.;
      else
        if $\overline{\pi}_j > \overline{\pi}_j^{\text{NEW}}$ **then**
          set $\overline{\pi}_j \leftarrow \overline{\pi}_j^{\text{NEW}}$ and UPDATE_BOUNDS$(P_j(x)) \leftarrow$ .true.;
        end if
      end if
      MARK_SENTENCE$(S_k) \leftarrow$ .false.;
    end if
  end if
end for

{* Mark label of $P_j(x)$ *}
MARK_PRED$(P_j(x)) \leftarrow$ .false.;

{* Further bound tightening using a special set of *inference rules* that include 2 sentences *}
EVAL_PRED_2SENTENCE$(P_j(x))$

Algorithm 4.4: EVAL_PRED_2SENTENCE($P_j(x)$)

{* Evaluating the probability interval of a predicate by taking two logical sentences into account *}

for all S_l such that $P_i(y) \in P_l$ and $|S_l| = 2$ **do**

  for all $S_k \neq S_l$ such that $P_i(y) \in P_k$ and $|S_k| = 2$ **do**

    Consider the probabilistic logic problem $(\mathcal{P}_e(S_k, S_i, P_j(x)))$ defined by $(S_k, [\underline{\pi}_{S_k}, \overline{\pi}_{S_k}])$, $(S_l, [\underline{\pi}_{S_l}, \overline{\pi}_{S_l}])$ and $(P_i(x), [\underline{\pi}_i, \overline{\pi}_i])$ for all $P_i(x) \in P_k \setminus \{P_j(x)\}$;

    if $\mathcal{P}_e$ is inconsistent **then**

      STOP: the initial system is inconsistent

    end if

    Solve probabilistic entailment problem using the special set of *inference rules* to determine the new interval probability $[\underline{\pi}_j^{\text{NEW}}, \overline{\pi}_j^{\text{NEW}}]$ of $P_j(x)$;

    if $\underline{\pi}_j > \overline{\pi}_j^{\text{NEW}}$ or $\overline{\pi}_j < \underline{\pi}_j^{\text{NEW}}$ **then**

      an inconsistency has been detected, STOP: the initial system is inconsistent ;

    end if

    if $\underline{\pi}_(i) < \underline{\pi}_i^{\text{NEW}}$ **then**

      update Bounds $(P_j(x)) \leftarrow$ true ;

    end if

    if $\overline{\pi}_j > \overline{\pi}_j^{\text{NEW}}$ **then**

      update Bounds $(P_j(x)) \leftarrow$ true ;

    end if

  end for

end for

Chapter 5

Experimental Results

5.1 Setup and Programming Environment

We have implemented two proposed algorithms, AD-SOLFOPL and AD-SOLFOPL+ under Linux environment in C++. The supported compilers that we used are gcc 3.4.4 and higher versions. The implementations amount to around 3000 lines of code, compiled and run under Linux Red Hat 3.4.4-2. For the optimization part, we have used ILOG CPLEX 10.1.1, in order to solve the linear programs (i.e., using the exact algorithm of [63], [29] which uses column generation technique). We run our test instances in computers with AMD dual processors, CPU speed 2392.132 MHz, RAM up to 15.6 GBs.

5.2 Instances for AD-SOLFOPL

We build a set of first order instances in order to evaluate the performance of the AD-SOLFOPL and AD-SOLFOPL+ algorithms. In the next paragraph, we explain some reasons

why we did not take the advantage of the available libraries of test problems for first order logic.

5.2.1 Instances from the tptp library

There is a collection of instances for theorem provers called The Thousands of Problems for Theorem Provers or TPTP in short. This library provides a variety of problems for different types of theorem provers. Initially we tried to work on problem instances from TPTP library. However we found that usually those problems are in a mixed format, i.e., some of them are in CNF while the others are in FOF format. Moreover, none of the problems has formulas having probabilities values associated with them. Most problems have a small number of formulas but each formula is very long. On the contrary, our focus is on problem instances where we have a large number of formulas but each individual formula is of moderate (small) size. That is why we use the strategy proposed by Jaumard *et al.* [30], instead of going for a multi-step preprocessing for using the problem instances from TPTP library. One, further interested about the collections and instances, can check http://www.cs.miami.edu/ tptp/ for more detail about TPTP library.

5.2.2 Generated instances

In order to run our proposed algorithms, we use randomly generated test instances similar to those proposed in Jaumard *et al.* [30]. For the generation of the instances, two different issues have to be taken care of. First, we need a set of consistent first order formulas which are generated based on a given set of input parameters. Second, for each of these generated formulas, we need a range of probability values that each formula is true, so that the overall

set of sentences is true (consistency).

More particularly, in our generator, the logical formulas correspond to clauses (disjunction of predicates) with at most 3 predicates. Formulas with 1, 2, 3 predicates are distributed as uniformly as possible. Again, the positive or negative predicates are distributed almost evenly, i.e., number of positive occurrences is almost equal to the number of negative occurrences. We keep at most two variables in each predicate in order to build a decidable instance [17]. Next comes the issue of the quantifiers, which are chosen and associated to variables of each formula in a random nature. We keep the number of variables equal to about 60 % of the number of formulas in a particular instance. The maximum size (number of predicates) of the additional sentence, defining the objective function, is choosen as two.

For the second issue of the instance generation, we associate consistent probability values to those first-order formulas we already have generated. For this, we try to generate a world (see Section 2.1.1 for the definition) for the complete set of formulas by randomly associating 0 or 1 values to each formula. Then we check whether this 0-1 truth assignment on the formulas leads to a possible world with the help of a theorem prover (theo 2006, see [48] for more information). If the world is found as a possible one, we keep it, otherwise, we reject this assignment and try to generate a new possible world. By following this strategy, we generate a given number of possible worlds for m first-order formulas. In the next step, a uniform probability distribution is assigned over the set of possible worlds, assigning probability distribution 1/p to each possible world. Then, by summing up all the probability distributions (i.e., 1/p) for which there is a 1 value for a particular formula, a fixed probability value is calculated.

Now, by subtracting a small value (e.g., 0.02) and adding up a similar one (e.g., 0.08)

to the fixed probability value, we finally generate the lower and upper probability bounds respectively. Consistency of these formulas are then checked with the associated interval probability values by using CPLEX tools and instances which are found inconsistent are discarded. An illustration of generating consistent probability values for a test instance with 7 first-order formulas (for 12 randomly generated possible worlds) is shown in the Appendix A.1.1. A set of instances that we have used for our proposed algorithm can be found in the Appendix too.

5.2.2.1 Generated input format

When we generate an input instance, we need to set several parameters. The structure and meaning of the parameters of a given problem instance can be seen below in Figure 2.

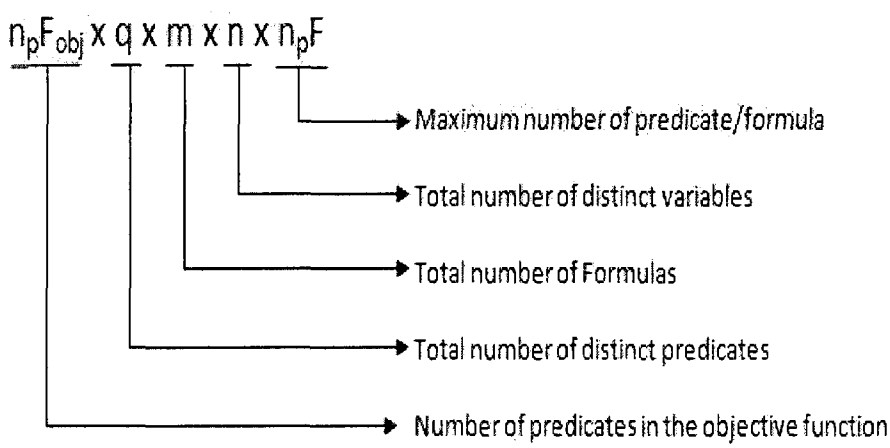

Figure 2: Structure and meaning of the parameters of a problem instance.

Here, n_pF_{obj} is the number of predicates in the additional sentence (objective function). In our generated problem instances, this parameter varies from 1 to 3. Parameters q and m are total number of distinct predicates used in a problem instance and total number of

formulas whereas n is the total number of variables. Usually, n is chosen as 60 % of the total number of formulas. Finally, n_pF expresses the maximum number of predicates used in any formula.

5.3 Performance of AD-SOLFOPL and AD-SOLFOPL+

Here, we discuss the computational results and the performance of our two proposed algorithms. Table 3 and Table 4 show a list of problem instances and its parameters as we described in Figure 2, upper and lower bounds for probability values and computation time to reach those bounds. In Table 3, we show the result of the first algorithm, AD-SOLFOPL, which can be better described as a straightforward extension from the AD-PPL proposed by Jaumard *et al.* [30]. In this algorithm, we followed a set of inference rules where only existential quantification is considered with those predicates which were not given explicitly.

Table 3: Results of AD-SOLFOPL.

