Hybrid Probabilistic Programs[*]

Alex Dekhtyar[†] and *V.S.Subrahmanian*[‡]

Department of Computer Science
A. V. Williams Building
University of Maryland
College Park, Maryland 20742, U.S.A.

Abstract

The precise probability of a compound event (e.g. $e_1 \vee e_2, e_1 \wedge e_2$) depends upon the known relationships (e.g. independence, mutual exclusion, ignorance of any relationship, etc.) between the primitive events that constitute the compound event. To date, most research on probabilistic logic programming [20, 19, 22, 23, 24] has assumed that we are ignorant of the relationship between primitive events. Likewise, most research in AI (e.g. Bayesian approaches) have assumed that primitive events are independent. In this paper, we propose a *hybrid* probabilistic logic programming language in which the user can explicitly associate, with any given probabilistic strategy, a conjunction and disjunction operator, and then write programs using these operators. We describe the syntax of hybrid probabilistic programs, and develop a model theory and fixpoint theory for such programs. Last, but not least, we develop three alternative procedures to answer queries, each of which is guaranteed to be sound and complete.

1 Introduction

Though there has now been considerable work in the area of *quantitative logic programming* [1, 12, 27, 31, 17], there has been relatively little work in the area of probabilistic logic programming [20, 19, 22, 23, 24]. The reason for this is that while connectives in multivalued logics can be interpreted in terms of the lattice's LUB (for disjunction) and GLB (for conjunction) operators, the same is not true in the case of probabilities. In particular, there is no single "formula" for computing the probability of a complex event $(e_1 \wedge e_2)$ where e_1, e_2 are primitive events. For instance:

1. If e_1, e_2 are *independent*, then $\mathbf{Prob}(e_1 \wedge e_2) = \mathbf{Prob}(e_1) \times \mathbf{Prob}(e_2)$.
2. If we are *ignorant* about the relationship between e_1, e_2, then all we can say[22] is that $\mathbf{Prob}(e_1 \wedge e_2)$ lies in the interval:
$$[\max(0, \mathbf{Prob}(e_1) + \mathbf{Prob}(e_2) - 1), \min(\mathbf{Prob}(e_1), \mathbf{Prob}(e_2))]$$
3. If we know that e_1, e_2 are *mutually exclusive*, then $\mathbf{Prob}(e_1 \wedge e_2) = 0$.
4. If we know that event e_1 implies event e_2 (called *positive correlation*), then $\mathbf{Prob}(e_1 \wedge e_2) = \mathbf{Prob}(e_1)$.

[*]http://www.cs.umd.edu/users/vs/papers/probpapers.html contains proofs of all results in this paper.
[†]Electronic Mail: dekhtyar@cs.umd.edu
[‡]Electronic Mail: vs@cs.umd.edu

The above list represents a small fraction of relationships between events, each leading to different possible probabilities for complex events such as $(e_1 \wedge e_2)$. *The same holds for disjunctive events as well.*

In previous efforts, probabilistic logic programming has assumed a fixed probabilistic strategy[20, 19, 22, 23, 24], such as (i) ignorance of the dependencies between events, or (ii) independence between events. However, an end user writing a probabilistic logic program should have the flexibility to write rules that reflect his/her specific knowledge about dependencies between events. For instance, the user should be able to express statements such as the two given below, that allow the user to explicitly articulate the probabilistic dependencies between events.

- "If the chairman of company C sells his stock and the chairman retires, *and we are ignorant of the dependencies between these two events*, then conclude that the stock in company C will drop, with probability between 40–90%."
- "If the chairman of company C sells his stock and the chairman retires, *and the retirement implies sale of stock* (e.g. in an employee owned company), then conclude that the stock in company C will drop, with probability between 5–20%."

Both rules above refer to the same two events, viz. sale of stock by the chairman, and retirement of the chairman. However, the first rules specifies what to conclude if we are ignorant of the relationship between these two events, while the second explicitly encodes specific knowledge about the dependencies between events. The rules lead to very different conclusions.

In this paper, we make the following contributions:

1. First, we define a general axiomatic notion of a *probabilistic strategy*. We show how a number of well known probabilistic strategies are special cases of our definition.
2. We then define the concept of a *hybrid probabilistic program* (*hp-program*, for short). If the user selects a set of probabilistic strategies $i_1, \ldots, i_k$ for use in an hp-program (s/he may select these in any way, as long as these selections satisfy the axioms defining probabilistic strategies), then this automatically defines a set of conjunction and disjunction connectives.
3. Subsequently, we define a fixpoint semantics for hp-programs, a model theoretic semantics for hp-programs, and a proof procedure, and prove that the fixpoint theory, model theory, and proof theory all lead to equivalent characterizations. *This applies to any selection of probabilistic strategies made by the user, as long as these selections satisfy the axioms defining probabilistic strategies.*

2 Probabilistic Strategies (p-strategies)

In this section, we provide an axiomatic definition of probabilistic strategies (p-strategies). As we have already seen in the Introduction (cf. the ignorance strategy), the probability of a compound event may be an *interval*, rather than a point, even if point probabilities are known for the primitive events involved. This was first shown by Boole[4] in 1854. Thus, p-strategies will be defined on intervals – points, in any case, are special cases of intervals.

Let $\mathcal{C}[0,1]$ denote the set of all closed intervals of $[0,1]$. Let $\mathcal{PC}[0,1]$ denote the powerset of $\mathcal{C}[0,1]$. If $[a,b] \in \mathcal{C}[0,1], [c,d] \in \mathcal{C}[0,1]$ then we write $[a,b] \leq_t [c,d]$ if $a \leq c$ *and* $b \leq d$. A probabilistic strategy, defined below, is a pair of functions that satisfy certain properties.

Definition 1 *A probabilistic strategy (p-strategy) is a pair of functions: $i = < c, d >$, such that:*

1. $c : \mathcal{C}[0,1] \times \mathcal{C}[0,1] \longrightarrow \mathcal{C}[0,1]$ *is called a probabilistic composition function satisfying the following axioms:*
   (a) Commutativity : $c([a_1,b_1],[a_2,b_2]) = c([a_2,b_2],[a_1,b_1])$
   (b) Associativity : $c(c([a_1,b_1],[a_2,b_2]),[a_3,b_3]) = c([a_1,b_1],c([a_2,b_2],[a_3,b_3]))$
   (c) Monotonicity : $c([a_1,b_1],[a_2,b_2]) \subseteq c([a_3,b_3],[a_2,b_2])$ if $[a_1,b_1] \subseteq [a_3,b_3]$
2. $d : \mathcal{C}[0,1] \longrightarrow 2^{\mathcal{C}[0,1]\times\mathcal{C}[0,1]}$ *is called a probabilistic decomposition function.*

The function c above is a *composition* function that generates a new interval from two input intervals. In contrast, the *decomposition* function d takes an interval, as input, and returns as output, a set of pairs of intervals. For now, there is no "connection" that ties c and d together: this will be made later through the concept of *coherence* (Definition 3). P-strategies are of two types, depending upon whether they satisfy certain extra axioms.

Definition 2 Conjunctive and Disjunctive p-strategies

- A *p-strategy* $< c, d >$ *is called* a conjunctive p-strategy if it satisfies the following axioms:
  1. Bottomline : *It is always the case that* $c([a_1,b_1],[a_2,b_2]) \leq_t [min(a_1,a_2), min(b_1,b_2)]$
  2. Identity : $c([a,b],[1,1]) = [a,b]$
  3. Annihilator :$c([a,b],[0,0]) = [0,0]$
- A *p-strategy* $< c, d >$ *is called* a disjunctive p-strategy if c satisfies the following axioms:
  1. Bottomline : $[max(a_1,a_2), max(b_1,b_2)] \leq_t c([a_1,b_1],[a_2,b_2])$
  2. Identity : $c([a,b],[0,0]) = [a,b]$
  3. Annihilator : $c([a,b],[1,1]) = [1,1]$

Intuitively, a composition function determines, given the probability ranges of two events, the probability range of their (either and- or or-composition). A decomposition function may be thought of as the inverse of composition: given the probability range of the result (and/or-composition of two events) it returns the set of all possible pairs of initial probabilistic ranges for the two events. To ensure that this holds we need the following definition:

Definition 3 A *p-strategy* $< c, d >$ *is called* coherent if

$$(\forall [a,b] \in \mathcal{C}[0,1])(([a_1,b_1],[a_2,b_2]) \in d([a,b]) \ \textbf{iff} \ c([a_1,b_1],[a_2,b_2]) = [a,b].$$

Throughout the rest of this paper, we will use the expression p-strategy to refer to coherent p-strategies, i.e. only coherent p-strategies will be considered. Before investigating the properties of p-strategies, we present some simple examples below.

2.1 Examples of P-strategies

In this section, we will present examples of various probabilistic assumptions that have been used extensively in reasoning with uncertainty. In particular, we show how the definition of a p-strategy is rich enough to capture these assumptions.

2.2 Independence

The strategy of independence may be described as the conjunctive p-strategy $inc =< c_{inc}, d_{inc} >$ and the disjunctive p-strategy $ind =< c_{ind}, d_{ind} >$, where:

- The conjunctive p-strategy $inc =< c_{inc}, d_{inc} >$ is given by:

$$\begin{aligned} c_{inc}([a_1, b_1], [a_2, b_2]) &= [a_1a_2, b_1b_2]. \\ d_{inc}(a, b) &= \{< [a_1, b_1], [a_2, b_2] > |(a_1a_2 = a \textbf{ and } b_1b_2 = b)\}. \end{aligned}$$

- The disjunctive p-strategy $ind =< c_{ind}, d_{ind} >$ is given by:

$$c_{ind}([a_1, b_1], [a_2, b_2]) = [\min(1, a_1 + a_2 - a_1a_2), \min(1, b_1 + b_2 - b_1b_2)]$$

$$d_{ind}([a, b]) \textsf{ contains } < [a_1, b_1], [a_2, b_2] >\in \mathcal{C}[0, 1] \times \mathcal{C}[0, 1]$$

iff

$$\textbf{if } a = 1 \textbf{ then } a_1 + a_2 - a_1a_2 \geq 1$$

$$\textbf{if } a < 1 \textbf{ then } a_1 + a_2 - a_1a_2 = a$$

$$\textbf{if } b = 1 \textbf{ then } b_1 + b_2 - b_1b_2 \geq 1$$

$$\textbf{if } b = 1 \textbf{ then } b_1 + b_2 - b_1b_2 = b$$

2.3 Ignorance

When nothing is known about the relationship between the events we are forced to use p-strategies that reflect *ignorance*[22]. $igc =< c_{igc}, d_{igc} >$ below is a *conjunctive* ignorance strategy, while $igd =< c_{igd}, d_{igd} >$ is a disjunctive ignorance strategy.

- Conjunctive ignorance p-strategy
  $igc =< c_{igc}, d_{igc} >$, where

$$c_{igc}([a_1, b_1], [a_2, b_2]) = [\max(0, a_1 + a_2 - 1), \min(b_1, b_2)]$$

$$d_{igc}([a, b]) \textsf{ contains } < [a_1, b_1], [a_2, b_2] >$$

iff

$$\textbf{if } a = 0 \textbf{ then } a_1 + a_2 \leq 1$$

$$\textbf{if } a > 0 \textbf{ then } a_1 + a_2 - 1 = a$$

$$(b = b_1 \textbf{ and } b_2 \geq b_1) \textbf{ or } (b = b_2 \textbf{ and } b_1 \geq b_2)$$

- Disjunctive ignorance p-strategy
  $igd =< c_{igd}, d_{igd} >$, where

$$c_{igd}([a_1, b_1], [a_2, b_2]) = [\max(a_1, a_2), \min(1, b_1 + b_2)]$$

$$d_{igd}([a, b]) \textsf{ contains } < [a_1, b_1], [a_2, b_2] >$$

iff

$$(a = a_1 \textbf{ and } a_2 \leq a_1) \textbf{ or } (a = a_2 \textbf{ and } a_1 \leq a_2)$$

$$\textbf{if } b = 1 \textbf{ then } b_1 + b_2 \geq 1$$

$$\textbf{if } b < 1 \textbf{ then } b_1 + b_2 = b$$

2.4 Positive Correlation

Sometimes we know that the fact that event e_1 has happened implies that some event e_2 also had to happen (f.e., one would assume that *"Jon rides a bus"* would imply *"Jon bought a ticket"*). Below are conjunctive and disjunctive strategies for thies case.

- Congunctive p-sttrategy
  $pcc =< c_{pcc}, d_{pcc} >$, where

$$c_{pcc}([a_1, b_1], [a_2, b_2]) = [\min(a_1, a_2), \min(b_1, b_2)]$$

$$d_{pcc}([a, b]) = \{< [a_1, b_1], [a_2, b_2] >\}$$

  iff

$$(a = a_1 \textbf{ and } a_2 \geq a_1) \textbf{ or } (a = a_2 \textbf{ and } a_1 \geq a_2)$$

  and

$$(b = b_1 \textbf{ and } b_2 \geq b_1) \textbf{ or } (b = b_2 \textbf{ and } b_1 \geq b_2)$$

- Disjunctive p-strategy
  $pcd =< c_{pcd}, d_{pcd} >$, where

$$c_{pcc}([a_1, b_1], [a_2, b_2]) = [\max(a_1, a_2), \max(b_1, b_2)]$$

$$d_{pcc}([a, b]) = \{< [a_1, b_1], [a_2, b_2] >\}$$

  iff

$$(a = a_1 \textbf{ and } a_2 \leq a_1) \textbf{ or } (a = a_2 \textbf{ and } a_1 \leq a_2)$$

  and

$$(b = b_1 \textbf{ and } b_2 \leq b_1) \textbf{ or } (b = b_2 \textbf{ and } b_1 \leq b_2)$$

2.5 Negative correlation

Sometimes, the fact that event e_1 took place means that event e_2 could not possibly happen. F.e., if "Jon came by bus" did happen, then "Jon came by train" did not. In this case we know that both events could not possibly happen together, therefore there is no conjunction p-strategy for negative correlation. However, it does make sense to ask what is the probability that one of the events took place. Below is the disjunctive p-strategy for that.

$$ncd =< c_{ncd}, d_{ncd} >$$

, where

$$c_{ncd}([a_1, b_1], [a_2, b_2]) = [\min(1, a_1 + a_2), \min(1, b_1 + b_2)]$$

$$d_{ncd}([a, b]) = \{< [a_1, b_1], [a_2, b_2] >\}$$

such that:

$$\textbf{if } a = 1 \textbf{ then } a_1 + a_2 \geq 1$$

$$\textbf{if } b = 1 \textbf{ then } b_1 + b_2 \geq 1$$

The following result, which is immediately verifiable from the definitions, asserts that the four p-strategies described here are all coherent.