Instances					AD-SOLFOPL		
					Probability		CPU time
$n_{pF_{obj}}$	q	m	n	n_{pF}	$\underline{\pi}$	$\overline{\pi}$	(sec.)
1	3	4	3	3	0.100	1.000	0.160
2	4	7	4	2	0.037	1.000	0.370
3	4	8	4	3	0.000	1.000	0.760
2	5	10	6	3	0.000	1.000	0.840
3	8	15	7	3	0.000	1.000	1.250
2	10	20	12	3	0.242	1.000	1.480
2	11	25	14	3	0.000	1.000	1.620
2	11	30	18	3	0.000	1.000	1.970
2	11	35	21	3	0.390	1.000	2.350
2	20	40	24	3	0.000	1.000	1.000
2	22	45	26	3	0.448	1.000	1.370
2	25	50	30	3	0.000	1.000	1.720
2	30	55	32	3	0.118	1.000	1.420
2	30	60	36	3	0.000	1.000	1.630
2	50	100	40	3	0.000	1.000	2.670

After the first algorithm, the set of inference rules were extended for all possible quantifications. With this extension we are able to find the tightest probability bounds for most

Table 4: Results of AD-SOLFOPL+.

Instances					AD-SOLFOPL+ Probability		AD-SOLFOPL+ CPU time
$n_{pF_{obj}}$	q	m	n	n_{pF}	$\underline{\pi}$	$\overline{\pi}$	(sec.)
1	3	4	3	3	0.100	1.000	0.540
2	4	7	4	2	0.037	1.000	1.190
3	4	8	4	3	0.274	1.000	2.250
2	5	10	6	3	0.551	1.000	2.190
3	8	15	7	3	0.249	0.978	2.810
2	10	20	12	3	0.450	1.000	2.360
2	11	25	14	3	0.188	1.000	4.900
2	11	30	18	3	0.000	1.000	3.720
2	11	35	21	3	0.390	1.000	3.400
2	20	40	24	3	0.960	1.000	2.960
2	22	45	26	3	0.448	1.000	4.330
2	25	50	30	3	0.138	1.000	3.970
2	30	55	32	3	0.446	1.000	4.710
2	30	60	36	3	0.150	1.000	4.570
2	50	100	40	3	0.238	1.000	5.810

of the instances. But the computation time becomes too high, specially, for larger instances. Therefore, we developed a heuristic technique to find out a reduced set of inference rules. With the help of this heuristic technique we are able to find the same probability intervals but the (CPU) computation time is reduced by a an approximate factor of 5.

We present the results using the second algorithm AD-SOLFOPL+ in Table 4. From these results (i.e., of Table 4), we find that the (CPU) computation time has increased slightly compared to those of Table 2 but the probability bounds are tightened a lot. The results of AD-SOLFOPL shows a wider probability bound or close to $[0.0, 1.0]$ in most of the cases with smaller computation time. But for AD-SOLFOPL+, the probability bounds are improved and the computation time is also reasonable.

5.4 Comparison of AD-SOLFOPL with an Exact Method

In this section we compare the result of our proposed algorithm with the exact algorithm called SOLFOPL proposed in [63], [29]. We evaluate the performance of our heuristic algorithm w.r.t. this exact one (see Chapter 3 for more information on this exact algorithm.)

At first, we present a comparison between the AD-SOLFOPL and SOLFOPL for a set of consistent examples in Table 4. The parameter values are arranged in the same way same as in the previous tables.

Table 5: Results of an exact method using column generation technique.

					SOLFOPL		
	Instances				Probability		CPU time
$n_{pF_{obj}}$	q	m	n	n_{pF}	$\underline{\pi}$	$\overline{\pi}$	(sec.)
1	3	4	3	3	0.100	1.000	0.09
2	4	7	4	2	0.037	1.000	0.19
3	4	8	4	3	0.462	1.000	0.22
2	5	10	6	3	0.551	1.000	0.33
3	8	15	7	3	0.249	0.978	2.06
2	10	20	12	3	0.450	1.000	1.65
2	11	25	14	3	0.188	1.000	18.7
2	11	30	18	3	0.000	1.000	18.37
2	11	35	21	3	0.390	1.000	11.10
2	20	40	24	3	—	—	$\geq$ 1 week
2	22	45	26	3	—	—	$\geq$ 1 week
2	25	50	30	3	—	—	$\geq$ 1 week
2	30	55	32	3	—	—	$\geq$ 1 week
2	30	60	36	3	—	—	$\geq$ 1 week
2	50	100	40	3	—	—	$\geq$ 1 week

From Table 4 and 7 we draw a few conclusion as below.

Firstly, the scalability issue: The exact algorithm, SOLFOPL, can solve problem instances of length up to 35 formulas whereas the AD-SOLFOPL+ solves up to 100 formulas (and possibly more) in a reasonable amount of time. For the larger instances, SOLFOPL was forced to stop after several days of ongoing computation. This improvement of scalability of AD-SOLFOPL+ is a good achievement over the exact algorithm [63] and [29].

Secondly, AD-SOLFOPL+ is able to find the tightest probability bounds for most of the cases when the instances are solved by the exact algorithm. This shows the merit of a good heuristic method.

Finally, we find a huge improvement in computation time when we use the AD-SOLFOPL+ algorithm. For the small instances, (e.g., 5 formulas), the computation time is slightly higher than the exact algorithm but the computation time benefit of the heuristic becomes obvious when we compute bounds for larger instances (e.g. 25 formulas). This is shown clearly in the graph of Figure 3 where x axis represents the increasing size (w.r.t. number of formulas) of the input problem instances and y axis represents the (CPU) computation time (in seconds) for the probability of an additional sentence. This graph clearly reveals, at the beginning, computation time for the exact algorithm is lower than that of the heuristic approach. As the problem size increases, the computation time for the exact algorithm increases abruptly whereas it increases almost linearly for the heuristic algorithm AD-SOLFOPL+.

Based on the random instance generator, even with the same input parameters that we currently use, the generated instances are different. Table 6 illustrate this fact with the results using the exact algorithm. That is why we consider this as a potential future work to find the complex relationship among the parameters. Other parameters which are not mentioned here (e.g., choice of quantifiers, size of the predicate, size of the formula and their proportion etc.) should be considered as well. Next, we also show a few cases (for randomly generated instances) where we compare how both the algorithms identified the inconsistency in the original example.

In the graph of Figure 3, we compare the computation time (i.e., CPU Time) between SOLFOPL and AD-SOLFOPL for input instances of different sizes. In this graph, the formula

Table 6: Results multiple instances for same generation parameters.

Instances					SOLFOPL		
					Probability		CPU time
$n_{pF_{obj}}$	q	m	n	n_{pF}	$\underline{\pi}$	$\overline{\pi}$	(sec.)
2	4	7	4	2	0.186	1.000	0.22
2	4	7	4	2	0.394	1.000	0.22
2	4	7	4	2	0.320	1.000	0.18
2	5	10	6	3	0.323	1.000	0.47
2	5	10	6	3	0.303	1.000	0.51
2	5	10	6	3	0.227	1.000	0.51
3	8	15	7	3	0.355	1.000	3.29
3	8	15	7	3	0.449	1.000	3.20
3	8	15	7	3	0.456	1.000	1.61

Table 7: Comparison between SOLFOPL and AD-SOLFOPL in case of inconsistent instances.

Instances					% INCONSISTENT		
					CPU time (sec.)		
$n_{pF_{obj}}$	q	m	n	n_{pF}	solfopl	AD-SOLFOPL	(problems found by AD-SOLFOPL)
2	11	35	21	3	> 2 sec	< 1 sec	100
2	20	40	24	3	—	< 1 sec	100
2	25	50	30	3	—	< 1 sec	100

size (i.e., number of formula in the instance) is plotted along the x axis and the computation time (CPU time in sec) is shown along the y axis. Here it is obvious that there is a great improvement in the computation time for larger instances, specially beginning from mid sized instances. There is a spike in the graph due to problem instance of 25 formulas, which is due to the specific nature (e.g., number of predicates in a formula, number of positive and negative predicates in the large formulas etc.) of that generated problem. It is not completely obvious to generate random instances with the same solution complexity when increasing the number of formulas without dealing with the other parameters as well. We wish to reveal more on the apparently random nature of the generated instances in future works.

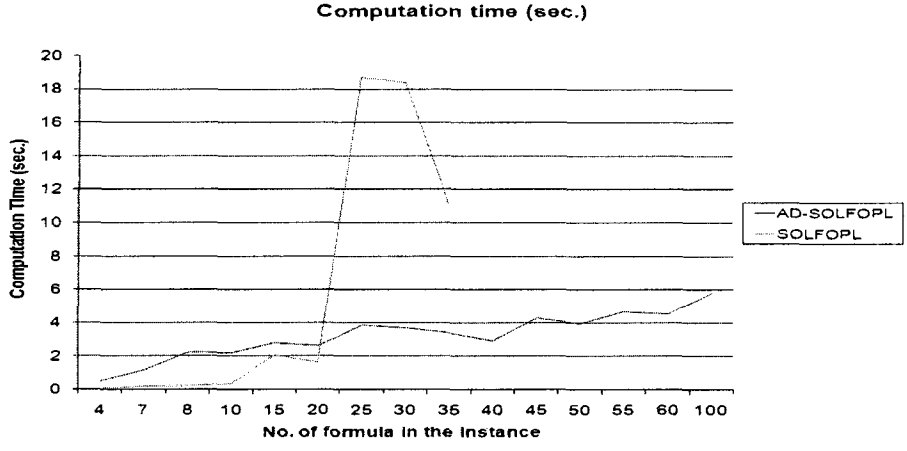

Figure 3: Comparison of computation time between the proposed algorithm and an exact one.

Chapter 6

Conclusion and Future Work

In this thesis, our main focus is on those models which deal with reasoning under uncertainty, a heuristic implementation technique where uncertainty is described in terms of probability values. In 1986, Nilsson published an article [50] on the *probabilistic logic* and mentioned its two forms: namely, the *decision* and the *optimization* form. The *decision* form (or *probabilistic satisfiability*) checks the consistency of a set of logical sentences together with their probabilities that these sentences are true. On the other hand, in the *optimization* form (or *probabilistic entailment problem*), it tries to find the best possible bounds on the probability values π_{m+1} associated with an additional sentence S_{m+1}, such that the overall system remains consistent. Except the probability issue, Nilsson's P^{PSAT} and P^{PENTAIL} problem can be reduced to the *satisfiability* and the *logical consequence* problem of AI respectively.

The probabilistic logic problem (combination of both P^{PSAT} and P^{PENTAIL}) can be addressed in two general ways: using exact methods or anytime deduction methods. Recall that an exact solution scheme is one which guarantees to find the solution, if there exists

one. Usually, it attains a greater accuracy at the cost of possibly a higher computation (CPU) time.

A straightforward exact method to solve the probabilistic logic problem could be by adopting a linear program solution or LP solution using the simplex or the revised simplex algorithm (see e.g., Chvatal [11], p.97 for more information). Unfortunately, this solution scheme becomes unmanageable even for a moderate sized problem. However, there are some powerful tools in operations research which are efficient enough to handle this kind of unmanageable situation. The Column generation techniques (see [11], p. 198) is one such example which is proved to be efficient in solving probabilistic logic problem in propositional sentences or PPL for short. This has been tested successfully for moderate sized (i.e., w.r.t. number of formulas) first order instances as well (see [63] for more details). The powerful expressive nature of First-order logic has made it closer to a human like representation as compared to, e.g., propositional calculus. That is one of the major issues that made us interested about investigating the anytime deduction technique for first-order probabilistic logic or FOPL for short. Another issue for interest has arisen when we closely studied the high computation time setback for solving large instances while using the exact method (implemented by R. Sultana in [63]).

In this thesis, first, we try to extend the anytime deduction solution for propositional probabilistic logic, or AD-SOLPPL proposed by Jaumard *et al.* in [32], to first order logic, AD-SOLFOPL. But the extension is not something really straightforward. There are lot of differences between the propositional case and the first order one, e.g., the consideration of quantifiers, the issue of decidability etc.

The general procedure is as such: we are given a set of decidable logical sentences in

first order form, $\mathcal{S} = \{S_1, S_2, \ldots, S_m\}$ associated with their probability intervals $[\underline{\pi}_{S_k}, \overline{\pi}_{S_k}]$ for $k = 1, 2, \ldots, m$, and defined on a set of boolean variables $X = \{x_1, x_2, \ldots, x_n\}$ by a set of predicates $\mathcal{P} = \{p_1, p_2, \ldots, p_r\}$ associated with probability intervals $[\underline{\pi}_i, \overline{\pi}_i]$ for $i = 1, 2, \ldots, r$. The AD-SOLFOPL procedure attempts to output a tight probability interval $[\underline{\pi}_{m+1}, \overline{\pi}_{m+1}]$ for an additional sentence S_{m+1}, in such a way that the overall system $(\mathcal{S} \cup S_{m+1}, [\underline{\pi}, \overline{\pi}])$ where $[\underline{\pi}, \overline{\pi}] \in [0, 1]^{m+1}$, remains consistent.

The AD-SOLFOPL procedure starts with the examination of the additional sentence S_{m+1} and takes one predicate at a time from it. Then it searches for an occurrence of the predicate $p_j(x)$ (either in negated or positive form) in the set of given sentences. Whenever a sentence is found, which contains $p_j(x)$, it tries to update the bounds of the remaining predicates of the newly identified sentence by following the same procedure. It's a recursive procedure which continues until all the predicates or sentences are either updated with their bounds or attempted to be updated.

As the second contribution, we have suggested a generalized set of inference rules, an extension to the set used in Jaumard *et al* . [32]. As compared to Jaumard *et al.* [32], our proposed set is more general as it considers the quantifications on the variables for which the predicates are defined.

As the third contribution, we have developed an improved version of the first algorithm AD-SOLFOPL and named that version as AD-SOLFOPL+. This improved version has better probability values and reaches a tighter bound as compared to the basic algorithm. Sometimes we have to trade off the tighter bound with the computation time.

Finally, we have implemented two versions of the heuristic algorithm successfully. The experiments and results of those experiments based on our implementation are described in

Chapter 5. A comparison is made between the performance of our algorithm, and that of SOLFOPL [63]. The SOLFOPL is an exact algorithm based on column generation method.

Drawbacks and future directions:

Firstly, when the generated instances are large enough, it tends to produce [0, 1] bounds more often for the additional sentence while solving the optimization problem for first order case. We tried to study the effect of the probability intervals for individual formulas on the bound of an additional sentence for the optimization problem. This led to an interesting observation. If the probability bounds for individual sentences are tighter, it tends to produce tighter bounds for an additional sentence. But to achieve that is not very easy, specially for larger instances, because the probability value variation is not smooth as it very quickly moves from a consistent system with [0,1] interval to an inconsistent system. Future works may be carried out in this direction in order to develop a more robust generator for first order case.

Secondly, the computation time becomes too large when we increase the size of the objective function (additional sentence) for large instances. So, still we keep at most two predicates for the objective function. Even for other formulas, the maximum size is three predicates per formula.

Thirdly, our proposed algorithm is explored only for unconditional probability in first order probabilistic logic. AD-SOLFOPL algorithm can be extended for conditional probability. Moreover, a tool can be developed in order to evaluate a heuristic algorithm considering how far we are from the tightest probability interval. This we leave as a possible future work.

Fourthly, an enhanced set of inference rules can be proposed with more complexity. The

more complex the set of inference rules become, i.e., more information is included, better the quality of the bound becomes. Although it is not practical, if we add more information in the inference rule and increase the size of it so that it includes all the formulas of the given problem instance, it behaves similar to the exact algorithm.

Appendix A

Appendix A

A.1 Input Generation Process

A.1.1 An illustration of the probability generation

An illustration of generating consistent probability values for a test instance with 7 first-order formulas is shown below.

A.1.2 An input file

We show the screen shot of an input file which shows the format of the formulas and the probability values associated with each one.

Generation of consistent probability values.

F_i	W_1	W_2	W_3	W_4	W_5	W_6	W_7	W_8	W_9	W_{10}	W_{11}	W_{12}	Probability Value
F_1	1	1	0	0	0	1	1	0	0	0	0	1	= 5 x 1/12 = 0.42
F_2	1	1	0	0	0	1	1	0	0	0	1	1	= 6 x 1/12 = 0.50
F_3	1	1	0	1	1	1	0	0	0	1	0	1	= 7 x 1/12 = 0.58
F_4	1	1	0	1	1	1	1	0	0	0	1	0	= 7 x 1/12 = 0.58
F_5	1	1	1	0	1	1	1	0	0	1	1	1	= 9 x 1/12 = 0.75
F_6	1	1	1	1	0	1	0	0	0	1	1	1	= 8 x 1/12 = 0.67
F_7	1	1	1	1	1	1	1	1	0	1	1	1	= 11 x 1/12 = 0.92

```
[POSITIVE FORMULAS]
fof(f_1,axiom,
(  ! [X5] : ? [X3] : big_p2(X5) |  ~ big_p3(X3,X5) )).
0.561
0.611
fof(f_2,axiom,
(  ? [X2] : ! [X4] : ! [X6] : ? [X1] : big_p1(X2) | big_p5(X2,X4) |  ~ big_p4(X6,X1) )).
0.851
0.901
fof(f_3,axiom,
(  ! [X2] : ? [X5] : ! [X6] : ! [X1] : big_p1(X2) |  ~ big_p2(X5) |  ~ big_p4(X6,X1) )).
0.657
0.707
fof(f_4,axiom,
(  ? [X2] : ? [X4] :  ~ big_p5(X2,X4) )).
0.561
0.611
fof(f_5,axiom,
(  ? [X2] : ? [X4] : ? [X5] : big_p1(X2) |  ~ big_p5(X2,X4) |  ~ big_p2(X5) )).
0.722
0.772
fof(f_6,axiom,
(  ? [X5] : ? [X3] : ! [X6] : ? [X1] : big_p2(X5) |  ~ big_p3(X3,X5) |  ~ big_p4(X6,X1) )).
0.867
0.917
fof(f_7,axiom,
(  ? [X3] : ? [X5] :  big_p3(X3,X5) )).
0.754
0.804
fof(f_8,axiom,
(  ! [X2] : ! [X5] : big_p1(X2) |  ~ big_p2(X5) )).
0.399
0.449
fof(f_9,axiom,
(  ? [X2] : ? [X3] : ? [X5] : big_p1(X2) |  ~ big_p3(X3,X5) )).
0.657
0.707
fof(f_10,axiom,
(  ! [X2] : ! [X3] : ! [X5] : ! [X6] : ? [X1] : big_p1(X2) |  ~ big_p3(X3,X5) |  ~ big_p4(X6,X1
0.657
0.707
fof(f_11,axiom,
(  ? [X2] : ? [X5] :  big_p1(X2) | big_p2(X5) )).
1.0
1.0
[NEGATIVE FORMULAS]
```