Proposition 1 *inc, igc and pcc are* conjunctive coherent p-strategies. *Similarly, ind, igd, pcd and ncd are* disjunctive coherent p-strategies.

2.6 Properties of P-Strategies

In this section, we define various aspects of p-strategies that will play a key role in the definition of our fixpoint semantics and our model theory. First, we need the following

Claim 1 *Let $i =< c, d >$ be a coherent p-strategy. Then a pair $< [a_1, b_1], [a_2, b_2] >\in d([a, b])$* **iff** *$< [a_2, b_2], [a_1, b_1] >\in d([a, b])$*

Proof. By *commutativity* of composition function if $c([a_1, b_1], [a_2, b_2]) = [a, b]$ then $c([a_2, b_2], [a_1, b_1]) = [a, b]$. Since i is a *coherent* p-strategy, both $< [a_1, b_1], [a_2, b_2] >$ and $< [a_2, b_2], [a_1, b_1] >$ are in $d[a, b]$. ■

The simple claim above merely assures us that if $< [a_1, b_1], [a_2, b_2] >\in d([a, b])$, then so is $< [a_2, b_2], [a_1, b_1] >$.

Claim 2 *Let $i =< c_i, d_i >$ be a coherent disjunctive or conjunctive p-strategy. Then $c_i([0, 1], [0, 1]) = [0, 1]$. More generally, $c_i([x, 1], [y, 1]) = [z, 1]$ and $c_i([0, x], [0, y]) = [0, z]$.*

Given a pair $[a, b]$, the projection set of decomposition function d w.r.t. $[a, b]$ is the set of all $[a', b']$'s such that $[a', b']$ can be composed with some $[a'', b'']$ via the composition function c to yield $[a, b]$.

Definition 4 *Let $i =< c, d >$. The "decomposition projection set" πD is defined to be:*

$$\pi D_i([a, b]) = \{[a', b'] \in \mathcal{C}[0, 1] | (\exists [a'', b''] \in \mathcal{C}[0, 1])(< [a', b'], [a'', b''] >\in d([a, b]))\}$$

Intuitively speaking, projection functions are used as follows: suppose we know that the probability of (say) some compound event $(e_1 \wedge e_2)$ lies in the interval $[a, b]$, when $\wedge$ is computed w.r.t. some conjunctive p-strategy $i =< c, d >$. In this case, $\pi D_i([a, b])$ specifies the set of all possible probability intervals for e_1 (and likewise for e_2) that could have led to $(e_1 \wedge e_2)$'s probability interval being $[a, b]$. In other words, in order for $(e_1 \wedge e_2)$'s probability interval to be $[a, b]$, e_1's probability interval must have been an element in $\pi D_i([a, b])$, but we do not know which one.

As a consequence, e_1's probability may be as low as the smallest point in $\bigcup_{[x,y]\in \pi D_i([a,b])}[x, y]$, or as large as the largest member of $\bigcup_{[x,y]\in \pi D_i([a,b])}[x, y]$. This yields an interval for e_1's probability, and motivates the following definition.

Definition 5 *Let $i =< c, d >$ be a p-strategy. A "maximal interval" md for $d([a, b])$ is defined as*

$$md_i([a, b]) = [\min_{[a',b']\in D_i([a,b])}(a'), \max_{[a',b']\in D_i([a,b])}(b')].$$

When computing probabilities of primitive events from known probabilities of more complex events, we need to be able to compute "maximal intervals" efficiently. The following theorem gives us a constant time method to compute "maximal intervals" w.r.t. conjunctive and disjunctive p-strategies.

Theorem 1 *Suppose $i =< c, d >$* is any conjunctive coherent p-strategy *and $j =< c', d' >$* is any disjunctive coherent p-strategy*. Then:*

1. *$(\forall [a, b] \in \mathcal{C}[0, 1])(md_i[a, b] = [a, 1])$.*
2. *$(\forall [a, b] \in \mathcal{C}[0, 1])(md_j[a, b] = [0, b])$.*

Proof.

1. Let $md_i([a,b]) = [a',b']$. Since i is conjunctive strategy, $c_i([a,b],[1,1]) = [a,b]$ (Identity), and since i is coherent, $[1,1] \in \pi D_i([a,b])$. Since $b' = \max_{[\hat{a},\hat{b}] \in \pi D_i([a,b])}(\hat{b})$, and $[1,1] \in \pi D_i([a,b])$, $b' = 1$.

   Since $c_i([a,b],[1,1]) = [a,b]$ and i is coherent, $[a,b] \in \pi D_i([a,b])$. By the bottomline axiom, $(\forall [\hat{a},\hat{b}] \in \pi D_i([a,b]))(a \leq \hat{a})$. Since $[a,b] \in \pi D_i([a,b])$ $a = \min_{[\hat{a},\hat{b}] \in \pi D_i([a,b])}(\hat{a})$, and therefore, $a' = a$. ■

2. Let $md_i([a,b]) = [a',b']$. Since i is disjunctive strategy, $c_i([a,b],[0,0]) = [a,b]$ (Identity), and since i is coherent, $[0,0] \in \pi D_i([a,b])$. Therefore, since $a' = \min_{[\hat{a},\hat{b}] \in \pi D_i([a,b])}(\hat{a})$, and $[0,0] \in \pi D_i([a,b])$, $a' = 0$.

   Since $c_i([a,b],[0,0]) = [a,b]$ and i is coherent, $[a,b] \in \pi D_i([a,b])$. By the bottomline axiom, $(\forall [\hat{a},\hat{b}] \in \pi D_i([a,b]))(b \geq \hat{b})$. Since $[a,b] \in \pi D_i([a,b])$ $b = \max_{[\hat{a},\hat{b}] \in \pi D_i([a,b])}(\hat{b})$, and therefore, $b' = a$. ■

3 Syntax of hp-programs

In hybrid probabilistic programs, we assume the existence of an arbitrary, but fixed set of conjunctive and disjunctive p-strategies. The programmer may augment this set with new strategies when s/he needs new ones for their application. The following definition says that each strategy has an associatated conjunction operator, and a disjunction operator.

Definition 6 *Let $\mathcal{CONJ}$ be a finite set of conjunctive p-strategies and $\mathcal{DISJ}$ be a finite set of disjunctive p-strategies. Let $\mathcal{S}$ denote $\mathcal{CONJ} \cup \mathcal{DISJ}$.*

- Let $i \in \mathcal{CONJ}$. *Connective $\wedge_i$ is called an* i-annotated conjunction
- Let $i \in \mathcal{DISJ}$. *Connective $\vee_i$ is called an* i-annotated disjunction

Hybrid basic formulas, defined below, are either conjunctions of atoms, or disjunctions of atoms (but not mixes of both) w.r.t. a single connective.

Definition 7 *Let i be a* p-strategy*, and $A_1, \ldots, A_k$ be atoms. Then*

$$A_1 \wedge_i A_2 \ldots \wedge_i A_k$$

and

$$A_1 \vee_i A_2 \ldots \vee_i A_k$$

are called hybrid basic formulas*. Suppose $bf_i(B_L)$ denotes the set of all ground hybrid basic formulas for the $\vee_i$ and $\wedge_i$ connectives. Let $bf_{\mathcal{S}}(B_L) = \cup_{i \in \mathcal{S}} bf_i(B_L)$. Similarly, $bf_{\mathcal{CONJ}} = \cup_{i \in \mathcal{CONJ}} bf_i(B_L)$ and $bf_{\mathcal{DISJ}} = \cup_{i \in \mathcal{DISJ}} bf_i(B_L)$.*

For instance, returning to our stock example, the formulas (ch-sells-stock(C) $\vee_{ig}$ ch-retires(C)) and (price-drop(C) $\wedge_{in}$ stable(C)) are basic formulas involving the ignorance and independence p-strategies. In order to proceed further we have to define a notion of *annotaion*. Definitions 8– 10 below were introduced in [23].

Let L be a language generated by finitely many constant and predicate symbols. We assume that L has no ordinary function symbols, but it may contain *annotation function* symbols for a fixed family of functions interpreted as follows:

Definition 8 An *annotation function* f of arity n is a total function $f : [0,1]^n \longrightarrow [0,1]$.

Let $\mathcal{F}^n[0,1]$ denote the set of all annotation functions of arity n and let $\mathcal{F}[0,1]$ denote $\cup_{n=0}^{\infty}\mathcal{F}^n[0,1]$.

We assume that each *annotation function* is computable, in the sense that for any annotation function $f \in \mathcal{F}^n[0,1]$ there exists a fixed procedure that given valid input computes the value of f on this input in a finite amount of time.

We also assume that all variable symbols from L are partitioned into two classes. We will call one class *object variable symbols* and this class will contain the regular first order logic variable symbols. Second class of variable symbols, *annotation variables* will contain variable symbols that can range over the interval $[0,1]$. These variables can apprear only inside *annotation items*, which are defined below:

Definition 9 An *annotation item* ρ is one of the following:

- *a constant in the* $[0,1]$ *interval*
- *an annotation variable symbol from* L
- *let* f *be an annotation function symbol from* L *of arity* n *and let* $\delta_1,\ldots,\delta_n$ *be annotation items. Then* $f(\delta_1,\ldots,\delta_n)$ *is also an annotation item.*

Definition 10 *Let* ρ_1 *and* ρ_2 *be annotation terms. Closed interval* $[\rho_1,\rho_2]$ *is called an annotation or an annotation term.*

Following the terminology introduced in [23] if an annotation term has no annotation variables in it, we call it a *c-annotation*. Otherwise it will be called a *v-annotation*.

Example 1 $[0,1]$ *and* $[0.3,0.6]$ *are* c-annotations. $[V_1,1]$ *and* $[0.5 * V_1, V_1]$ *are* v-annotations.

Let B_L denote the Herbrand base of L. Since L contains no first-order logic function symbols, B_L is finite.

Definition 11 *A* hybrid probabilistic annotated basic formula *(hp-annotated basic formula) is an expression of the form* $B : \mu$ *where* B *is a hybrid basic formula and* μ *is an annotation .*

Informally speaking, $B : \mu$ may be read as "The probability of B occuring lies in the interval μ." For example, the annotated basic formula (ch-sells-stock(C) $\vee_{ig}$ ch-retires(C)):$[0.4,0.9]$ may be read as: "The probability that the chairman sells stock or the chairman retires lies in the 40–90% interval, assuming (no knowledge) ignorance of the relationship between these two primitive events." Hybrid rules may now be constructed from hybrid annotated formulas as follows.

Definition 12 *Let* $B_0, B_1, \ldots, B_k$ *be hybrid basic formulas. Let* $\mu_0, \mu_1, \ldots, \mu_k$ *be annotations, such that every annotation variable (if any) occurring in* μ_0 *also occurs in at least one of* $\mu_1, \ldots, \mu_k$ *. A* hybrid probabilistic clause *(hp-clause) is a construction of the form:*

$$B_0 : \mu_0 \longleftarrow B_1 : \mu_1 \wedge \ldots \wedge B_k : \mu_k$$

Informally speaking, the above rule is read: "If the probability of B_1 falls in the interval μ_1 and $\cdots$ the probability of B_k falls within the interval μ_k, then the probability of B_0 falls within the interval μ_0.

Notice that the definition above contains a requirement that every annotation variable that appears in the annotation for the head of the clause also appears in one or more annotations for the body of the hp-clause. Therefore,

Example 2 • *$A : [V_1, V_1] \longleftarrow$ is <u>not</u> an hp-clause.*

- *$A : [V_1, V_2] \longleftarrow (B \wedge_{ind} C) : [0, V_1] \wedge D : [V_2, 1]$ is an hp-clause.*

Definition 13 *A hybrid probabilistic program (hp-program) over set $\mathcal{S}$ of p-strategies is a finite set of hp-clauses involving only connectives from $\mathcal{S}$.*

For example, the following four clauses constitute a simple hp-program using the p-strategies of ignorance and independence.

```
price-drop(C):[0.4, 0.9] ⟵ (ch-sells-stock(C) ∨_igd ch-retires(C)):[0.6, 1].
price-drop(C):[0.5, 1] ⟵ (strike(C) ∨_ind accident(C)):[0.3, 1].
buy-stock(C):[0.7, 1] ⟵ (price-drop(C)  ∧_inc stable(C)):[0.3, 1].
sell-stock(C):[0.5, 1] ⟵ (price-drop(C) ∧_inc unstable(C)):[0.4, 1] ∧ have-stock(C):[1, 1].
stable(c):[0.8, 1] ⟵.
strike(c):[0.4, 0.5] ⟵.
unnstable(C):[V1, V2] ⟵ stable(C):[1 − V2, 1 − V1].
```

The program above is a very simple example of a market decision making program. The first two rules tell us when to expect that the stock of company C will drop. According to the first rule, it will drop with probability between 40% and 90% if the probability that CEO of the company will sell the stock or that he will retire is more than 60%. We use *ignorance* assumption here, because we do not know if there is any connection between the two events. In fact, for different companies the *correlation* may range from the two being independent, to one being a consequence of the other. The *Ignorance* assumption here gives us a certain "lowest common denominator" in terms of relationship between the two events.

The second rule states that if the probability that the company's employees will go on strike or that an accident happens on premises of the company is over 30%, then the probability that the stock of the company will drop is *at least* 40%. It is more or less safe to assume that the causes for strikes and for accidents to occur are completely different, therefore, the two events are independent of each other.

The next two rules deal with decision-making. The third rule of the program, says that we should buy stock of company C if its price drops, but (and) the compnay is generally known to be stable. We want to assume that our knowledge of the stability of company C is independent of the price drop under consideration, therefore, the conjunction of the two events is made under the assumption of independence. The fourth rule provides an alternative to the third by declaring that if the price drops and there is a high probability that the company is unstable, the stock has to be sold. For this rule to fire, however, we need one more condition: one can sell stock of company C only if one owns this stock. This is why we must know for sure (i.e. with probability 100%) that we own this stock if we want to sell it.