Screen shot of a input file 2p4x5x3p

A.2 List of Generated Inputs

A.2.1 Instance 1 (4 formulas):

$$F_1 \equiv \forall x\, \exists y \quad [P(x,y) \vee Q(y)] \qquad [0.85, 0.9]$$

$$F_2 \equiv \exists x \quad [\neg R(x)] \qquad [0.55, 0.6]$$

$$F_3 \equiv \exists y \quad [\neg Q(y)] \qquad [0.55, 0.6]$$

$$F_4 \equiv \forall x\, \forall y \quad [\neg P(x,y) \vee R(x) \vee Q(y)] [0.65, 0.7]$$

$$F_5 \equiv \exists x\, \exists y \quad [\neg P(x,y)] \qquad [\underline{\pi}, \overline{\pi}]$$

A.2.2 Instance 2 (7 formulas):

$$F_1 \equiv \forall w\, \forall y \quad [B(w) \vee \neg A(y)] \qquad [0.61, 0.66]$$

$$F_2 \equiv \exists y\, \exists x\, \forall z \quad [A(y) \vee \neg C(x, z)] \qquad [0.71, 0.76]$$

$$F_3 \equiv \exists y \qquad [\neg A(y)] \qquad [0.64, 0.69]$$

$$F_4 \equiv \forall x\, \exists z\, \forall y \quad [C(x, z) \vee A(y)] \qquad [0.71, 0.76]$$

$$F_5 \equiv \exists y\, \forall w\, \forall v\, \exists x \quad [\neg A(y) \vee \neg B(w) \vee \neg D(v, x)] \qquad [0.91, 0.96]$$

$$F_6 \equiv \exists y\, \exists v\, \exists x \quad [\neg A(y) \vee \neg D(v, x)] \qquad [0.85, 0.90]$$

$$F_7 \equiv \exists v\, \exists x \quad [D(v, x)] \qquad [0.55, 0.60]$$

$$F_8 \equiv \exists w\, \exists x\, \exists z \quad [B(w) \vee \neg C(x, z)] \qquad [\underline{\pi}, \overline{\pi}]$$

A.2.3 Instance 3 (8 formulas):

$$F_1 \equiv \exists w \quad [B(w)] \qquad [0.72, 0.77]$$

$$F_2 \equiv \forall y\, \forall w\, \exists x\, \exists z \quad [A(y) \vee B(w) \vee \neg C(x,z)] \quad [0.72, 0.77]$$

$$F_2 \equiv \forall y\, \forall w\, \exists x\, \forall z \quad [A(y) \vee \neg B(w) \vee \neg C(x,z)] \quad [0.72, 0.77]$$

$$F_4 \equiv \exists y\, \forall w \quad [A(y) \vee B(w)] \qquad [0.63, 0.68]$$

$$F_5 \equiv \forall y\, \exists w\, \exists v\, \exists x \quad [A(y) \vee \neg B(w) \vee \neg D(v,x)] \quad [0.81, 0.86]$$

$$F_6 \equiv \exists y\, \exists v\, \forall x \quad [\neg A(y) \vee \neg D(v,x)] \qquad [0.72, 0.77]$$

$$F_7 \equiv \exists y\, \forall x\, \forall z \quad [A(y) \vee \neg C(x,z)] \qquad [0.68, 0.73]$$

$$F_8 \equiv \exists v\, \exists x\, \exists w \quad [D(v,x) \vee \neg B(w)] \qquad [0.81, 0.86]$$

$$F_9 \equiv \exists y\, \exists x\, \exists z\, \exists v \quad [A(y) \vee C(x,z) \vee D(v,x)] \qquad [\underline{\pi}, \overline{\pi}]$$

A.2.4 Instance 4 (10 formulas):

$$F_1 \equiv \forall x\, \exists v \quad [B(x) \vee \neg C(v,x)] \qquad [0.56, 0.61]$$

$$F_2 \equiv \exists u\, \forall w\, \forall y\, \exists t \quad [A(u) \vee E(u,w) \vee \neg D(y,t)] \qquad [0.85, 0.90]$$

$$F_3 \equiv \forall u\, \exists x\, \forall y\, \forall t \quad [A(u) \vee \neg B(x) \vee \neg D(y,t)] \qquad [0.66, 0.71]$$

$$F_4 \equiv \exists u\, \exists w \quad [\neg E(u,w)] \qquad [0.56, 0.61]$$

$$F_5 \equiv \exists u\, \exists w\, \exists x \quad [A(u) \vee \neg E(u,w) \vee \neg B(x)] \qquad [0.72, 0.77]$$

$$F_6 \equiv \exists x\, \exists v\, \forall y\, \exists t \quad [B(x) \vee \neg C(v,x) \vee \neg D(y,t)] \qquad [0.87, 0.92]$$

$$F_7 \equiv \exists v\, \exists x \quad [C(x,z)] \qquad [0.75, 0.81]$$

$$F_8 \equiv \forall u\, \forall x \quad [A(u) \vee \neg B(x)] \qquad [0.40, 0.45]$$

$$F_9 \equiv \exists u\, \exists v\, \exists x \quad [A(u) \vee \neg C(v,x)] \qquad [0.66, 0.71]$$

$$F_{10} \equiv \forall u\, \forall v\, \forall x\, \forall y\, \exists t \quad [A(u) \vee \neg C(v,x) \vee \neg D(y,t)] \qquad [0.66, 0.71]$$

$$F_{11} \equiv \exists u\, \exists x \quad [A(u) \vee B(x)] \qquad [\underline{\pi}, \overline{\pi}]$$

A.2.5 Instance 5 (15 Formulas) :

$$F_1 \equiv \forall t\, \forall v\, \exists x \quad [E(t,v) \vee \neg C(x)] \qquad [0.5170.567]$$

$$F_2 \equiv \forall t\, \exists v\, \forall x\, \forall s\, \exists u \quad [A(t) \vee F(v,x) \vee \neg H(s,u)] \qquad [0.6250.675]$$

$$F_3 \equiv \exists w\, \exists y\, \exists s\, \exists u \quad [G(w,y) \vee \neg B(w) \vee \neg H(s,u)] \qquad [0.80.85]$$

$$F_4 \equiv \forall t\, \exists w \quad [A(t) \vee B(w)] \qquad [0.6820.732]$$

$$F_5 \equiv \exists w \quad [\neg B(w)] \qquad [0.640.69]$$

$$F_6 \equiv \forall t\, \forall v\, \exists x\, \exists w \quad [A(t) \vee \neg F(v,x) \vee \neg B(w)] \qquad [0.7860.836]$$

$$F_7 \equiv \exists v\, \forall t\, \forall x \quad [D(v) \vee \neg E(t,v) \vee \neg C(x)] \qquad [0.7380.788]$$

$$F_8 \equiv \exists w\, \exists y \quad [G(w,y)] \qquad [0.5070.557]$$

$$F_9 \equiv \exists x \quad [C(w)] \qquad [0.850.9]$$

$$F_{10} \equiv \forall t\, \forall v \quad [A(t) \vee \neg D(v)] \qquad [0.7010.751]$$

$$F_{11} \equiv \exists v\, \exists x \quad [F(v,x)] \qquad [0.5990.649]$$

$$F_{12} \equiv \exists t\, \exists x \quad [A(t) \vee \neg C(x)] \qquad [0.6290.679]$$

$$F_{13} \equiv \exists w\, \forall y\, \forall v\, \forall x \quad [G(w,y) \vee \neg F(v,x) \vee \neg B(w)] \qquad [0.80.85]$$

$$F_{14} \equiv \exists t\, \forall v\, \exists w \quad [E(t,v) \vee D(v) \vee B(w)] \qquad [0.8660.916]$$

$$F_{15} \equiv \forall w\, \forall y\, \forall t\, \forall v\, \exists x \quad [G(w,y) \vee \neg E(t,v) \vee \neg C(x)] \qquad [0.5010.551]$$

$$F_{16} \equiv \exists t\, \exists v \quad [A(t) \vee D(v)] \qquad [\underline{\pi}, \overline{\pi}]$$