Two facts that follow describe our current knowledge of situation, expressed probabilistically. The first fact states that company c is stable with probability more than 80%. The second fact states that the probability of a strike for this company is between 40% and 50%.

Finally the last rule can be used to establish the connection between the information about the stability of company C and its nonstability. Indeed, if we assume that each company is either stable or unstable (a reasonable assumption for our example), then, if we know that the probability that company C is stable is p, the probability that C is unstable (i.e., *not stable*) than would have to be $1-p$. We exend this simple observation to the notion of probabilistic intervals to obtain that if C is stable with probability between $V1$ and $V2$ then it is unstable with probability between $1 - V2$ and $1 - V1$.

4 Declarative Semantics of hp-programs

Having completed the definition of the syntax of hp-programs, we are now in a position to develop the declarative semantics of such programs. We will first develop a fixpoint semantics of hp-programs, followed by a model theoretic semantics, and show that the two are essentially equivalent characterizations of hp-programs. Later, in Section 5, we will provide a proof procedure for hp-programs.

4.1 Fixpoint Semantics

As usual, suppose we have a logical language L consisting of variable symbols, constant symbols, function symbols, and predicate symbols, and that B_L denotes the Herbrand base of this language. An atomic function, defined below, merely assigns closed intervals to ground atoms.

Definition 14 *A function* $f : B_L \longrightarrow \mathcal{C}[0,1]$ *is called an* atomic function .

Intuitively, if $f(A) = \mu$, then this means that according to the atomic function f, the probability of the ground atom A lies in the interval μ. We may impose on ordering, $\leq$, on the set of atomic functions as follows:

$$f_1 \leq f_2 \textbf{ iff } (\forall A \in B_L) f_1(A) \supseteq f_2(A).$$

Intuitively, as one "climbs up" the $\leq$-ordering, the atomic functions involved assign "sharper" probability ranges to ground atoms. Though atomic functions do not, by themselves, make assignments to basic formulas, they may be extended to do so,

Definition 15 Let f be *an atomic function.* Then the function
$h_f : bf_{\mathcal{S}}(B_L) \longrightarrow \mathcal{C}[0,1]$, is defined as follows:

1. $h_f(F) = f(F)$, if $F \in B_L$
2. $h_f(F_1 \wedge_i F_2) = c_i(h_f(F_1), h_f(F_2))$ where $i =< c_i, d_i > \in \mathcal{CONJ}$
3. $h_f(F_1 \vee_i F_2) = c_i(h_f(F_1), h_f(F_2))$ where $i =< c_i, d_i > \in \mathcal{DISJ}$

is called a hybrid formula function (based on the atomic function f).

Suppose $\mathcal{HFF}$ denotes the set of all hybrid formula functions. The $\leq$-ordering on atomic functions may be extended to basic formulas in the obvious way: $h_1 \leq h_2$ iff $(\forall F \in bf_{\mathcal{S}}(B_L)) h_1(F) \supseteq h_2(F)$.

Lemma 1 1. *If* f, g are atomic functions such that $f \leq g$, *then* $h_f \leq h_g$.

2. $< \mathcal{HFF}, \leq >$ *is a complete lattice.*

Proof

1. *Induction on the structure of formula* F.
   - *Base Case.* F is atom. Then by definition:
     $h_f(F) = f(F) \supseteq g(F) = h_g(F)$ since $f \leq g$.
   - *Induction Step* Let $h_f(F_1) \supseteq h_g(F_1)$ and $h_f(F_2) \supseteq h_g(F_2)$. Let $*_i$ denote any conncetive of a form $\wedge_i, i \in \mathcal{CONJ}$ or $\vee_i, i \in \mathcal{DISJ}$. Let also $(F_1 *_i F_2) \in bf_{\mathcal{S}}(B_L)$.
     In this case, by def. $h_f(F_1 *_i F_2) = c_i(h_f(F_1), h_f(F_2))$ and $h_g(F_1 *_i F_2) = c_i(h_g(F_1), h_g(F_2))$.
     By monotonicity property of composition function,

$$h_f(F_1 *_i F_2) = c_i(h_f(F_1), h_f(F_2)) \supseteq c_i(h_f(F_1), h_g(F_2)) \supseteq c_i(h_g(F_1), h_g(F_2)) = h_g(F_1 *_i F_2)$$

■

2. Let $\mathcal{H} \subseteq \mathcal{HFF}$. We define *least upper bound* of $\mathcal{H}$ to be:
   $\bigsqcup(\mathcal{H})(F) = \cap\{\mu | h(F) = \mu, h \in \mathcal{H}\}$ and *greatest lower bound* of $\mathcal{H}$ as:
   $\sqcap(\mathcal{H})(F) = closure(\cup\{\mu | h(F) = \mu, h \in \mathcal{H}\})$.

   The top element of $\mathcal{HFF}$ is the function $\top$ such that $\forall F \in bf_{\mathcal{S}}(B_L)$ $h(F) = \emptyset$. The bottom element is the function $\perp$ such that $\forall F \in bf_{\mathcal{S}}(B_L)$ $h(F) = [0,1]$. ■

Given any hp-program P, we wish to associate with P, an operator T_P that maps hybrid formula functions to hybrid formula functions. We do this by first defining a (similar) intermediate operator S_P that is used subsequently to define T_P.

Definition 16 *Let P be* a hybrid probabilistic program. *Operator $S_P : \mathcal{HFF} \longrightarrow \mathcal{HFF}$ is defined as follows (where F is a basic formula):*

$$S_P(h)(F) = \cap M \textit{ where}$$

$$M = \{\mu\sigma | F : \mu \longleftarrow F_1 : \mu_1 \wedge \ldots \wedge F_n : \mu_n$$

is a ground instance of some clause in P ;*σ is a ground substitution of annotation variables and* $(\forall j \leq n) h(F_j) \subseteq \mu_j\sigma\}$ *if $M = \emptyset$ $S_P(h)(F) = [0,1]$.*

The operator S_P is very simple. Given $h \in \mathcal{HFF}$ and a basic formula F, it proceeds as follows: (i) First, it finds all ground instances of rules in P such that the head of the rule instance is of the form $F : \mu$ and such that for each $F_i : \mu_i$ in the body, $h(F_i) \subseteq \mu_i$, i.e. h says that F_i's probability does in fact lie within the interval μ_i. (ii) It then takes the intersection of the intervals associated with the heads of all rules identified in the preceding step. Note that in the above definition, it is entirely possible that $S_P(h)(F)$ could be the empty set. In this case, there is an intuitive inconsistency, because the formula function $S_P(h)$ is saying that F's probability lies in the empty set. However, this is absurd, as the empty set cannot contain anything. This will be discussed in further detail in Section 4.2.

Example 3 *Consider our stock example.* Let h assign the following values to the atoms:

h(ch-sells-stock(c)) $= [0.8, 0.8]$

h(ch-retires(c)) $= [0.1, 0.1]$

h(strike(c)) $= [0.4, 0.5]$

h(price-drop(c)) $= [0.7, 0.9]$

h(stable(c)) $= [0.5, 0.6]$

Assume that for all other ground atoms A, $h(A) = [0,1]$.

Now, suppose we want to compute $S_P(h)$(price-drop(c)). There are two ground rule instances with price-drop(c) as their head in the set of all groundizations of rules in P:

price-drop(c):$[0.4, 0.9] \longleftarrow$ (ch-sells-stock(c) $\vee_{igd}$ ch-retires(c)):$[0.6, 1]$.

price-drop(c):$[0.5, 1] \longleftarrow$ (strike(c) $\vee_{ind}$ accident(c)):$[0.3, 1]$.

First we compute

- h((strike(c) $\vee_{ind}$ accident(c)) $= c_{ind}(h$(strike(c)),accident(c)) $= c_{ind}([0.4, 0.5], [0, 1]) = [\min(1, 0.4 + 0 - 0.4 * 0), \min(1, 0.5 + 1 - 0.5 * 1)] = [0.4, 1] \subseteq [0.3, 1]$..

- $h((\text{ch-sells-stock(c)} \vee_{igd} \text{ch-retires(c)})) = c_{igd}(\text{ch-sells-stock(c)},\text{ch-retires(c)}) = c_{igd}([0.8, 0.8], [0.1, 0.1]) = [max(0.8, 0.1), min(1, 0.8 + 0.9)] = [0.8, 0.9] \subseteq [0.6, 1]$.

Since both rules will fire, $M = \{[0.4, 0.9], [0.5, 1]\}$ and therefore, $S_P(h)(\text{price-drop(c)}) = [0.4, 0.9] \cap [0.5, 1] = [0.5, 0.9]$.

However, the S_P operator is not quite "right." The reason is that in order to determine F's probability, it is not enough to merely look for rule instances whose head is identical to F. For instance, F might be $(p \wedge_{ig} q)$. The probability of $(p \wedge_{ig} q)$ may certainly be influenced by rules with head $p : \mu'$ because such rules may impose lower bounds on p's probability – and hence on $(p \wedge_{ig} q)$'s probability. Thus, S_P, by itself, does not allow us to accurately infer the probability associated with a formula F. S_P needs to be *augmented appropriately* in order to do so. However, before defining T_P, we present a simple monotonicity property of S_P. *Note that S_P is monotonic regardless of what p-strategies appear in P.*

Lemma 2 S_P is monotonic, i.e., if h_1, h_2 are two formula functions and $h_1 \leq h_2$ then, $S_P(h_1) \leq S_P(h_2)$.

Proof.

Let F be a hybrid basic formula. We have $h_1(F) \leq h_2(F)$. By def. of S_P,

$$S_P(h_1)(F) = \cap M_1$$

$$M_1 = \{\mu | F : \mu \longleftarrow F_1 : \mu_1 \wedge \ldots \wedge F_n : \mu_n$$
$$\text{is a ground instance of some clause in } P \text{ ; } (\forall j \leq n) h_1(F_j) \subseteq \mu_j\}$$

Since $h_1(F_j) \subseteq \mu_j$ can be rewritten as $\mu_j \leq h_1(F_j)$, using transitivity of $\leq$, we obtain that for any ground instance $F : \mu \longleftarrow F_1 : \mu_1 \wedge \ldots \wedge F_n : \mu_n$ of a rule of program P, such that $\mu \in M_1$, $\mu \in M_2$, where

$$M_2 = \{\mu | F : \mu \longleftarrow F_1 : \mu_1 \wedge \ldots \wedge F_n : \mu_n$$
$$\text{is a ground instance of some clause in } P \text{ ; } (\forall j \leq n) h_2(F_j) \subseteq \mu_j\}$$

and therefore, $M_1 \subseteq M_2$. Therefore, $S_P(h_2)(F) = \cap M_1 = (\cap M_1) \cap (M_2 - M_1) = S_P(h_1)(F) \cap (M_2 - M_1) \subseteq S_P(h_1)(F)$ i.e., $S_P(h_1)(F) \leq S_P(h_2)(F)$. ■

Let us now define the T_P operator.

Definition 17 *Let $F = F_1 *_i \ldots *_i F_n$, $G = G_1 *_i \ldots *_i G_k$, $H = H_1 *_i \ldots *_i H_m$ where $* \in \{\wedge, \vee\}$. We will write $G \oplus H = F$* **iff:**

1. $\{G_1, \ldots, G_k\} \subset \{F_1, \ldots, F_n\}$ *and*
   $\{H_1, \ldots, H_m\} \subset \{F_1, \ldots, F_n\}$ *and*
2. $\{G_1, \ldots, G_k\} \cup \{H_1, \ldots, H_m\} = \{F_1, \ldots, F_n\}$ *and*
3. $\{G_1, \ldots, G_k\} \cap \{H_1, \ldots, H_m\} = \emptyset$ *and*
4. $G \neq F$, $H \neq F$.

The definition above provides a convenient notation for expressing the fact that formula F can be partitioned into subformulas G and H *exactly*.

Definition 18 *Let P be* a hybrid probabilistic program. We inductively define operator $T_P : \mathcal{HFF} \longrightarrow \mathcal{HFF}$ as follows:

1. *Let F be an atomic formula.*
   - *if $S_P(h)(F) = \emptyset$ then $T_P(h)(F) = \emptyset$.*
   - *if $S_P(h)(F) \neq \emptyset$, then let*

     $$M = \{\langle \mu\sigma, i\rangle | (F *_i G) : \mu \longleftarrow F_1 : \mu_1 \wedge \ldots \wedge F_n : \mu_n$$

     where $* \in \{\vee, \wedge\}$ *and σ is a ground substitution of the annotation varables and $i \in \mathcal{S}$ and $(\forall j \leq n) h(F_j) \subseteq \mu_j\sigma\}$.* We define

     $$T_P(h)(F) = (\cap\{md_i(\mu\sigma) | \langle \mu\sigma, i\rangle \in M\}) \cap S_P(h)(F)$$

2. *(F not atomic) Let $F = F_1 *_i \ldots *_i F_n$.*
   Let $M' = \{\langle \mu\sigma, i\rangle | D_1 *_i \ldots *_i D_k : \mu \longleftarrow E_1 : \mu_1 \wedge \ldots E_m : \mu_m \in ground(P)$; $(\forall 1 \leq j \leq m) h(E_j) \subseteq \mu_j$; $\{F_1, \ldots, F_n\} \subset \{D_1, \ldots D_k\}, n < k\}$
   Then:

   $$T_P(h)(F) = S_P(h)(F) \cap (\cap\{c_i(T_P(h)(G), T_P(h)(H)) | G \oplus H = F\}) \cap (\cap\{md_i(\mu\sigma) | \langle \mu\sigma, i\rangle \in M'\})$$

The T_P operator may be justified as follows: (i) Consider an atomic formula A: if $S_P(h)(F) = \emptyset$, then this means that an inconsistency (to be made more formal in Section 4.2) has occurred. For instance, if we have an hp-program containing two facts $a : [0, 0]$ and $a : [1, 1]$, then whatever h we pick, $S_P(h)(a) = \emptyset$, reflecting the (in this case flagrant) inconsistency in P. Thus, $T_P(h)$ must also assign $\emptyset$ to F. If $S_P(h)(F) \neq \emptyset$, then it may be case that $S_P(h)$ has assigned too "wide" an interval to F, because it ignores rules that are "associated" with F. As F is atomic, there might be rules whose bodies are satisfied by h, which include F in its head. We must find all such rules, and "split" the rule head into its F part, and the non-F part, say G. Clearly, the rule head must be of the form $(F *_i G)$ where $*$ is either $\wedge$ or $\vee$. As the rule's body is satisfied by h, it means that the head of this rule, viz. $(F *_i G)$ has probability in the interval μ. The rule in question thus allows us to conclude that F's probability ranges anywhere in $md(\mu)$ which is the "maximal interval" associated with F w.r.t. the connection $*_i$. We repeat this for each rule with F as part of the head.