A.2.6 Instance 6 (20 formulas):

$$F_1 \equiv \exists o\, \forall n \quad [E(o) \vee \neg A(n)] \qquad [0.98, 1.0]$$

$$F_2 \equiv \exists w\, \forall o\, \forall q\, \exists u \quad [E(o) \vee F(o,q) \vee \neg J(u,w)] \qquad [0.86, 0.91]$$

$$F_3 \equiv \forall r\, \exists y\, \forall q\, \forall s \quad [D(r) \vee \neg C(y) \vee \neg H(q,s)] \qquad [0.71, 0.76]$$

$$F_4 \equiv \exists o\, \exists q \quad [\neg I(o,q)] \qquad [0.53, 0.58]$$

$$F_5 \equiv \forall w\, \exists o\, \forall q \quad [B(w) \vee \neg F(o,q) \vee \neg I(o,q)] \qquad [0.76, 0.81]$$

$$F_6 \equiv \forall o\, \exists y\, \exists n \quad [E(o) \vee \neg C(y) \vee \neg A(n)] \qquad [0.68, 0.73]$$

$$F_7 \equiv \exists r \quad [D(r)] \qquad [0.39, 0.44]$$

$$F_8 \equiv \exists n \quad [A(n)] \qquad [0.77, 0.83]$$

$$F_9 \equiv \exists w\, \forall o \quad [B(w) \vee \neg E(o)] \qquad [0.68, 0.73]$$

$$F_{10} \equiv \exists o\, \exists q \quad [F(o,q)] \qquad [0.66, 0.71]$$

$$F_{11} \equiv \forall w\, \forall n \quad [B(w) \vee \neg A(n)] \qquad [0.55, 0.60]$$

$$F_{12} \equiv \forall r\, \exists o\, \exists q\, \exists y \quad [D(r) \vee \neg F(o,q) \vee \neg C(y)] \qquad [0.68, 0.73]$$

$$F_{13} \equiv \forall z\, \forall o\, \forall y\, \exists q \quad [G(z,o) \vee C(y) \vee I(o,q)] \qquad [0.94, 0.99]$$

$$F_{14} \equiv \forall r\, \exists z\, \forall o\, \forall n \quad [D(r) \vee \neg G(z,o) \vee \neg A(n)] \qquad [0.64, 0.69]$$

$$F_{15} \equiv \forall z\, \forall o\, \exists y \quad [G(z,o) \vee \neg C(y)] \qquad [0.50, 0.55]$$

$$F_{16} \equiv \forall q\, \exists s\, \exists w \quad [\neg H(q,s) \vee \neg B(w)] \qquad [0.71, 0.76]$$

$$F_{17} \equiv \forall q\, \exists s\, \forall z\, \exists o \quad [H(q,s) \vee G(z,o) \vee \neg E(o)] \qquad [0.98, 1.0]$$

$$F_{18} \equiv \forall r\, \exists n\, \exists u\, \exists w \quad [D(r) \vee \neg A(n) \vee \neg J(u,w)] \qquad [0.98, 1.0]$$

$$F_{19} \equiv \exists o \quad [E(o)] \qquad [0.98, 1.0]$$

$$F_{20} \equiv \exists n\, \exists r \quad [\neg A(n) \vee \neg D(r)][0.98, 1.0]$$

$$F_{21} \equiv \exists w\, \exists y \quad [B(w) \vee C(y)] \qquad [\underline{\pi}, \overline{\pi}]$$

A.2.7 Instance 7 (25 formulas) :

$$F_1 \equiv \forall l\, \forall t \quad [E(l) \vee \neg C(t)] \qquad [0.73, 0.78]$$

$$F_2 \equiv \exists r\, \exists x\, \exists z\, \exists v \quad [A(r) \vee J(x, z) \vee \neg L(v, x)] \qquad [0.61, 0.66]$$

$$F_3 \equiv \exists n\, \forall q\, \exists u\, \forall w \quad [D(n) \vee \neg B(q) \vee \neg K(u, w)] \quad [0.68, 0.72]$$

$$F_4 \equiv \forall r\, \exists q \quad [A(r) \vee B(q)] \qquad [0.74, 0.79]$$

$$F_5 \equiv \exists q \quad [\neg B(q)] \qquad [0.62, 0.67]$$

$$F_6 \equiv \exists r\, \exists z\, \forall q \quad [A(r) \vee \neg F(z) \vee \neg B(q)] \qquad [0.72, 0.79]$$

$$F_7 \equiv \forall l\, \forall t\, \forall v\, \exists u\, \exists w \quad [E(l) \vee \neg I(t, v) \vee \neg G(u, w)][0.76, 0.81]$$

$$F_8 \equiv \exists w\, \exists y \quad [H(w, y)] \qquad [0.63, 0.68]$$

$$F_9 \equiv \exists u\, \exists w \quad [G(u, w)] \qquad [0.61, 0.66]$$

$$F_{10} \equiv \exists r\, \forall l \quad [A(r) \vee \neg E(l)] \qquad [0.55, 0.60]$$

$$F_{11} \equiv \exists x\, \exists z \quad [J(x, z)] \qquad [0.50, 0.55]$$

$$F_{12} \equiv \exists z \quad [\neg F(z)] \qquad [0.71, 0.76]$$

$$F_{13} \equiv \exists n \quad [\neg D(n)] \qquad [0.81, 0.86]$$

$$F_{14} \equiv \forall r\, \forall t \quad [A(r) \vee \neg C(t)] \qquad [0.71, 0.76]$$

$$F_{15} \equiv \exists w\, \forall y\, \exists z\, \forall q \quad [H(w, y) \vee \neg F(z) \vee \neg B(q)] \qquad [0.98, 1.0]$$

$$F_{16} \equiv \forall n\, \forall t\, \exists u\, \exists w \quad [D(n) \vee \neg C(t) \vee \neg G(u, w)] \qquad [0.98, 1.0]$$

$$F_{17} \equiv \forall m\, \exists o\, \exists q \quad [M(m, o) \vee \neg B(q)] \qquad [0.98, 1.0]$$

$$F_{18} \equiv \forall u\, \exists w\, \forall y \quad [K(u, w) \vee \neg H(w, y)] \qquad [0.98, 1.0]$$

$$F_{19} \equiv \exists u\, \forall w\, \exists t\, \exists m\, \forall o \quad [K(u, w) \vee C(t) \vee \neg M(m, o)] \qquad [0.98, 1.0]$$

$$F_{20} \equiv \exists n\, \exists t\, \exists v\, \forall x \quad [D(n) \vee \neg C(t) \vee \neg L(v, x)] \qquad [0.98, 1.0]$$

$$F_{21} \equiv \exists l \quad [E(l)] \qquad [0.98, 1.0]$$

$$F_{22} \equiv \forall q\, \forall t\, \exists v\, \forall n \quad [\neg B(q) \vee \neg I(t, v) \vee \neg D(n)] \qquad [0.98, 1.0]$$

$$F_{23} \equiv \exists u\, \forall w\, \exists y \quad [\neg G(u, w) \vee \neg H(w, y)] \qquad [0.98, 1.0]$$

$$F_{24} \equiv \exists t \quad [\neg C(t)] \qquad [0.98, 1.0]$$

$$F_{25} \equiv \forall r\, \forall x\, \forall z \quad [A(r) \vee \neg J(x, z) \vee negF(z)] \qquad [0.98, 1.0]$$

$$F_{26} \equiv \exists r\, \exists t\, \exists v \quad [A(r) \vee I(t, v)] \qquad [\underline{\pi}, \overline{\pi}]$$