(ii) When F is not a ground atom, there can be three sources of bounds on F's probability interval. First source, taken care by S_P operator are the rules with F as their head. Second source is information that can be inductively obtained by computing T_P for every pair G, H of formulas such that $G \oplus H = F$ (notice that we require both G and H to be non-empty), and using c_i to combine these values. Finally, some rules of the program may contain F as the proper subset. The probability range of F from each of such rules is determined by the md_i function. Combining (intersecting) the ranges obtained from all three sources we obtain the final value of T_P operator.

The following example demonstrates how T_P is computed.

Example 4 *Let us consider the stock program P and the formula function h from the previous example.* Suppose we want to compute $T_P(h)$(price-drop(c)$\wedge_{pcc}$buy-stock(c)) (i.e., the probability of the fact that the drop in price of stocks will result in purchases of new stock of company c).

It is easy to see that $T_P(h)$(price-drop(c)$\wedge_{pcc}$buy-stock(c)= $c_{pcc}(T_P(h)$(price-drop(c)), $T_P(h)$(buy-stock(c)), as the heads of all rules in P are atomic.

$T_P(h)$(price-drop(c)) = $S_P(h)$(price-drop(c)) = $[0.5, 0.9]$ (see **Example 3**). $T_P(h)$(buy-stock(c)) = $S_P(h)$(buy-stock(c)). To find the latter we consider the following ground rule in P :

buy-stock(c):$[0.7, 1] \longleftarrow$ (price-drop(c) $\wedge_{inc}$ stable(c)):$[0.3, 1]$.

Recall from **Example 3** that h(price-drop(c)) = $[0.7, 0.9]$ and h(stable(c)) = $[0.5, 0.6]$. Then, h((price-drop(c) $\wedge_{inc}$ stable(c))) = $c_{inc}(h$(price-drop(c)), h(stable(c))) = $c_{inc}([0.7, 0.9], [0.5, 0.6]) = [0.7 * 0.5, 0.9 * 0.6] = [0.35, 0.54] \subseteq [0.3, 1]$, which entails that $S_P(h)$(buy-stock(c))) = $[0.7, 1]$

Finally, $T_P(h)$(price-drop(c)$\wedge_{pcc}$buy-stock(c)= $c_{pcc}(T_P(h)$(price-drop(c)), $T_P(h)$(buy-stock(c)) = $c_{pcc}([0.5, 0.9], [0.7, 1]) = [\min(0.5, 0.7), \min(0.9, 1)] = [0.5, 0.7]$.■

It follows immediately from the definition of the T_P operator that, for any program P, formula function h and formula F, $T_P(h)(F) \subseteq S_P(h)(F)$. The following result says that regardless of which p-strategies are considered in P, the T_P operator is guaranteed to be monotonic.

Theorem 2 T_P is monotonic,i.e., if h_1, h_2 are two formula functions and $h_1 \leq h_2$ then, $T_P(h_1) \leq T_P(h_2)$.

Proof.

Let F be a hybrid basic formula. We proceed by induction on $rank(F)$.

- F is an atomic formula. We have $h_1(F) \leq h_2(F)$. Let us assume that both $S_P(h_1)(F)$ and $S_P(h_2)(F)$ are non-empty. (Otherwise, we must have $S_P(h_2)(F) = \emptyset$ which implies $T_P(h_2)(F) = \emptyset$ and therefore, it must be the case that $T_P(h_2)(F) \subseteq T_P(h_1)(F)$). By lemma $S_P(h_1)(F) \leq S_P(h_2)(F)$. Let us consider

$$M_1 = \{\mu | (F *_i G) : \mu \longleftarrow F_1 : \mu_1 \wedge \ldots \wedge F_n : \mu_n$$
$$\text{where } * \in \{\vee, \wedge\} \text{ and } i \in \mathcal{S} \text{ and } (\forall j \leq n) h_1(F_j) \subseteq \mu_j\}$$

  Since $h_1(F_j) \subseteq \mu_j$ can be rewritten as $\mu_j \leq h_1(F_j)$, using transitivity of $\leq$, we obtain that for any ground instance $F : \mu \longleftarrow F_1 : \mu_1 \wedge \ldots \wedge F_n : \mu_n$ of a rule of program P, such that $\mu \in M_1$, $\mu \in M_2$, where

$$M_2 = \{\mu | (F *_i G) : \mu \longleftarrow F_1 : \mu_1 \wedge \ldots \wedge F_n : \mu_n$$
$$\text{is a ground instance of some clause in } P \,; (\forall j \leq n) h_2(F_j) \subseteq \mu_j\}$$

  Therefore, $M_1 \subseteq M_2$. But this means that $M_1' = \{md(\mu) | \mu \in M_1\} \subseteq M_2' = \{md(\mu) | \mu \in M_2\}$. Then $\cap M_2' \subseteq \cap M_1'$, i.e., $\cap M_1' \leq \cap M_2'$.

  Since $T_P(h_1)(F) = S_P(h_1)(F) \cap (\cap M_1')$, $T_P(h_2)(F) = S_P(h_2)(F) \cap (\cap M_2')$, $S_P(h_1)(F) \leq S_P(h_2)(F)$ and $\cap M_1' \leq \cap M_2'$, we obtain that

$$T_P(h_1)(F) \leq T_P(h_2)(F)$$

- Let the theorem hold for all basic hybrid formulas of ranks less than k. Let $rank(F) = k$ and $F = F_1 \vee_i \ldots \vee_i F_n$ or $F = F_1 \wedge_i \ldots \wedge_i F_n$.

  ¿From Lemma 1 we know that $S_P(h_1)(F) \leq S_P(h_2)(F)$.

Let G, H be such formulas, that $G \oplus H = F$. By the induction hypothesis, (since $rank(G) < k$ and $rank(H) < k$, we have $T_P(h_1)(G) \leq T_P(h_2)(G)$ and $T_P(h_1)(H) \leq T_P(h_2)(H)$, therefore, by monotonicity axiom for p-strategies (applied twice) we have:

$$c_i(T_P(h_2)(G), T_P(h_2)(H)) \subseteq c_i(T_P(h_1)(G), T_P(h_1)(H))$$

i.e.

$$c_i(T_P(h_1)(G), T_P(h_1)(H)) \leq c_i(T_P(h_2)(G), T_P(h_2)(H))$$

From this is follows that

$$(\cap\{c_i(T_P(h_1)(G), T_P(h_1)(H)) | G \oplus H = F\}) \leq (\cap\{c_i(T_P(h_2)(G), T_P(h_2)(H)) | G \oplus H = F\})$$

Finally, let $M_1 = \{D_1 *_i \ldots *_i D_k : \mu \longleftarrow E_1 : \mu_1 \wedge \ldots E_m : \mu_m \in ground(P) | (\forall 1 \leq j \leq m) h_1(E_j) \subseteq \mu_j;\ \{F_1, \ldots, F_n\} \subset \{D_1, \ldots D_k\}, n < k\}$ and $M_2 = \{D_1 *_i \ldots *_i D_k : \mu \longleftarrow E_1 : \mu_1 \wedge \ldots E_m : \mu_m \in ground(P) | (\forall 1 \leq j \leq m) h_2(E_j) \subseteq \mu_j;\ \{F_1, \ldots, F_n\} \subset \{D_1, \ldots D_k\}, n < k\}$ Let $M_1' = \mu | D : \mu \longleftarrow Body \in M_1$ and $M_2' = \mu | D : \mu \longleftarrow Body \in M_2$

since $h_1 \leq h_2$, we can claim that if some ground instance $C \in M_1$, C also is in M_2, i.e., $M_1 \subseteq M_2$. Therefore $(\cap\{\mu | \mu \in M_2'\}) \subseteq (\cap\{\mu | \mu \in M_1'\})$, i.e., $(\cap\{\mu | \mu \in M_1'\}) \leq (\cap\{\mu | \mu \in M_2'\})$.

Combining the established results into one, using the formula for $T_P(h)(F)$ we obtain the desired $T_P(h_1)(F) \leq T_P(h_2)(F)$.

■

Again, note that the above result applies <u>*regardless*</u> *of what set of p-strategies occur in program* P. It is easy to see now that we may define the *iterations* of T_P as:

Definition 19 *1.* $T_P^0 = h_\perp$ *where* $\perp$ *is the atomic function that assigns* $[0,1]$ *to all ground atoms* A.

2. $T_P^\alpha = T_P(T_P^{\alpha-1})$ *where* α *is a successor ordinal whose predecessor is denoted by* $\alpha - 1$.

3. $T_P^\gamma = \sqcup\{T_P^\alpha | \alpha < \gamma\}$, *where* γ *is limit ordinal.*

In [7] it was established that if all clauses in P have only constant annotations then $lfp(T_P) = T_P^\omega$, where $lfp(T_P)$ is the least fixed point of T_P. This, however, turns out to not be the case when P has clauses with variable annotations. The follwoing example is from [23].

Example 5 Consider the program
$A : [0, V/2] \longleftarrow A : [0, V]$
$B : [0, 0] \longleftarrow A : [0, 0]$
Since $T_P^0(A) = [0, 1]$, after the first iteration $T_P^1(A) = [0, 0.5]$. At each subsequent iteration, we will get the interval assigned to A narrow by half. Eventually T_P^ω will assign $[0, 0]$ interval to A. Then $T_P^{\omega+1}$ will finally assign $[0, 0]$ to B.

4.2 Probabilistic Model Theory

We are now ready to define a logical model theory for hp-programs. For this purpose, hybrid basic formula functions will play the role of an "interpretation." The key inductive definition of satisfaction is given below.

Definition 20 Satisfaction. Let h be hybrid basic formula function, $F \in bf_{\mathcal{S}}(B_L)$, $\mu \in \mathcal{C}[0,1]$. We say that

- $h \models F : \mu$ **iff** $h(F) \subseteq \mu$.
- $h \models F_1 : \mu_1 \wedge \ldots \wedge F_n : \mu_n$ **iff** $(\forall\, 1 \le j \le n) h \models F_j : \mu_j$.
- $h \models F : \mu \longleftarrow F_1 : \mu_1 \wedge \ldots \wedge F_n : \mu_n$ **iff** either $h \models F : \mu$ or $h \not\models F_1 : \mu_1 \wedge \ldots \wedge F_n : \mu_n$.
- $h \models (\exists x)(F : \mu)$ **iff** $h \models F(t/x) : \mu$ *for some ground term* t.
- $h \models (\forall x)(F : \mu)$ **iff** $h \models F(t/x) : \mu$ *for every ground term* t.

A formula function h *is called* a model *of an hp-program* P *(*$h \models P$*)* **iff** $(\forall p \in P)(h \models p)$.

As usual, we say that $F : \mu$ is a *consequence* of P iff for every model h of P, it is the case that $h(F) \subseteq \mu$.

Recall, from Section 4.1, that we can have cases where a hybrid formula function, h, could assign $\emptyset$ to some formula. When $h(F) = \emptyset$, h is "saying" that F's probability lies in the empty set. This corresponds to an inconsistency because, by definition, nothing is in the empty set.

Definition 21 *Formula function* h *is called* fully defined *iff*
$\forall (F \in bf_{\mathcal{S}}(B_L))(h(F) \neq \emptyset)$.

The following important result fully ties together, the fixpoint theory associated with hp-programs, and the model theoretical characterization of hp-programs, regardless of which p-strategies occur in the hp-program being considered.

Theorem 3 *Let* P *be any hp-program. Then:*

1. h *is a model of* P *iff* $T_P(h) \le h$.
2. P *has a model iff* $lfp(T_P)$ *is fully defined.*
3. *If* $lfp(T_P)$ *is fully defined, then it is the least model of* P*, and* $F : \mu$ *is a logical consequence of* P *iff* $T_P^{\omega}(F) \subseteq \mu$.

Proof.

1. Claim 1. $T_P(h) \le h \Longrightarrow h \models P$.

   Let $F \in bf_{\mathcal{S}}(B_L)$.
   Let $P' = \{p \in ground(P) | p$ is of form $F : \mu \longleftarrow F_1 : \mu_1 \wedge \ldots \wedge F_n : \mu_n\}$.

   Two cases are possible. If $P' = \emptyset$ then P has no rules with F in the head and therefore $h \models P'$ by def.

   Let $P' \neq \emptyset$.
   Consider a rule $p' \in P'$. p' is of a form $F : \mu \longleftarrow F_1 : \mu_1 \wedge \ldots \wedge F_n : \mu_n$. Two cases are possible.

   - $(\forall\ \ 1 \le j \le n)(\mu_j \le h(F_j))$ In this case, we know that
     $h \models F_1 : \mu_1 \wedge \ldots \wedge F_n : \mu_n$. We have to show that
     $h \models F : \mu$, i.e. $h(F) \subseteq \mu$.
     By our assumption, $T_P(h)(F) \le h(F)$, i.e., $h(F) \subseteq T_P(h)(F)$. By definition of T_P and S_P operators, it is always the case that $T_P(h)(F) \subseteq S_P(h)(F)$. We now show that $S_P(h)(F) \subseteq \mu$.

By definition, $S_P(h)(F) = \cap\mathcal{F}$ where $\mathcal{F} = \{\mu | F : \mu \longleftarrow F_1 : \mu_1 \wedge \ldots \wedge F_n : \mu_n$ is a ground instance of a rule in $P; (\forall\ 1 \leq j \leq n)(\mu_j \leq h(F_j))\}$. We know that $p' : \mu \in \mathcal{F}$, therefore, $S_P(h)(F) \subseteq \mu$, which implies that $T_P(h)(F) \subseteq \mu$. Combining together we obtain: $h(F) \subseteq T_P(h)(F) \subseteq S_P(h)(F) \subseteq \mu$ which implies $h \models F : \mu$, therefore, $h \models p'$.

- $(\exists\ 1 \leq j \leq n)(h(F_j) \not\subseteq \mu_j)$ in this case $h \not\models F_j : \mu_j$, therefore, $h \not\models F_1 : \mu_1 \wedge \ldots \wedge F_n : \mu_n$, and therefore, $h \models p'$.

This proves the first claim.

Claim 2. $h \models P \Longrightarrow T_P(h) \leq h$.

Let $F \in bf_{\mathcal{S}}(B_L)$. We prove the claim by induction on $rank(F)$.

- Base Case. $rank(F) = 0$, i.e., F is atomic. Let

$$F : \mu_1 \longleftarrow \ldots$$

$$\ldots$$

$$F : \mu_k \longleftarrow \ldots$$

$$(F *_{i_1} G_1) : \nu_1 \longleftarrow \ldots$$

$$\ldots$$

$$(F *_{i_m} G_1) : \nu_m \longleftarrow \ldots$$

be the list of all rules from program P that contain F in the head, such that, h satisfies their bodies.

By definiton of T_P, $T_P(h)(F) = \mu_1 \cap \mu_2 \cap \ldots \cap \mu_k \cap md_{i_1}(\nu_1) \cap \ldots \cap md_{i_m}(\nu_m)$.

Since h satisfies all the bodies of these rules, h must also satisfy all the heads, i.e., $(\forall\ 1 \leq \jmath \leq k)(h(F) \subseteq \mu_j)$ and
$(\forall\ 1 \leq \jmath \leq l)(h(F *_{i_j} G) \subseteq \nu_j)$. From first set of inequalities we obtain: $h(F) \subseteq \mu_1 \cap \mu_2 \cap \ldots \cap \mu_k$.

From second set of inequalities: $h(F *_{i_j} G) = c_{i_j}(h(F), h(G))$ and therefore $h(F) \subseteq md_{i_j}(\nu_j)$. This leads to $h(F) \subseteq md_{i_1}(\nu_1) \cap \ldots \cap md_{i_m}(\nu_m)$, which combined with previous result gives us desired $h(F) \subseteq T_P(h)(F)$ i.e., $T_P(h)(F) \leq h(F)$.

- Induction Step. Let our claim hold for all basic formulas rank less than k. Let $rank(F) = k$ and $F = A_1 *_i \ldots *_i A_k$.

Let

$$F : \mu_1 \longleftarrow \ldots$$

$$\ldots$$

$$F : \mu_k \longleftarrow \ldots$$

be all the rules with F as the head, such that h satisfies their bodies. We must therefore, conclude that for each of these rules h satisfies its head, i.e., $h(F) \subseteq \mu_1 \cap \mu_2 \cap \ldots \cap \mu_k = S_P(h)(F)$.

Let now G and H be basic formulas such that $G \oplus H = F$. By definition, $rank(g) < k$ and $rank(H) < k$, therefore, by the induction hypothesis, $h(G) \subseteq T_P(h(G))$ and $h(H) \subseteq T_P(h(H))$. Since $G \oplus H = F$, $G *_i H \equiv F$ and therefore $h(F) = h(G *_i H) = c_i(h(G), h(H)) \subseteq$

$c_i(T_P(h)(G), T_P(h)(F)$ (the last inequality is due to monotonicity property of composition function). Therefore we conclude that

$$h(F) \subseteq (\cap\{c_i(T_P(h)(G), T_P(h)(H)) | G \oplus H = F\})$$

Let now

$$(F *_i D_1) : \nu_1 \longleftarrow \ldots$$
$$\ldots$$
$$(F *_i D_s) : \nu_s \longleftarrow \ldots$$

be all the ground instances of rules in P such that h satisfies thier bodies and F is a part of their heads. Since $h \models P$, $h \models (F *_i D_1) : \nu_1, \ldots, h \models (F *_i D_s) : \nu_s$, i.e., $(\forall 1 \leq j \leq s)(h(F *_i D_j) \subseteq \nu_j)$. But we know that $h(F *_i D_j) = c_i(h(F), h(D_j)) \subseteq \nu_j$. For this to be true it must be the case that $h(F) \subseteq md_i(\nu_j)$. Therefore, $h(F) \subseteq md_i(\nu_1) \cap \ldots \cap md_i(\nu_s)$. Combining the three inequalities together we obtain:

$$h(h) \subseteq S_P(F) \cap (\cap\{c_i(T_P(h)(G), T_P(h)(H)) | G \oplus H = F\}) \cap (md_i(\nu_1) \cap \ldots \cap md_i(\nu_s)) = T_P(h)(F)$$

which proves the theorem.■

2. Let $lfp(T_P)$ be fully defined. Since we know that $T_P(lfp(T_P)) = lfp(T_P)$, it is also the case that $T_P(lfp(T_P)) \leq lfp(T_P)$. According to *part 1* of this theorem, $lfp(T_P)$ is a model of P.

   Assume now that P has a model h. By definition of a model, h is fully defined. We know that $T_P(h) \leq h$. By construction of $lfp(T_P)$, and because of the monotonicity of T_P operator $lfp(T_P) \leq T_P(h)$. Therefore $lfp(T_P) \leq h$. This means that for all basic formulas F, $h(F) \subseteq lfp(T_P)(F)$. Since h is fully-defined, $lfp(T_P)$ **has** to be fully defined too. ■

3. *Part 3* of this theorem is a direct corollary of *Part 2* and **theorem 2**.

The second result above links consistency of P programs with the fully defined-ness property of T_P^ω. Of course, if there exists an integer i such that either S_P^i or T_P^i are not fully defined, then T_P^ω cannot be fully defined either, and hence, P would not have a model.

5 Proof Procedure

At this stage, we have provided a complete description of what is a logical consequence of an hp-program P. In this section, we develop three query processing procedures.

- The first query processing procedure (Section 5.2), termed hp-resolution, builds upon previous approaches of Ng and Subrahmanian [22] by first requiring that programs P be compiled to a new set, $CL(P)$. Queries are then processed by a process akin to linear input resolution, with the difference that clauses from $CL(P)$ may be considered input clauses. This process suffers from the major flaw that usually, construction of $CL(P)$ is prohibitively expensive.

- The second procedure (Section 5.3), termed HR_P-refutations, is more pragmatic. Rather than requiring a compilation step, when a query Q is posed, HR_P refutations allow relevant parts of the $CL(P)$ to be dynamically constructed. This has two advantages over hp-refutations. First, hp-refutations often "lose" right at the beginning as the compilation process may take a tremendous amount of time and space. This does not happen with HR_P-refutations. Second, HR_P-refutations only need a small part of $CL(P)$, not all of it, and this small part may be constructed as needed.

- The third procedure (Section 5.4), expands upon HR_P, to use tabling, as initially introduced in logic programming by Tamaki and Sato[29]. This procedure assumes caches (or tables) are bounded a priori in size - a situation certainly true in practical implementations where tables cannot grow in an unbounded fashion. Furthermore, table management in probabilistic logic programs is much more complicated than in ordinary logic programming for many reasons. First, a query does not merely have a set of answers. Rather, a query has associated answer substitutions, each of which has an associated probability range. As computation proceeds, these ranges may get refined or sharpened – something that does not happen in classical logic programm tables. Second, caches in our framework may contain basic formulas with associated probabilities. Such caches implicitly contain probability ranges for basic formulas implied by the cached formulas, as well as basic formulas that imply the cached formulas. A third difference between our work and classical logic program tabling is that there are often many ways to update a table in the case of probabilistic logic programs. We define cache update strategies, and show several different such strategies. We show how HR_P-refutations may be extended with arbitrary cache update strategies.

Unlike classical resolution, when dealing with annotated conjunctions and disjunctions, unifiers may not be unique, as noted by Ng and Subrahmanian[22]. Before proceeding to describe our different notions of resolution, we summarize observation of [22] below as it is necessary for the further development of our proof procedure.

5.1 Unification in HPPs

As rules of clauses in hp-programs may contain annotated basic formulas, any notion of unification must be able to handle unification of annotated basic formulas. In this section, we recapitulate from [22] how this may be done.

Definition 22
- Θ is a *unifier* of annotated conjunctions $C_1 \equiv A_1 \wedge_i \ldots \wedge_i A_{n_1}$ *and* $C_2 \equiv B_1 \wedge_j \ldots \wedge_j A_{n_2}$ **iff** $i, j \in \mathcal{CONJ}$ and $i = j$ and $\{A_k\Theta | 1 \leq k \leq n_1\} = \{B_k\Theta | 1 \leq k \leq n_2\}$
- Θ is a *unifier* of annotated disjunctions $D_1 \equiv A_1 \vee_i \ldots \vee_i A_{n_1}$ and $D_2 \equiv B_1 \vee_j \ldots \vee_j A_{n_2}$ **iff** $i, j \in \mathcal{DISJ}$ and $i = j$ and $\{A_k\Theta | 1 \leq k \leq n_1\} = \{B_k\Theta | 1 \leq k \leq n_2\}$

In order to proceed we need to define a notion of maximally general unifier.

Definition 23 Let $U(C_1, C_2)$ denote the set of all unifiers of C_1 and C_2. Let $\Theta_1, \Theta_2 \in U(C_1, C_2)$.

1. $\Theta_1 \leq \Theta_2$ **iff** there exists a substitution γ, such that $\Theta_1 = \Theta_2\gamma$.
2. $\Theta_1 \equiv \Theta_2$ **iff** $\Theta_1 \leq \Theta_2$ and $\Theta_2 \leq \Theta_1$.
3. Let $[\Theta] = \{\Theta' \in U(C_1, C_2) | \Theta \equiv \Theta'\}$.
4. $[\Theta_1] \leq [\Theta_2]$ **iff** there exists such γ that $[\Theta_1] = [\Theta_2\gamma]$.
5. $[\Theta_1] < [\Theta_2]$ **iff** $[\Theta_1] \leq [\Theta_2]$ and $[\Theta_2] \neq [\Theta_1]$.

From the above definition, it is easy to see that $\equiv$ is an equivalence relation on elements of $U(C_1, C_2)$ and $\leq$ is a partial order on $\{[\Theta] | \Theta \in U(C_1, C_2)\}$. We can define a notion of maximally general unifier.

Definition 24 $\Theta \in U(C_1, C_2)$ is a maximally general unifier (max-gu) of C_1 and C_2 **iff** there is **no** such other unifier $\Theta' \in U(C_1, C_2)$ that $[\Theta] \leq [\Theta']$.

Lemma 3 *If two basic formulas are unifiable then they have a max-gu (not necessarily unique).*

5.2 hp-Resolution

In general, in the presence of basic formulas, just "straight" resolution is not sufficient for query processing. The reason is that to establish a basic formula, e.g. $(p \wedge_i q) : \mu$, we might need to *separately* prove $p : \mu_1$ and $q : \mu_2$ and then combine μ_1, μ_2 using the composition function associated with p-strategy i. There are two ways to do this: (i) allow resolution not against hp-clauses in P, but against hp-clauses in an *expanded* version of P, or (ii) introduce, in addition to resolution, new rules of inference corresponding the the "expansion" steps alluded above. Both cases are essentially equivalent from the point of view of completeness. In what follows we discuss the former procedure.

First, we add to P all "tautologies". Any formula of the form $F : [0,1]$ is a tautology as F's probability certainly, lies in the $[0,1]$ interval.

Definition 25 Let P be an hp-program. Then REDUN(P) is defined as

$$REDUN(P) = P \cup \{A : [0,1] \longleftarrow | A \in B_L\}.$$

In addition to the above tautologies, we need to "merge" rules together and/or infer "implied" rules. For example, if one rule has $F_1 : \mu_1$ in the head, and another has $F_2 : \mu_2$ in the head, and these are unifiable via max-gu Θ, then these two rules may jointly provide some information on the probability of $(F_1 \wedge_i F_2)$ where i is some p-strategy. Likewise, if $(F_1 *_i F_2) : \mu'$ is in the head of some rule, then this rule certainly provides some information about F_1's probability, and F_2's probability. The closure of P, defined below, expands the rules in P by performing such merges and/or inferences.

Definition 26 Let P be an hp-program. Then CL(P) (closure of P) is defined as follows

- $\mathrm{CL}^0(P) = REDUN(P)$.
- 1. For each pair of clauses $F_1 : \mu_1 \longleftarrow Body_1$ and $F_2 : \mu_1 \longleftarrow Body_2 \in \mathrm{CL}^j(P)$, such that their heads $F_1 : \mu_1$ and $F_2 : \mu_2$ are unifiable via max-gu Θ add clause $(F_1 : \mu_1 \cap \mu_2 \longleftarrow Body_1 \wedge Body_2)\Theta$ to $\mathrm{CL}^{j+1}(P)$.
  2. For each clause $F_1 *_i F_2 : \mu \longleftarrow Body \in \mathrm{CL}^j(P)$ add the following two clauses to $\mathrm{CL}^{j+1}(P)$:
     - $F_1 : md_i(\mu) \longleftarrow Body$
     - $F_2 : md_i(\mu) \longleftarrow Body$
  3. For each two clauses $(A_1 *_i \ldots *_i A_k) : \mu_1 \longleftarrow Body_1$ and $(B_1 *_i \ldots *_i B_l) : \mu_1 \longleftarrow Body_2 \in \mathrm{CL}^j(P)$, $k > 1, l \geq 1$, add the clause

     $$(A_1 *_i \ldots *_i A_k *_i B_1 *_i \ldots *_i B_l) : c_i(\mu_1, \mu_2) \longleftarrow Body_1 \wedge Body_2$$

     to $\mathrm{CL}^{j+1}(P)$.
  4. if A and B are atoms, and $\mathrm{CL}^j(P)$ contains clauses $A : \mu_1 \longleftarrow Body_1$ and $B : \mu_2 \longleftarrow Body_2$, add

     $$(A *_i B) : c_i(\mu_1, \mu_2) \longleftarrow Body_1 \wedge Body_2$$

     for each $i \in \mathcal{CONJ} \cup \mathcal{DISJ}$ to $\mathrm{CL}^{j+1}(P)$.

- $\mathrm{CL}(P) = \bigcup_{j \geq 0} \mathrm{CL}^j(P)$

The following result says that the above steps are all sound. No new rule is produced that was not already a logical consequence of P.

Lemma 4 *For every clause $C \in CL(P)$, $P \models C$.*

Proof. Let C be a clause in $\mathrm{CL}(P)$. We have to consider a number of possibilities.