A.2.8 Instance 8 (30 formulas):

$$F_1 \equiv \exists p\, \exists k \quad [F(p) \vee \neg D(k)] \qquad [0.84, 0.89]$$

$$F_2 \equiv \exists r\, \exists x\, \exists z \quad [\neg M(i,k) \vee G(r) \vee \neg K(i,k)] \qquad [0.53, 0.58]$$

$$F_3 \equiv \forall r\, \exists j\, \forall s\, \forall u \quad [\neg B(q) \vee \neg I(t,v) \vee \neg D(n)] \qquad [0.98, 1.0]$$

$$F_4 \equiv \exists j\, \exists l \quad [\neg I(j,l)] \qquad [0.48, 0.53]$$

$$F_5 \equiv \exists w\, \exists o\, \exists q\, \forall j\, \forall l \quad [\neg B(w) \vee J(o,q) \vee \neg I(j,l)] \quad [0.98, 1.0]$$

$$F_6 \equiv \forall p\, \forall q\, \exists s\, \exists k \quad [F(p) \vee \neg L(q,s) \vee \neg D(k)] \qquad [0.67, 0.72]$$

$$F_7 \equiv \exists r \quad [E(r)] \qquad [0.51, 0.56]$$

$$F_8 \equiv \exists k \quad [D(k)] \qquad [0.75, 0.80]$$

$$F_9 \equiv \exists w\, \forall p \quad [B(w) \vee \neg F(p)] \qquad [0.61, 0.66]$$

$$F_{10} \equiv \exists r \quad [G(r)] \qquad [0.48, 0.54]$$

$$F_{11} \equiv \forall w\, \forall k \quad [B(w) \vee \neg C(k)] \qquad [0.41, 0.46]$$

$$F_{12} \equiv \exists p\, \forall q\, \exists s\, \forall k \quad [E(r) \vee \neg J(o,q) \vee \neg A(j)] \qquad [0.98, 1.0]$$

$$F_{13} \equiv \forall r\, \forall k \quad [E(r) \vee \neg C(k) \vee \neg D(k)] \qquad [0.71, 0.76]$$

$$F_{14} \equiv \forall r\, \exists t\, \forall j \quad [N(r,t) \vee \neg A(j)] \qquad [0.98, 1.0]$$

$$F_{15} \equiv \exists s\, \exists u\, \forall i\, \exists k \quad [\neg H(s,u) \vee \neg O(i,k)] \qquad [0.53, 0.58]$$

$$F_{16} \equiv \forall s\, \exists u\, \forall k\, \exists p \quad [\neg H(s,u) \vee C(k) \vee \neg F(p)] \qquad [0.93, 0.98]$$

$$F_{17} \equiv \exists r\, \forall t\, \exists j \quad [E(r) \vee \neg C(k) \vee K(i,k)] \qquad [0.88, 0.93]$$

$$F_{18} \equiv \exists p \quad [F(p)] \qquad [0.71, 0.76]$$

$$F_{19} \equiv \exists j\, \exists l\, \forall i\, \forall k \quad [\neg I(j,l) \vee \neg A(j) \vee \neg K(i,k)] \qquad [0.98, 1.0]$$

$$F_{20} \equiv \forall k\, \exists r \quad [\neg D(k) \vee \neg E(r)] \qquad [0.83, 0.88]$$

$$F_{21} \equiv \exists k \quad [C(k)] \qquad [0.98, 1.0]$$

$$F_{22} \equiv \forall w\, \exists o\, \forall q\, \exists j\, \forall l \quad [M(i,k) \vee \neg G(r) \vee \neg J(o,q)] \qquad [0.98, 1.0]$$

$$F_{23} \equiv \forall k\, \forall r\, \exists t \quad [C(k) \vee \neg N(r,t) \vee G(r)] \qquad [0.98, 1.0]$$

$$F_{24} \equiv \forall w\, \exists j\, \forall l \quad [B(w) \vee I(j,l)] \qquad [0.98, 1.0]$$

$$F_{25} \equiv \exists j\, \forall k \quad [A(j) \vee \neg C(k)] \qquad [0.98, 1.0]$$

$$F_{26} \equiv \exists w\, \exists o\, \forall q \quad [B(w) \vee \neg J(o,q)] \qquad [0.98, 1.0]$$

$$F_{27} \equiv \exists k\, \forall q\, \exists s \quad [D(k) \vee L(q,s)] \qquad [0.98, 1.0]$$

$$F_{28} \equiv \exists j\, \exists i\, \exists k \quad [\neg A(j) \vee O(i,k)] \qquad [0.98, 1.0]$$

$$F_{29} \equiv \exists j\, \forall s\, \forall u\, \exists w \quad [A(j) \vee \neg H(s,u) \vee \neg B(w)] \qquad [0.98, 1.0]$$

$$F_{30} \equiv \exists q\, \exists s \quad [\neg L(q,s)] \qquad [0.98, 1.0]$$

$$F_{31} \equiv \exists w\, \exists j \quad [B(w) \vee A(j)] \qquad [\underline{\pi}, \overline{\pi}]$$

A.2.9 Instance 9 (35 formulas):

$$F_1 \equiv \exists e\, \exists o \quad [G(e) \vee \neg B(o)] \qquad [0.79, 0.84]$$

$$F_2 \equiv \exists v\, \forall f\, \exists h\, \forall l\, \forall n \quad [A(v) \vee J(f,h) \vee \neg O(l,n)] \quad [0.65, 0.7]$$

$$F_3 \equiv \exists k\, \exists v\, \exists i \quad [H(k) \vee \neg C(v) \vee \neg P(i,k)] \qquad [0.65, 0.70]$$

$$F_4 \equiv \exists v \quad [A(v) \vee C(v)] \qquad [0.98, 1.0]$$

$$F_5 \equiv \exists r \quad [\neg D(r)] \qquad [0.69, 0.74]$$

$$F_6 \equiv \forall v\, \forall m\, \forall o\, \exists r \quad [A(v) \vee \neg K(m,o) \vee \neg D(r)] \quad [0.98, 1.0]$$

$$F_7 \equiv \exists y\, \exists e\, \forall t\, \forall o \quad [Q(y,e) \vee \neg F(t) \vee \neg B(o)] \qquad [0.98, 1.0]$$

$$F_8 \equiv \exists k \quad [H(k)] \qquad [0.31, 0.37]$$

$$F_9 \equiv \exists o \quad [\neg B(o)] \qquad [0.69, 0.74]$$

$$F_{10} \equiv \forall v\, \exists e \quad [A(v) \vee \neg G(e)] \qquad [0.66, 0.71]$$

$$F_{11} \equiv \exists f\, \exists h \quad [J(f,h)] \qquad [0.49, 0.54]$$

$$F_{12} \equiv \forall v\, \exists o \quad [A(v) \vee \neg B(o)] \qquad [0.73, 0.78]$$

$$F_{13} \equiv \forall k\, \forall m\, \forall o\, \forall v \quad [H(k) \vee \neg K(m,o) \vee \neg C(v)] \quad [0.79, 0.84]$$

$$F_{14} \equiv \exists l\, \forall n\, \exists v\, \exists r \quad [L(l,n) \vee C(v) \vee D(r)] \qquad [0.98, 1.0]$$

$$F_{15} \equiv \forall u\, \exists w\, \forall f\, \exists o \quad [M(u,w) \vee \neg E(f) \vee \neg B(o)] \quad [0.81, 0.86]$$

$$F_3 \equiv \forall l\, \exists n\, \exists v \quad [L(l,n) \vee \neg C(v)] \qquad [0.56, 0.61]$$

$$F_{17} \equiv \forall h\, \exists j \quad [\neg N(h,j) \vee \neg I(h,j)] \qquad [0.56, 0.61]$$

$$F_{18} \equiv \exists h\, \exists j\, \forall f\, \forall e \quad [N(h,j) \vee E(f) \vee \neg G(e)] \qquad [0.98, 1.0]$$

$$F_{19} \equiv \forall k\, \exists f\, \forall l\, \exists n \quad [H(k) \vee \neg E(f) \vee \neg O(l,n)] \quad [0.82, 0.87]$$

$$F_{20} \equiv \exists y\ \exists e \quad [Q(y,e)] \qquad [0.98, 1.0]$$

$$F_{21} \equiv \forall r\ \exists t\ \forall u\ \forall w \quad [\neg D(r) \vee \neg F(t) \vee \neg m(u,w)] \qquad [0.98, 1.0]$$

$$F_{22} \equiv \forall h\ \exists j\ \forall v\ \exists e \quad [I(h,j) \vee C(v) \vee \neg G(e)] \qquad [0.98, 1.0]$$

$$F_{23} \equiv \forall o\ \exists k \quad [\neg B(o) \vee \neg H(k)] \qquad [0.98, 1.0]$$

$$F_{24} \equiv \exists f \quad [\neg E(f)] \qquad [0.98, 1.0]$$

$$F_{25} \equiv \forall v\ \exists f\ \exists h\ \forall m\ \exists o \quad [A(v) \vee \neg J(f,h) \vee \neg K(m,o)] \qquad [0.98, 1.0]$$

$$F_{26} \equiv \forall f\ \exists l\ \exists n\ \exists h \quad [E(f) \vee L(l,n) \vee J(f,h)] \qquad [0.98, 1.0]$$

$$F_{27} \equiv \exists l\ \exists n\ \forall v \quad [I(h,j) \vee C(r)] \qquad [0.98, 1.0]$$

$$F_{28} \equiv \forall t\ \exists f \quad [F(t) \vee \neg E(f)] \qquad [0.98, 1.0]$$

$$F_{29} \equiv \forall v\ \exists m\ \exists o \quad [A(v) \vee \neg K(m,o)] \qquad [0.98, 1.0]$$

$$F_{30} \equiv \exists t\ \exists h\ \forall j\ \exists v \quad [F(t) \vee \neg N(h,j) \vee \neg A(v)] \qquad [0.98, 1.0]$$

$$F_{31} \equiv \exists v \quad [\neg C(v)] \qquad [0.0, 0.03]$$

$$F_{32} \equiv \exists u\ \exists w\ \exists e \quad [M(u,w) \vee G(e)] \qquad [0.98, 1.0]$$

$$F_{33} \equiv \forall u\ \forall w\ \exists l\ \exists n\ \exists i\ \forall k \quad [M(u,w) \vee L(l,n) \vee \neg P(i,k)] \qquad [0.98, 1.0]$$

$$F_{34} \equiv \exists h\ \exists j \quad [N(h,j)] \qquad [0.98, 1.0]$$

$$F_{35} \equiv \forall r\ \forall o \quad [\neg D(r) \vee \neg B(o)] \qquad [0.98, 1.0]$$

$$F_{36} \equiv \exists h\ \exists j\ \exists t \quad [I(h,j) \vee F(t)] \qquad [\underline{\pi}, \overline{\pi}]$$

A.3 Inference Rules

Table 8: Examples of inference rules for propositional probabilistic logic.