- Base Case.
  1. $C \in P$. Then by definition of $\models$, $P \models C$.
  2. $C \notin P$, $C \in REDUN(P)$. In this case C is of the form $A : [0,1] \longleftarrow$, and A is a ground instance of an atom. Let h be a formula function, such that $h \models P$. It is always the case that $h(C) \subseteq [0,1]$, which yields $h \models C$.
- Induction Step.

  Assume that for each clause $C \in \mathrm{CL}^j(P)$, $P \models C$. Let $C \in \mathrm{CL}^{j+1}(P) - \mathrm{CL}^j(P)$.

  As $C \in \mathrm{CL}^{j+1}(P) - \mathrm{CL}^j(P)$, C must have been inserted into $\mathrm{CL}^{j+1}(P)$ by the means of one of the cases $1-4$ from **Def. 26**. We have to consider each case separately.

  1. Suppose C was inserted by the means of case 1. Then there exist such clauses $C_1 \equiv F_1 : \mu_1 \longleftarrow Body_1$ and $C_2 \equiv F_2 : \mu_2 \longleftarrow Body_2$, such that, $C_1 \in \mathrm{CL}^j(P), C_2 \in \mathrm{CL}^j(P)$, F_1 and F_2 are unifiable via max-gu Θ, and
  $$C \equiv (F_1 : \mu_1 \cap \mu_2 \longleftarrow Body_1 \wedge Body_2)\Theta$$
  We need to show that $P \models C$. Suppose h is a model of P, i.e., $h \models P$, and $C\gamma$ is a ground instance of C, such that $h \models (Body_1 \wedge Body_2)\Theta\gamma$. By the induction hypothesis, $h \models C_1$ and $h \models C_2$, therefore, $h \models C_1\Theta\gamma$ and $h \models C_2\Theta\gamma$. As $h \models Body_1\Theta\gamma$, we conclude that $h(F_1\Theta\gamma) \subseteq \mu_1$. Likewise we can conclude that $h(F_2\Theta\gamma) \subseteq \mu_2$.

     But since Θ is a max-gu of F_1 and F_2, $F_1\Theta\gamma = F_2\Theta\gamma$, and therefore $h(F_1\Theta\gamma) \subseteq \mu_1 \cap \mu_2$, i.e. $h \models F_1\Theta\gamma : \mu_1 \cap \mu_2$.
  2. Suppose C was inserted by the means of case 2. Then there exists such a clause $C_1 \equiv (F_1 *_i F_2) : \mu \longleftarrow Body \in \mathrm{CL}^j(P)$, that either
  $$C \equiv F_1 : md_i(\mu) \longleftarrow Body$$
  or
  $$C \equiv F_2 : md_i(\mu) \longleftarrow Body.$$
  We will consider the former case, the latter case is symmetric. We need to show that $P \models C$. Let $C\gamma$ be a ground insance of C and let $h \models P$ and $h \models Body\gamma$. By induction hypothesis, $h \models C_1$, and therefore, $h((F_1 *_i F_2)\gamma) \subseteq \mu$. By the definitions of md_i and h, this yields $h(F_1) \subseteq md_i(\mu)$, i.e., $h \models F_1 : md_i(\mu)$.
  3. Let C be inserted by the means of case 3. In this case, $\mathrm{CL}^j(P)$ will contain two clauses, $C_1 \equiv (A_1 *_i \ldots *_i A_k) : \mu_1 \longleftarrow Body_1$ and $C_2 \equiv (B_1 *_i \ldots *_i B_l) : \mu_2 \longleftarrow Body_2$, such that, $k > 1, l \geq 1$, and
  $$C \equiv (A_1 *_i \ldots *_i A_k *_i B_1 *_i \ldots *_i B_l) : c_i(\mu_1, \mu_2) \longleftarrow Body_1 \wedge Body_2.$$

We need to show $P \models C$. Let $C\gamma$ be a ground instance of C and let $h \models P$ and $h \models (Body_1 \wedge Body_2)\gamma$. By induction hypothesis, $h \models C_1$ and $h \models C_2$, and therefore, $h \models C_1\gamma$ and $h \models C_2\gamma$. Since $h \models Body_1\gamma$ and $h \models Body_2\gamma$, we have $h \models (A_1 *_i \ldots *_i A_k)\gamma : \mu_1$ and $h \models (B_1 *_i \ldots *_i B_l)\gamma : \mu_2$, i.e., $h((A_1 *_i \ldots A_k)\gamma) \subseteq \mu_1$, and $h((B_1 *_i \ldots *_i B_l)\gamma) \subseteq \mu_2$. But then,

$$h((A_1 *_i \ldots *_i A_k *_i B_1 *_i \ldots *_i B_l)\gamma) = c_i(h((A_1 *_i \ldots *_i A_k)\gamma), h((B_1 *_i \ldots *_i B_l)\gamma)) \subseteq c_i(\mu_1, \mu_2),$$

which means $h \models C$.

4. Finally, let C be inserted in $\mathrm{CL}^{j+1}(P)$ by the means of case 4. Then, $\mathrm{CL}^j(P)$ will contain 2 clauses, $C_1 \equiv A : \mu_1 \longleftarrow Body_1$ and $C_2 \equiv B : \mu_2 \longleftarrow Body_2$, such that both A and B are atomic, and
$$C \equiv (A *_i B) : c_i(\mu_1, \mu_2) \longleftarrow Body_1 \wedge Body_2$$
for some p-strategy i.
We have to show $P \models C$. Let $C\gamma$ be a ground instance of C and let $h \models P$ and $h \models (Body_1 \wedge Body_2)\gamma$. By induction hypothesis, $h \models C_1$ and $h \models C_2$, therefore, $h \models C_1\gamma$ and $h \models C_2\gamma$. Since $h \models Body_1\gamma$ and $h \models Body_2\gamma$, we obtain $h \models A\gamma : \mu_1$ and $h \models B\gamma : \mu_2$, i.e. $h(A\gamma) \subseteq \mu_1$ and $h(B\gamma) \subseteq \mu_2$. Hence, $h((A *_i B)\gamma) = c_i(h(A\gamma), h(B\gamma)) \subseteq c_i(\mu_1, \mu_2)$, which means that $h \models C\gamma$ and thefore $h \models C$. ■

We now present a refutation procedure for query processing.

Definition 27 A query is a formula of the form $\exists(F_1 : \mu_1 \wedge \ldots \wedge F_n : \mu_n)$, where $(\forall 1 \leq i \leq n)$ $(F_i \in bf_{\mathcal{S}}(B_L))$. F_is need not be ground.

Definition 28 Suppose $C \equiv G : \lambda \longleftarrow G_1 : \lambda_1 \wedge \ldots \wedge G_m : \lambda_m \in \mathrm{CL}(P)$ and $Q \equiv \exists(F_1 : \mu_1 \wedge \ldots \wedge F_n : \mu_n)$ is a query. Let C and Q be standardized apart. Let also G and F_i be unifiable for some $1 \leq i \leq n$. Then

$$\exists((F_1 : \mu_1 \wedge \ldots \wedge F_{i-1} : \mu_{i-1} \wedge G_1 : \lambda_1 \wedge \ldots \wedge G_m : \lambda_m \wedge F_{i+1} : \mu_{i+1} \wedge \ldots \wedge F_n : \mu_n)\Theta)$$

is an hp-resolvent of C and Q **iff**:

1. Θ is a max-gu of G and F_i
2. $\lambda\Theta$ and $\mu_i\Theta$ are ground and $\lambda\Theta \subseteq \mu_i\Theta$

If Θ is a unifier but not necessarily a max-gu, we call the resolvent an unrestricted hp-resolvent.

Definition 29 Let $Q \equiv \exists(F_1 : \mu_1 \wedge \ldots \wedge F_n : \mu_n)$ be an initial query, and P an hp-program. An hp-deduction of Q from P is a sequence $< Q_1, C_1, \Theta_1 > \ldots < Q_r, C_r, \Theta_r > \ldots$ where, $Q = Q_1$, for all $i \geq 1$, C_i is a renamed version of a clause in $\mathrm{CL}(P)$ and Q_{i+1} is an *hp-resolvent* of Q_i and C_i via max-gu Θ_i.
If the Θ_i's are not restricted to be max-gu's, we call the resulting sequence an unrestricted hp-deduction.

Definition 30 Let $Q \equiv \exists(F_1 : \mu_1 \wedge \ldots \wedge F_n : \mu_n)$ be an initial query, and P an hp-program. An hp-refutation of Q from P is a *finite hp-deduction* $< Q_1, C_1, \Theta_1 > \ldots < Q_r, C_r, \Theta_r >$ where, the hp-resolvent of Q_r and C_r via Θ_r is the empty query. $\Theta_1 \ldots \Theta_r$ is called the *computed answer substitution.*

We are now in a position to state the soundness and completeness of hp-resolution.

Theorem 4 (Soundness of hp-refutation).
Let P be an hp-program, and Q be an initial query. If there exists an hp-refutation of $Q \equiv \exists(F_1 : \mu_1 \wedge \ldots \wedge F_n : \mu_n)$ from P with the answer substitution Θ then $P \models \forall((F_1 : \mu_1 \wedge \ldots \wedge F_n : \mu_n)\Theta)$.

Proof. Let $< Q_1, C_1, \Theta_1 > \ldots < Q_n, C_n, \Theta_n >$ be our hp-refutation. We proceed by induction on n.

Base case: $n = 1$

In this case $Q_1 \equiv F_1 : \mu_1$, $C_1 \equiv G_1 : \nu_1 \longleftarrow \in \mathrm{CL}(P)$, $F_1\Theta_1 = G_1\Theta_1$ and $\nu_1 \subseteq \mu_1$. Let $h \models P$. By the previous lemma, $h \models C_1$. Therfore, $h \models \forall(G_1 : \nu_1)$ and in particular $h \models \forall((G_1 : \nu_1)\Theta_1)$. But, since $F_1\Theta_1 = G_1\Theta_1$ and $\nu_1 \subseteq \mu_1$, we get $h \models \forall(F_1 : \mu_1)\Theta_1)$.

Induction Step.

Suppose the theorem holds for any hp-refutation $< Q_2, C_2, \Theta_2 > \ldots < Q_n, C_n, \Theta_n >$. Consider an hp-refutation $< Q_1, C_1, \Theta_1 >, < Q_2, C_2, \Theta_2 > \ldots < Q_n, C_n, \Theta_n >$. Let $h \models P$. Let $Q_1 \equiv (F_1 : \mu_1 \wedge \ldots \wedge F_m : \mu_m)$ and $C_1 \equiv G : \nu \longleftarrow \mathit{Body}$ be (a renamed version of) a clause in $\mathrm{CL}(P)$, such that for some $1 \leq i \leq m$, $F_i\Theta_1 = G\Theta_1$ and $\nu \subseteq \mu_i$. Then, $Q_2 \equiv (F_1 : \mu_1 \wedge \ldots \wedge F_{i-1} : \mu_{i-1} \wedge \mathit{Body} \wedge F_{i+1} : \mu_{i+1} \wedge \ldots \wedge F_m : \mu_m)\Theta_1$. By induction hypothesis, $h \models \forall(Q_2\Theta_2 \ldots \Theta_n)$, i.e. $h \models \forall(F_1 : \mu_1 \wedge \ldots \wedge F_{i-1} : \mu_{i-1} \wedge \mathit{Body} \wedge F_{i+1} : \mu_{i+1} \wedge \ldots \wedge F_m : \mu_m)\Theta_1\Theta_2 \ldots \Theta_n)$. Therefore, $h \models \forall(F_1 : \mu_1 \wedge \ldots \wedge F_{i-1} : \mu_{i-1} \wedge F_{i+1} : \mu_{i+1} \wedge \ldots \wedge F_m : \mu_m)\Theta_1\Theta_2 \ldots \Theta_n)$ and $h \models \forall(\mathit{Body}\Theta_1 \ldots \Theta_n)$. Since also $h \models C_1$, we obtain $h \models \forall((G\Theta_1 \ldots \Theta_n) : \nu)$. Since $\nu \subseteq \mu_i$, we obtain $h \models \forall((F_i\Theta_1 \ldots \Theta_n) : \mu_i))$, i.e. $h \models \forall((F_i : \mu_i)\Theta_1 \ldots \Theta_n)$. Combining with $h \models \forall(F_1 : \mu_1 \wedge \ldots \wedge F_{i-1} : \mu_{i-1} \wedge F_{i+1} : \mu_{i+1} \wedge \ldots \wedge F_m : \mu_m)\Theta_1\Theta_2 \ldots \Theta_n)$ we get the desired: $h \models \forall(Q_1\Theta_1 \ldots \Theta_n)$, i.e., $h \models \forall(Q_1\Theta)$. ■

In order to prove completeness thorem we have to establish first a number of facts. The following two lemmas can be proved by a straightforward application of mgu and lifting lemmas for classical logic programming in Lloyd[21].

Lemma 5 *(***Max-gu Lemma***). Let Q be a query that has an* unrestricted *hp-refutation from an hp-program P. Then, Q has an hp-refutation of the same length and if $\Theta_1, \ldots, \Theta_m$ are the unifiers form the unrestricted hp-refutation, and $\Theta'_1, \ldots, \Theta'_m$ are the max-gu's from the hp-refutation, then, for some γ $\Theta_1 \ldots \Theta_m = \Theta'_1, \ldots, \Theta'_m\gamma$.*

Lemma 6 *(***Lifting Lemma***) Let P be an hp-program, Q be a query, Θ be a substitution. Let $Q\Theta$ have an hp-refutation from P. Then Q has an hp-refutation from P of the same length. Also, if $\Theta_1, \ldots, \Theta_m$ are the max-gu's from the refutation of $Q\Theta$ and $\Theta'_1, \ldots, \Theta'_m$ are the max-gu's from the refutation of Q then, for some substitution γ: $\Theta\Theta_1 \ldots \Theta_m = \Theta'_1, \ldots, \Theta'_m\gamma$.*

Now we can prove completeness theorem.

Theorem 5 (Completeness of hp-refutation).
Let P be a consistent hp-program and Q' be a query. Then, if $P \models \exists(Q')$ then there exists an hp-refutation of Q' from P.

Proof.

Since $P \models \exists(Q')$, there exists such a ground substitution Θ that $P \models Q'\Theta$. Let $Q \equiv Q'\Theta$. We will prove that Q has an hp-refutation from P. By Lifting Lemma, Q' will also have a refutation from P.

Let $Q \equiv F_1 : \mu_1 \wedge \ldots \wedge F_m : \mu_m$. Since $P \models Q$, it must be the case that $P \models F_i : \mu_i$, $1 \leq i \leq m$.

Claim 1: *Let $F : \mu$ and $G : \nu$ be ground annotated formuals which have hp-refutations from P. Then, so does $F : \mu \wedge G : \nu$.*

Proof. Let $< F : \mu, C_1, \Theta_1 > \ldots < Q^F_l, C_l, \Theta_l >$ be the hp-refutation for $F : \mu$. Let $< G : \nu, D_1, \Gamma_1 >$ $\ldots < Q^G_k, D_k \Gamma_k >$ be the hp-refutation for $G : \nu$. Then, as $F : \mu$ and $G : \nu$ are ground, the following will be the hp-refutation for $F : \mu \wedge G : \nu$:

$$< F : \mu \wedge G : \nu, C_1, \Theta_1 >, < Q^F_2 \wedge G : \nu, C_2, \Theta_2 > \ldots < Q^F_l \wedge G : \nu, C_l, \Theta_l >, < G : \nu, D_1, \Gamma_1 > \ldots < Q^G_k, D_k \Gamma_k >$$

. ■(**Claim 1**).

Now all we have to prove is

Claim 2: *Let $P \models \exists(F' : \mu)$. Then there exists a refutation of $F' : \mu$ from P.*

Proof. Since $P \models F' : \mu$, there exists a ground substitution θ such that $P \models F'\theta : \mu$. Let $F = F'\theta$. We show that $F : \mu$ has an hp-refutation from P, and by *lifting lemma* so will $F' : \mu$.

Since $P \models F : \mu$, by **Theorem 3**, $T^\omega_P(F) \subseteq \mu$. By definition of T^ω_P, there exists such an $\alpha < \omega$ that $T^\alpha_P(F) \subseteq \mu$. Consider the smallest such integer. We now proceed by induction on α.

Base Case: $\alpha = 0$. By definition of T^0_P, $T^0_P(F) = [0, 1]$. Therefore, $\mu = [0, 1]$.

If F is atomic, then, since F is ground, a clause

$$C \equiv F : [0, 1] \longleftarrow$$

is in $REDUN(P)$, and therefore it is in $\mathrm{CL}(P)$. Then, $< F : \mu, C, e >$ (e is the empty substitution) is the hp-refutation for $F : \mu$.

Let $F \equiv (A_1 *_i A_2 *_i \ldots *_i A_n)$, where each A_i is atomic. Then, a set of clauses

$$C_i \equiv A_j : [0, 1] \longleftarrow$$

is in $REDUN(P)$ and therefore each of these clauses is in $\mathrm{CL}^0(P)$. By definition of $\mathrm{CL}(P)$ and because for any p-strategy i $c_i([0, 1], [0, 1]) = [0, 1]$, $\mathrm{CL}^n(P)$ (and therefore $\mathrm{CL}(P)$) will contain the clause

$$C \equiv (A_1 *_i A_2 *_i \ldots *_i A_n) : [0, 1] \longleftarrow$$

(In fact we can argue that the above clause will be contained in $\mathrm{CL}^{\log_2(n)}(P)$).

Then the refutation for $F : \mu$ will be $< F : \mu, C, e >$ (e is the empty substitution).

Induction Step. Assume that for any formula $G : \mu$ such that $T^{\alpha-1}_P \models G : \mu$, there exists a refutation ξ of $G : \mu$ from P.

We prove the claim by induction on the structure of F.

Base Case. F is atomic.

Let $M' = \{\mu' | G : \mu' \longleftarrow Body \in P; T^{\alpha-1}_P \models Body$, where G is unifiable with $F\}$. We notice that $S_P(T^{\alpha-1}_P)(F) = \cap\{\mu' | \mu' \in M'\}$.

Let $M'' = \{\mu'' | (G *_i H) : \mu'' \longleftarrow Body \in P; T^{\alpha-1}_P \models Body$, where G is unifiable with $F\}$.

We have, by definition of T^α_P ($\alpha > 0$):

$$T^\alpha_P(F) = S_P(T^{\alpha-1}_P)(F) \cap (\cap\{md(\mu'') | \mu'' \in M''\}) \subseteq \mu$$

.

Two cases are possible.

1. $|M' \cup M''| = 1$.

Assume $M' \neq \emptyset$. Than, there is a unique rule $C' \equiv G : \mu' \longleftarrow Body \in P$, s.t., G unifies with F, $T_P^{\alpha-1} \models Body$, and $S_P(T_P^{\alpha-1})(F) = \mu'$. (Notice that $C' \in P$ implies $C' \in \mathrm{CL}(P)$). Let Θ' be the max-gu for G and F.

By induction hypothesis, there exists an hp-refutation $< Body, C_1, \Theta_1 > \ldots < Q_k, C_k, \Theta_k >$ for $Body$. Then

$$< F : \mu, C', \Theta' >, < Body, C_1, \Theta_1 > \ldots < Q_k, C_k, \Theta_k >$$

is the refutation for $F : \mu$.

Assume now that $M'' \neq \emptyset$. Then there is a unique rule $C' \equiv (G *_i H) : \mu' Body \in P$, s.t., G unifies with F via max-gu Θ', $T_P^{\alpha-1} \models Body$ and $T_P(T_P^{(}\alpha-1)(F) = md_i(\mu')$. Since $C' \in P$, $C' \in \mathrm{CL}^0(P)$ and therfore the following clause $C'' \equiv G : md_i(\mu') \longleftarrow Body$ is in $\mathrm{CL}^1(P)$. By the induction hypothesis, there exists an hp-refutation for $Body$: $< Body, C_1, \Theta_1 > \ldots < Q_k, C_k, \Theta_k >$. Then

$$< F : \mu, C'', \Theta' >, < Body, C_1, \Theta_1 > \ldots < Q_k, C_k, \Theta_k >$$

is the refutation for $F : \mu$.

2. $|M' \cup M''| > 1$

   Let $\mathcal{C}' = \{G : \mu' \longleftarrow Body \in P | T_P^{\alpha-1} \models Body\}$, where G is unifiable with F, and $M' = \{\mu' | G : \mu' \longleftarrow Body \in \mathcal{C}'\}$.

   Let also $\mathcal{C}'' = \{(D *_i H) : \mu'' \longleftarrow Body \in P | T_P^{\alpha-1} \models Body\}$, where D is unifiable with F and $M'' = \{\mu'' | (D *_i H) : \mu'' \longleftarrow Body \in \mathcal{C}''\}$.

   Since all clauses from $\mathcal{C}'$ are in P, they are also in $\mathrm{CL}^0(P)$. Let

$$G_1 : \mu'_1 \longleftarrow Body'_1$$

$$\ldots$$

$$G_s : \mu'_s \longleftarrow Body'_s$$

   be all clauses in $\mathcal{C}'$. Since they are in $\mathrm{CL}^0(P)$, we can claim that the clause

$$C_1 \equiv G_1\Theta' : \mu'_1 \cap \ldots \cap \mu'_s \longleftarrow Body'_1 \wedge \ldots \wedge Body'_s$$

   will be in $\mathrm{CL}^s(P)$ (actually, it will already be in $\mathrm{CL}^{\log_2(s)}(P)$) where Θ' is the max-gu of $G_1, \ldots, G_s$ (such a substitution must exist since we know that each of G_j is unifiable with F).

   Let

$$(D_1 *_{i_1} H_1) : \mu''_1 \longleftarrow Body''_1$$

$$\ldots$$

$$(D_r *_{i_r} H_r) : \mu''_r \longleftarrow Body''_r$$

   be all clauses in $\mathcal{C}''$. Since $\mathcal{C}'' \subseteq P$, every clause in $\mathcal{C}''$ is also in $\mathrm{CL}^0(P)$. Therefore, the following set of clauses:

$$D_1 : md_{i_1}(\mu''_1) \longleftarrow Body''_1$$

$$\ldots$$

$$D_r : md_{i_r}(\mu''_r) \longleftarrow Body''_r$$

   will be a subset of $\mathrm{CL}^1(P)$. Then, we can claim that $\mathrm{CL}^r(P)$ (or even $\mathrm{CL}^{\log_2(r)}(P)$ will contain the following clause:

$$C_2 \equiv D_1\Theta'' : md_{i_1}(\mu''_1) \cap \ldots \cap md_{i_r}(\mu''_r) \longleftarrow Body''_1 \wedge \ldots \wedge Body''_r$$

where Θ'' is the max-gu for $D_1, \ldots D_r$.

Let $l = \max(r, s)$. Since both $C_1 \in \mathrm{CL}^l(P)$ and $C_2 \in \mathrm{CL}^l(P)$, the following clause

$$C \equiv G_1\Theta'_1 : \mu'_1 \cap \ldots \cap \mu'_s \cap md_{i_1}(\mu''_1) \cap \ldots \cap md_{i_r}(\mu''_r) \longleftarrow Body'_1 \wedge \ldots \wedge Body'_s \wedge Body''_1 \wedge \ldots \wedge Body''_r$$

(where Θ'_1 is the max-gu of $G_1\Theta'$ and $D_1\Theta''$) will be in $\mathrm{CL}^{l+1}(P)$ and therefore, in $\mathrm{CL}(P)$.

Notice that $\mu'_1 \cap \ldots \cap \mu'_s \cap md_{i_1}(\mu''_1) \cap \ldots \cap md_{i_r}(\mu''_r) = T^\alpha_P(F) \subseteq \mu$. Also, by induction hypothesis, each $Body'_j$ and $Body''_j$ has an hp-refutation, therefore by **Claim 1** of the theorem, their conjunction has an hp-refutation. Let $< Q_1, C_1, \Theta_1 >, \ldots, < Q_z, C_z, \Theta_z >$ be such an hp-refutation. Than the following is an hp-refutation for $F : \mu$:

$$< F : \mu, C, \Gamma >, < Q_1, C_1, \Theta_1 >, \ldots, < Q_z, C_z, \Theta_z >$$

where Γ is the max-gu of F and $G_1\Theta'_1$.

Induction Step. Assume that the theorem holds for every formula of size less than k and let $F = A_1 *_i \ldots *_i A_k$, where $A_1, \ldots A_k$ are atomic.

Let $\mathcal{C}_1 = \{G : \mu' \longleftarrow Body \in P | T^{\alpha-1}_P \models Body$, where G is unifiable with $F\}$, and $M_1 = \{\mu' | G : \mu' \longleftarrow Body \in \mathcal{C}_2\}$.

Let $\mathcal{C}_2 = \{(D *_i E) : \mu'' \longleftarrow Body \in P | T^{\alpha-1}_P \models Body$, where D is unifiable with $F\}$ and $M_2 = \{\mu'' | (D *_i E) : \mu'' \longleftarrow Body \in \mathcal{C}_2\}$.

Let

$$G_1 : \mu'_1 \longleftarrow Body'_1$$

$$\ldots$$

$$G_s : \mu'_s \longleftarrow Body'_s$$

be all clauses in $\mathcal{C}_1$. Since they are in $\mathrm{CL}^0(P)$, we can claim that the clause

$$C^F_1 \equiv G_1\Theta' : \mu'_1 \cap \ldots \cap \mu'_s \longleftarrow Body'_1 \wedge \ldots \wedge Body'_s$$

will be in $\mathrm{CL}^s(P)$ (actually, it will already be in $\mathrm{CL}^{\log_2(s)}(P)$) where Θ' is the max-gu of $G_1, \ldots, G_s$ (such a substitution must exist since we know that each of G_j is unifiable with F).

Let

$$(D_1 *_{i_1} E_1) : \mu''_1 \longleftarrow Body''_1$$

$$\ldots$$

$$(E_r *_{i_r} E_r) : \mu''_r \longleftarrow Body''_r$$

be all clauses in $\mathcal{C}_2$. Since $\mathcal{C}_2 \subseteq P$, every clause in $\mathcal{C}_2$ is also in $\mathrm{CL}^0(P)$. Therefore, the following set of clauses:

$$D_1 : md_{i_1}(\mu''_1) \longleftarrow Body''_1$$

$$\ldots$$

$$D_r : md_{i_r}(\mu''_r) \longleftarrow Body''_r$$

will be a subset of $\mathrm{CL}^1(P)$. Then, we can claim that $\mathrm{CL}^r(P)$ (or even $\mathrm{CL}^{\log_2(r)}(P)$ will contian the following clause:

$$C_2^F \equiv D_1\Theta'' : md_{i_1}(\mu_1'') \cap \ldots \cap md_{i_r}(\mu_r'') \longleftarrow Body_1'' \wedge \ldots \wedge Body_r''$$

where Θ'' is the max-gu for $D_1, \ldots D_r$.

Now, consider any pair of basic formulas H and I such that $H \oplus I = F$. Since $F \equiv (H *_i I)$, we must conclude that $T_P^\alpha(F) = T_P^\alpha(H *_i I) = c_i(T_P^\alpha(H), T_P^\alpha(I)$. By our assumption $T_P^\alpha(F) \subseteq \mu$ therefore, $c_i(T_P^\alpha(H), T_P^\alpha(I)) \subseteq \mu$. Let $\nu' = T_P^\alpha(H), \nu'' = T_P^\alpha(I)$. We can now say that $T_P^\alpha \models H : \nu_1$ and $T_P^\alpha \models I : \nu_2$, such that $c_i(\nu_1, \nu_2) \subseteq \mu$.

By the induction hypothesis, there exist hp-refutations for $H : \nu'$ and $I : \nu'\prime$. Let

$$< H : \nu', C_1^H, \Theta_1^H >< Q_2^H, C_2^H, \Theta_2^H > \ldots < Q_t^H, C_t^H, \Theta_t^H >$$

and

$$< I : \nu'', C_1^I, \Theta_1^I >< Q_2^I, C_2^I, \Theta_2^I > \ldots < Q_u^I, C_u^I, \Theta_u^I >$$

be these respective hp-refutations. Let us look at the clauses C_1^H and C_1^I. These clauses have to be of a (respective) form:

$$C_1^H \equiv H' : \lambda' \longleftarrow Body'$$

where, $\lambda' \subseteq \nu'$, $T_P^{\alpha-1} \models Body'\prime$, $H\prime$ is unifiable with H, and

$$C_1^I \equiv I' : \lambda'' \longleftarrow Body''$$

where, $\lambda' \subseteq \nu'$, $T_P^{\alpha-1} \models Body''$, I' is unifiable with I.