Inference Rules	Probability Assigned	Consistency Conditions	Probability Bounds
$S: x_1 \vee x_2$ x_1 x_2	$[\underline{\pi}_S, \overline{\pi}_S]$ $[\underline{\pi}_1, \overline{\pi}_1]$ π_2 ?	$\underline{\pi}_i \leq 1 \quad i = 1, S$ $\overline{\pi}_i \geq 0 \quad i = 1, S$ $\underline{\pi}_i \leq \overline{\pi}_i \quad i = 1, S$ $\overline{\pi}_S \geq \underline{\pi}_1$	$\underline{\pi}_2 = \max\{0, \underline{\pi}_S - \overline{\pi}_1\}$ $\overline{\pi}_2 = \min\{1, \overline{\pi}_S\}$
$S: x_1 \vee x_2 \vee x_3$ x_1 x_2 x_3	$[\underline{\pi}_S, \overline{\pi}_S]$ $[\underline{\pi}_1, \overline{\pi}_1]$ $[\underline{\pi}_2, \overline{\pi}_2]$ π_3 ?	$\underline{\pi}_i \leq 1 \quad i = 1, 2, S$ $\overline{\pi}_i \geq 0 \quad i = 1, 2, S$ $\underline{\pi}_i \leq \overline{\pi}_i \quad i = 1, 2, S$ $\overline{\pi}_S \geq \underline{\pi}_2$ $\overline{\pi}_S \geq \underline{\pi}_1$	$\underline{\pi}_3 = \max\{0, \underline{\pi}_S - \overline{\pi}_1 - \overline{\pi}_2\}$ $\overline{\pi}_3 = \min\{1, \overline{\pi}_S\}$
$S: x_1 \vee x_2 \vee x_3 \vee x_4$ x_1 x_2 x_3 x_4	$[\underline{\pi}_S, \overline{\pi}_S]$ $[\underline{\pi}_1, \overline{\pi}_1]$ $[\underline{\pi}_2, \overline{\pi}_2]$ $[\underline{\pi}_3, \overline{\pi}_3]$ π_4?	$\underline{\pi}_i \leq 1 \quad i = 1, 2, 3, S$ $\overline{\pi}_i \geq 0 \quad i = 1, 2, 3, S$ $\underline{\pi}_i \leq \overline{\pi}_i \quad i = 1, 2, 3, S$ $\overline{\pi}_S \geq \underline{\pi}_1, \overline{\pi}_S \geq \underline{\pi}_2$ $\overline{\pi}_S \geq \underline{\pi}_3$	$\underline{\pi}_4 = \max\{0, \underline{\pi}_S - \overline{\pi}_1 - \overline{\pi}_2 - \overline{\pi}_3\}$ $\overline{\pi}_4 = \min\{1, \overline{\pi}_S\}$
$S_1: \overline{x}_1 \vee x_2$ $S_2: x_1 \vee \overline{x}_2$ x_1 x_2	$[\underline{\pi}_{S_1}, \overline{\pi}_{S_1}]$ $[\underline{\pi}_{S_2}, \overline{\pi}_{S_2}]$ $[\underline{\pi}_1, \overline{\pi}_1]$ π_2?	$\overline{\pi}_i \geq 0 \quad i = 1, S_1, S_2$ $\underline{\pi}_i \leq \overline{\pi}_i \quad i = 1, S_1, S_2$ $\underline{\pi}_i \leq 1 \quad i = 1, S_1, S_2$ $\underline{\pi}_1 \leq \overline{\pi}_{S_2}$ $\overline{\pi}_1 + \overline{\pi}_{S_2} \geq 1$ $\overline{\pi}_{S_1} + \overline{\pi}_{S_2} \geq 1$	$\underline{\pi}_2 = \max\{1 - \overline{\pi}_{S_2}, \underline{\pi}_1 + \underline{\pi}_{S_1} - \overline{\pi}_{S_2}, \underline{\pi}_1 + \underline{\pi}_{S_1} - 1\}$ $\overline{\pi}_2 = \min\{\overline{\pi}_{S_1}, \overline{\pi}_1 + \overline{\pi}_{S_1} - \underline{\pi}_{S_2}, 1 + \overline{\pi}_1 - \underline{\pi}_{S_2}\}$
$S_1: \overline{x}_1 \vee \overline{x}_3$ $S_2: \overline{x}_2 \vee x_3$ x_1 x_2 x_3 $S_3: \overline{x}_1 \vee \overline{x}_2$	$[\underline{\pi}_{S_1}, \overline{\pi}_{S_1}]$ $[\underline{\pi}_{S_2}, \overline{\pi}_{S_2}]$ $[\underline{\pi}_1, \overline{\pi}_1]$ $[\underline{\pi}_2, \overline{\pi}_2]$ $[\underline{\pi}_3, \overline{\pi}_3]$ π_{S_3} ?	$\underline{\pi}_i \leq 1 \quad i = 1, 2, 3, S_1, S_2$ $\overline{\pi}_i \geq 0 \quad i = 1, 2, 3, S_1, S_2$ $\underline{\pi}_i \leq \overline{\pi}_i \quad i = 1, 2, 3, S_1, S_2$ $\underline{\pi}_3 \leq \overline{\pi}_{S_2}$ $\overline{\pi}_i + \overline{\pi}_{S_1} \geq 1 \quad i = 1, S_2, 3$ $\overline{\pi}_2 + \overline{\pi}_{S_2} \leq 1$ $\underline{\pi}_2 + \underline{\pi}_{S_2} - \overline{\pi}_3 \leq 1$ $\underline{\pi}_1 + \underline{\pi}_3 + \underline{\pi}_{S_1} \leq 2$ $\underline{\pi}_1 + \underline{\pi}_2 + \underline{\pi}_{S_1} + \underline{\pi}_{S_2} \leq 3$	$\underline{\pi}_{S_3} = \max\{0, 1 - \overline{\pi}_1, 1 - \overline{\pi}_2, \underline{\pi}_{S_2} - \overline{\pi}_3, \underline{\pi}_{S_1} + \underline{\pi}_{S_2} - 1, \underline{\pi}_3 + \underline{\pi}_{S_1} - 1, 3 - \overline{\pi}_1 - \overline{\pi}_2 - \overline{\pi}_{S_1} - \overline{\pi}_{S_2}\}$ $\overline{\pi}_{S_3} = \min\{1, 1 + \overline{\pi}_{S_1} - \underline{\pi}_2, 1 + \overline{\pi}_{S_2} - \underline{\pi}_1, \overline{\pi}_{S_2} - \underline{\pi}_1 - \underline{\pi}_3 - \underline{\pi}_{S_1} + 2, 2 - \underline{\pi}_1 - \underline{\pi}_2, \overline{\pi}_3 + \overline{\pi}_{S_1} - \underline{\pi}_2 - \underline{\pi}_{S_2} + 1\}$

Bibliography

[1] S. Amarger, D. Dubois, and H. Prade. Constraint propagation with imprecise conditional probabilities. In *Proc. of the seventh conference on Uncertainty in artificial intelligence*, pages 26–34, 1994.

[2] B. Aspvall, M.F. Plass, and R.E. Tarjan. A linear-time algorithm for testing the truth of certain quantified boolean formulas. *Information Processing Letters*, 8(3):121–123, 1979.

[3] F. Bacchus. *Representing and Reasoning with Probabilistic Knowledge.* The MIT Press, Cambridge, Massachusetts, 1990.

[4] J. F. Baldwin and M. R. M. Monk. Evidence theory, fuzzy logic and logic programming. *Tech. Report ITRC No. 109. University of Bristol, Bristol, UK*, 1987.

[5] H. A. Blair and V. S. Subrahmanian. Paraconsistent foundations for logic programming. *Journal of Non-classical Logic*, 5(2):45–73, 1989.

[6] G. Boole. Proposed question in the theory of probabilities. *The Cambridge and Dublin Mathematical Journal*, 6(186):268–269, 1851.

[7] G. Boole. *The Laws of Thoughts.* Dover Publications, 1854.

[8] R. Carnap. *The Logical Foundations of Probability.* University of Chicago Press, 1962.

[9] C.L. Chang and R.C. Lee. *Symbolic Logic and Mechanical Theorem Proving.* Academic Press, 1973.

[10] A. Church. An unsolvable problem of elementary number theory. *American Journal of Mathematics*, 2:345–363, 1936.

[11] V. Chvatal. *Linear Programming.* W.H. Freeman and Company, 1983.

[12] W.L. Craig. Incompatibility, with respect to validity in every finite nonempty domain, of first order functional calculus. In *Proc. of the International Congress of Mathematicians*, 1950.

[13] B. Dreben and W.D. Goldfarb. *The Decision Problem: Solvable Classes of Quantified Formulas.* Addison-Wesley, 1979.

[14] D. Dubois and H. Prade. *Possibility Theory.* Plenum Press, New York, 1988.

[15] A.M. Frisch and P. Haddawy. Anytime deduction for probabilistic logic. *Artificial Intelligence*, 69(1-2):93–122, 1994.

[16] G. Georgakopoulos, D. Kavvadias, and C.H. Papadimitriou. Probabilistic satisfiability. *Journal of Complexity*, 4:1–11, 1988.

[17] E. Grädel, P.G. Kolaitis, and M.Y. Vardi. On the decision problem for two-variable first-order logic. *The Bulletin of Symbolic Logic*, 3:53–69, 1997.

[18] I. Hacking. *Logic of Statistical Inference.* Cambridge University Press, Cambridge, U.K., 1965.

[19] T. Hailperin. Best possible inequalities for the probability of a logical function of events. *Monthly American Mathematics*, 72:343–359, 1965.

[20] T. Hailperin. *Boole's Logic and Probability.* North-Holland, Amsterdam, 1986.

[21] T. Hailperin. *Sentential Probability Logic.* Lehigh University Press, Bethelehem and associated University Press, London, 1996.

[22] J.Y. Halpern. An analysis of first-order logics of probability. *Artificial Intelligence*, 46(3):311–350, 1990.

[23] P. Hansen and B. Jaumard. Probabilistic satisfiability. 2000.

[24] P. Hansen, B. Jaumard, and M.P.de Arago. Boole's conditions of possible experience and reasoning under uncertainty. *Discrete Applied Mathematics*, 60:181–193, 1995.

[25] P. Hansen, B. Jaumard, M.P.de Arago, and C.C. de Souza. Correctness of anytime deduction for probabilistic logic. *Les Cahiers du GERAD G-97-02, Groupe d'Etudes et de Recherche en Analyse des Decisions*, 1997.

[26] P. Hansen and S. Perron. Merging the local and global approaches to probabilistic satisfiability. *International Journal of Approximate Reasoning*, 47(2):125–140, 2007.

[27] J. E. Hopcroft, R. Motwani, and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation (2nd Edition).* Addison Wesley, 2000.

[28] U. Hustadt, R. A. Schmidt, and L. Georgieva. A survey of decidable first-order fragments and description logics. *Journal on Relation Methods in Computer Science*, 1:251–276, 2004.

[29] B. Jaumard, A. Fortin, I. Shahriar, and R. Sultana. First order probabilistic logic. In *Fuzzy Information Processing Society, 2006. NAFIPS 2006. Annual meeting of the North American*, pages 341–346, 2006.

[30] B. Jaumard, P. Hansen, and M.P.de Arago. Column generation methods for probabilistic logic. *ORSA Journal on Computing*, 3(2):135–148, 1991.

[31] B. Jaumard and A. Nongaillard. Automated mechanism design: using column generation for the design of multi-agent exchanges. *Web Intelligence and Agent Systems: An International Journal, submitted*, 2008.

[32] B. Jaumard and A. Parreira. An anytime deduction algorithm for the probabilistic logic and entailment problems. *Fuzzy Information Processing Society, 2006. NAFIPS 2006. Annual meeting of the North American, IEEE*, 2007.

[33] B. Jaumard and A.D. Parreira. An anytime deduction algorithm for the probabilistic logic and entailment problems. *Les Cahiers du GERAD G-2006*, 79, 2006.

[34] D. Jovanović, N. Mladenović, and Z. Ognjanović. Variable neighborhood search for the probabilistic satisfiability problem. In *The 6th Metaheuristics International conference, MIC 2005*, 2005.

[35] M. Kifer and A. Li. On the semantics of rule-based expert systems with uncertainty. In *2nd Int. Conf. on Database Theory*, pages 102–117, 1988.

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

[37] G.J. Klir and B. Yuan. *Fuzzy Sets and Fuzzy Logic Theory and Applications.* Prentice Hall PTR, first edition, 1995.

[38] L. V. S. Lakshmanan and F. Sadri. Probabilistic deductive databases. In *Proc. Int. Logic Programming Symposium*, pages 254–268, 1994.

[39] L.V.S. Lakshmanan and F. Sadri. On a theory of probabilistic deductive databases. *Journal of Logic Programming*, 1(1):5–35, 2001.

[40] G.F. Luger. *Artificial Intelligence Structures and Strategies for Complex Problem Solving.* Pearson Education, fourth edition, 2002.

[41] T. Lukasiewicz. Magic inference rules for probabilistic deduction under taxonomic knowledge. In *Proc. of the 14th Conference on Uncertainty in Artificial Intelligence*, pages 354–361, 1998.

[42] T. Lukasiewicz. Probabilistic logic programming. In *Proc. of the 13th Biennial European Conference on Artificial Intelligence*, pages 388–392, 1998.

[43] T. Lukasiewicz. Local probabilistic deduction from taxonomic and probabilistic knowledge-bases over conjunctive events. volume 21, pages 23–61, 1999.

[44] T. Lukasiewicz. Probabilistic deduction with conditional constraints over basic events. *Journal of Artificial Intelligence Research*, 10:199–241, 1999.

[45] B. Milch and S. Russell. First-order probabilistic languages: Into the unknown. In *Proc. of the 16th International Conference on Inductive Logic Programming. Berlin: Springer*, 2007.

[46] N. Mladenović and P. Hansen. Variable neighborhood search. *Computational Operations Research*, 24(11):1097–1100, 1997.

[47] M. Mortazavi. Logics of probabilistic reasoning and imperfect agents. *To appear.*, 2007.

[48] M. Newborn. Automated theorem proving: Theory and practice. *Springer*, 2005.

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

[50] N.J. Nilsson. Probabilistic logic. *Artificial Intelligence*, 28(1):71–87, 1986.

[51] N.J. Nilsson. Probabilistic logic revised. *Artificial Intelligence*, 59(1-2):39–42, 1993.

[52] D. Page and A. Srinivasan. Ilp: a short look back and a longer look forward. *Journal of Machine Learning Research*, 4:415–430, 2003.

[53] W. C. Purdy. Inexpressiveness of first-order fragments. *Australasian Journal of Logic*, 1(2):1–12, 2006.

[54] A. Riazanov and A. Voronkov. Adaptive saturation-based reasoning. In *PSI: 4th International Andrei Ershov Memorial Conference, Revised Papers*, volume 2244, pages 1–12, 2001.

[55] A. Riazanov and A. Voronkov. Splitting without backtracking. In *Proc. of International Joint Conferences on Artificial Intelligence, IJCAI*, volume 1, pages 611–617, 2001.

[56] A. Riazanov and A. Voronkov. Limited resource strategy in resolution theorem proving. *Journal of Symbolic Computation*, 36:101–115, 2003.

[57] A. Riazanov and A. Voronkov. Efficient instance retrieval with standard and relational path indexing. In *19th International Conference on Automated Deduction*, volume 199, pages 228–252, 2005.

[58] G. Rozenberg and A. Salomaa. *Cornerstones of Undecidability.* Prentice Hall, 1994.

[59] H. Schmidt, N. Steger, U. Guntzer, W. Kiebling, A. Azone, and R. Bayer. Combining deduction by certainty with the power of magic. In *Proc. 1st Int. Conf. on Deductive and Object-oriented Databases*, pages 103–122, 1989.

[60] M. Smithson. *Ignorance and Uncertainty: Emerging Paradigms.* Springer Verlag, New York, 1989.

[61] N. Steger, H. Schmidt, U. Guntzer, and W. Kiebling. Semantics and efficient compilation for quantitative deductive databases. In *Proc. IEEE Int. Conf. on Data Engineering.*, pages 660–669, 1989.

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

[63] R. Sultana. An exact algorithm for first-order probabilistic logic. Master's thesis, Concordia University, Montreal, Canada, 2007.

[64] B. Trakhtenbrot. The impossibility of an algorithm for the decision problem for finite models. *Dokl. Akad. Nauk SSSR, 70:596–572, 1950. English translation in: AMS Transl. Ser. 2*, 23(1063):1–6, 1950.

[65] A.M. Turing. On computable numbers, with an application to the entscheidungsproblem. In *London Mathematical Society*, volume 43, pages 544–546, 1937.

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

[67] L. Wos and G. Robinson. Paramodulation and set of support. In *IRIA Symposium on Automatic Demonstration at Versailles, France*, 1968.

[68] L. Zadeh. Fuzzy sets. *Information and Control*, 8:338–353, 1965.

[69] L. Zadeh. Fuzzy sets as a basis for a theory of possibility. *Fuzzy Sets and Systems*, 1:3–28, 1978.