By definition of hp-refutation, both C_1^H and C_1^I are in $\mathrm{CL}(P)$. Let w be the smallest integer such that both $C_1^I \in \mathrm{CL}^w(P)$ and $C_1^H \in \mathrm{CL}^w(P)$. Then we can claim that $\mathrm{CL}^{w+1}(P)$ will contain the following clause:

$$C^{H *_i I} \equiv (H' *_i I') : c_i(\lambda', \lambda'') \longleftarrow Body' \wedge Body''$$

Since both $Body'$ and $Body''$ have hp-refutations, so does $Body' \wedge Body''$. In fact, we know that $< Q_2^H, C_2^H, \Theta_2^H > \ldots < Q_t^H, C_t^H, \Theta_t^H >$ is an hp-refutation for $Body'$ ($Q_2^H = Body'$) and $< Q_2^I, C_2^I, \Theta_2^I >$ $\ldots < Q_u^I, C_u^I, \Theta_u^I >$ is an hp-refutation for $Body''$ ($Q_2^I = Body''$. Then, the following will be an hp-refutation for $(H' *_i I') : c_i(\lambda', \lambda'')$:

$$< (H' *_i I') : c_i(\lambda', \lambda''), C^{H *_i I}, \Theta >, < Body' \wedge Body'', C_2^H, \Theta_2^H > \ldots < Q_t^H \wedge Body'', C_t^H, \Theta_t^H >, < Body'', C_2^I, \Theta_2^I > .$$

Let now $\mathcal{HI}\{< H_1, I_1 >, \ldots < H_m, I_m >\}$ be *all* possible pairs of basic formulas such that for each $< H, I > \in \mathcal{HI}$ $H \oplus I = F$. By applying the reasoning above we will conclude that for each pair $< H_j, I_j >$ $\mathrm{CL}(P)$ contains a clause

$$C_j \equiv (H_j' *_i I_j') : \lambda_j \longleftarrow Body_j$$

that $\lambda_j \subseteq \mu$, $(H_j' *_i I_j')$ is unifiable with F and H_j is unifiable with H_j' and I_j is unifiable with I_j', $T_P^{\alpha-1} \models Body_j$. Let $q = \max\{q_1, \ldots q_m\}$, where $(\forall 1 \leq j \leq m)(C_j \in \mathrm{CL}^{q_j}(P)$ and $C_j \notin \mathrm{CL}^{q_j-1}(P))$. Then $\mathrm{CL}^{q+m}(P)$ will contain the clause

$$C_3^F \equiv F\Theta_3^F : \lambda_1 \cap \ldots \cap \lambda_m \longleftarrow (Body_1 \wedge \ldots Body_m)\Theta_3^F$$

where Θ_3^F is the max-gu of $(H_1 *_i I_1), \ldots 9H_m *_i I_m)$. Since all $Body_1, \ldots Body_m$ have hp-refutations, so does $Body_1 \wedge \ldots Body_m$ (and therefore $Body_1 \wedge \ldots Body_m)\Theta_3^F$).

Now we can combine clauses C_1^F, C_2^F and C_3^F together into:

$$C^F \equiv F\Theta^F : \mu' \cap \mu'' \cap \lambda \longleftarrow Body^1 \wedge Body^2 \wedge Body^3$$

where Θ^F is a max-gu of the heads of C_1^F, C_2^F and C_3^F, μ',μ'' and λ are probability ranges of C_1^F, C_2^F and C_3^F respectively and $Body^1$, $Body^2$ and $Body^3$ are their respective bodies. It is clear that (i) $\mu' \cap \mu'' \cap \lambda \subseteq \mu$ and (ii) $C^F \in \mathrm{CL}(P)$. We also know that there exists an hp-refututation $< Body^1 \wedge Body^2 \wedge Body^3, C_1^B, \Theta_1^B > \ldots < Q_v^B, C_v^B, \Theta_v^B >$ of $Body^1 \wedge Body^2 \wedge Body^3$. Then the following is an hp-refutation for $F : \mu$:

$$< F : \mu, C^F, \Theta >, < Body^1 \wedge Body^2 \wedge Body^3, C_1^B, \Theta_1^B > \ldots < Q_v^B, C_v^B, \Theta_v^B >$$

, where Θ is a max-gu unifier of F and the head of C^F.

■(Completeness theorem).

5.3 HR_P Refutations for HP-programs

Note that the hp-refutation procedure assumes that $CL(P)$ has been constructed prior to processing a query. In practice, this is an extremely expensive process, both in terms of time taken to construct $CL(P)$, and in terms of space requirements. Even for propositional programs, P, it is easy to see that $CL(P)$ can contain exponentially many clauses. The HR_P refutation framework described here avoids the construction of $CL(P)$.

In the definition below, anytime a formula $(F *_i G)$ is written it is assumed that $* \in \{\wedge, \vee\}$ and if $* = \wedge$ then $i \in \mathcal{CONJ}$ and if $* = \vee$ then $i \in \mathcal{DISJ}$.

Definition 31 *Let P be an hp-program. We define a formal system HR_P as follows:*

1. *Axioms of HR_P include all clauses from P and all clauses of the form: $A : [0,1] \longleftarrow$ where $A \in B_L$.*

2. *Inference Rules. There are 5 inference rule schemes in HR_P.*

   - *A-Composition: Let $A_1, A_2 \in B_L$*

$$\frac{A_1 : \mu_1 \longleftarrow Body_1 \qquad A_2 : \mu_2 \longleftarrow Body_2}{(A_1 *_i A_2) : c_i(\mu_1, \mu_2) \longleftarrow Body_1 \wedge Body_2}$$

   - *F-Composition: Let $A_1, \ldots A_k, B_1 \ldots B_k \in B_L$*

$$\frac{(A_1 *_i \ldots *_i A_k) : \mu_1 \longleftarrow Body_1 \qquad (B_1 *_i \ldots *_i B_l) : \mu_2 \longleftarrow Body_2}{(A_1 *_i \ldots *_i A_k *_i B_1 *_i \ldots *_i B_l) : c_i(\mu_1, \mu_2) \longleftarrow Body_1 \wedge Body_2}$$

   - *Decomposition:* *L-Decomposition* $\frac{(F *_i G):\mu \longleftarrow Body}{F : md_i(\mu) \longleftarrow Body}$ *R-Decomposition* $\frac{(F *_i G):\mu \longleftarrow Body}{G : md_i(\mu) \longleftarrow Body}$

   - *Clarification:*

$$\frac{F_1 : \mu_1 \longleftarrow Body_1 \qquad F_2 : \mu_2 \longleftarrow Body_2}{(F_1 : \mu_1 \cap \mu_2 \longleftarrow Body_1 \wedge Body_2)\Theta}$$

   if F_1 and F_2 are unifiable via max-gu Θ

- *<u>Exchange</u>: Let $A_1, \ldots, A_k \in B_L$, and let $B_1, \ldots, B_k$ be a* **permutation** *of $A_1, \ldots, A_k$*

$$\frac{(A_1 *_i \ldots *_i A_k) : \mu \longleftarrow Body}{(B_1 *_i \ldots *_i B_k) : \mu \longleftarrow Body}$$

3. *A finite sequence $C_1 \ldots C_r$ of hp-clauses is called an* <u>HR_P-**derivation**</u> iff *each clause C_j is either* an axiom *or can be constructed from one or more previous of $C_1 \ldots C_{j-1}$ by applying one of the* inference rule schemes *to them. We call clause C_r* <u>the result</u> *of the HR_P-derivation.*

4. *An hp-clause C is derivable in HR_P* **iff** *there exists such an HR_P-derivation $C_1, \ldots C_r$ that $C_r = C$. We denote it by $P \vdash C$.*

The following theorems tell us that the system of axioms and inference rules describing HR_P precisely captures the closure, $CL(P)$, of P.

Theorem 6 *(soundness of HR_P) For each hp-clause C, if $P \vdash C$ then $C \in CL(P)$.*

Proof. We notice first that the set of all axioms of HR_P is exactly $P \cup REDUN(P) = \text{CL}^0(P)$. Next we notice that the first 4 inference rule schemes preciesely match the 4 rules used to add new hp-rules to $\text{CL}(P)$. Finally, the last inference rule scheme (*Exchange*) does not create a new basic formula, it just rearranges the order of atoms in it. ■

Theorem 7 *(completeness of HR_P) For each hp-clause C, if $C \in CL(P)$ then $P \vdash C$.*

Proof. if $C \in \text{CL}(P)$ then there exists such an integer n that $C\text{CL}^n(P)$ and $C \notin \text{CL}^{n-1}(P)$. We prove the theorem using induction on n.

In the base case $n = 0$ and we know that $\text{CL}^0(P) = P \cap REDUN(P)$. As it was noticed in the previous theorem, this set is exactly the set of all axioms of HR_P, therefore, C is an axiom of HR_P.

On the induction step, we consider a clause C added to $\text{CL}^n(P)$. By **Def 26** C was added to $\text{CL}^n(P)$ by one of four rules. Since these rules match exactly the four inference rules of HR_P and by induction hypothesis for every clause $C'\text{CL}^{n-1}(P)$ we know that $P \vdash_{HR_P} C'$, we can obtain the proof of C in HR_P by application of a matching rule to the same clauses. ■

Definition 32 Let $Q \equiv \exists(F_1 : \mu_1 \wedge \ldots \wedge F_n : \mu_n)$ be an initial query, and P an hp-program. An hp-refutation via HR_P of Q from P is a *finite* sequence $< Q_1, C_1, \Theta_1 > \ldots < Q_r, C_r, \Theta_r >$ where,

- $Q_1 = Q$
- Q_r is empty
- $P \vdash C_i$ for all $1 \leq i \leq r$
- Q_{i+1} is an hp-resolvent of Q_i and C_i with max-gu Θ_i, for all $1 \leq i < r$.

The following results tell us that hp-refutations using HR_P are both sound and complete and thus,they constitute the first sound and complete proof procedure for probabilistic logic programs (including those in [22]) that do not require the construction of a program closure. Here is a simple example of HR_P-refutations.

Example 6 (**HR_P refutations**) Consider the HP-program P given by:

$a : [1, 1] \longleftarrow (b \wedge_{ind} c \wedge_{ind} d) : [0.25, 1] \wedge f : [0.5, 0.9].$
$e : [1, 1] \longleftarrow (b \wedge_{ind} c \wedge_{ind} d) : [0.25, 1] \wedge f : [0.5, 1].$
$(f \wedge_{ind} g) : [0.7, 0.8] \longleftarrow b : [1, 1].$
$(f \vee_{ig} g) : [0.7, 0.9] \longleftarrow .$
$b : [1, 1] \longleftarrow (c \wedge_{ind} d) : [0.3, 1].$
$c : [0.6, 1] \longleftarrow .\ d : [0.5, 1] \longleftarrow .$

A refutation of the query $Q = a[0.9, 1] \wedge e : [1, 1]$ is given by:

$Q_1 = Q = a[0.9, 1] \wedge e : [1, 1]$
$P \ni C_1 = a : [1, 1] \longleftarrow (b \wedge_{ind} c \wedge_{ind} d) : [0.25, 1] \wedge f : [0.5, 0.9].$

$Q_2 = (b \wedge_{ind} c \wedge_{ind} d) : [0.25, 1] \wedge f : [0.5, 0.9] \wedge e : [1, 1]$
$P \vdash C_2 = (b \wedge_{ind} c \wedge_{ind} d) : [0.3, 1] \longleftarrow (c \wedge_{ind} d) : [0.3, 1].$

$Q_3 = (c \wedge_{ind} d) : [0.3, 1] \wedge f : [0.5, 0.9] \wedge e : [1, 1]$
$P \vdash C_3 = (c \wedge_{ind} d) : [0.3, 1]) \longleftarrow$

$Q_4 = f : [0.5, 0.9] \wedge e : [1, 1]$
$P \vdash C_4 = f : [0.7, 0.9] \longleftarrow b : [1, 1].$

$Q_5 = b : [1, 1] \wedge e : [1, 1]$
$P \ni C_5 = b : [1, 1] \longleftarrow (c \wedge_{ind} d) : [0.3, 1].$

$Q_6 = (c \wedge_{ind} d) : [0.3, 1] \wedge e : [1, 1]$
$P \vdash C_6 = (c \wedge_{ind} d) : [0.3, 1]) \longleftarrow$

$Q_7 = e : [1, 1]$
$P \ni C_7 = e : [1, 1] \longleftarrow (b \wedge_{ind} c \wedge_{ind} d) : [0.25, 1] \wedge f : [0.5, 0.1].$

$Q_8 = (b \wedge_{ind} c \wedge_{ind} d) : [0.25, 1] \wedge f : [0.5, 0.1].$
$P \vdash C_8 = (b \wedge_{ind} c \wedge_{ind} d) : [0.3, 1] \longleftarrow (c \wedge_{ind} d) : [0.3, 1].$

$Q_9 = (c \wedge_{ind} d) : [0.3, 1] \wedge f : [0.5, 0.1].$
$P \vdash C_9 = (c \wedge_{ind} d) : [0.3, 1]) \longleftarrow$

$Q_{10} = f : [0.5, 0.1].$
$P \vdash C_{10} f : [0.7, 0.9] \longleftarrow b : [1, 1].$

$Q_{11} = b : [1, 1].$
$P \ni C_{11} = b : [1, 1] \longleftarrow (c \wedge_{ind} d) : [0.3, 1].$

$Q_{12} = (c \wedge_{ind} d) : [0.3, 1].$
$P \vdash C_{12} = (c \wedge_{ind} d) : [0.3, 1]) \longleftarrow$

$Q_{13} = \Box$

Theorem 8 (Soundness of HR_P)-Refutations).
Let P be an hp-program, and Q be an initial query. If there exists an hp-refutation via HR_P of $Q \equiv \exists(F_1 : \mu_1 \wedge \ldots \wedge F_n : \mu_n)$ from P with the answer substitution Θ then $P \models \forall((F_1 : \mu_1 \wedge \ldots \wedge F_n : \mu_n)\Theta)$.

Theorem 9 (Completeness of HR_P)-Refutations).
Let P be a consistent hp-program and Q' be a query. Then, if $P \models \exists(Q')$ then there exists an hp-refutation of Q' from P via HR_P.

Both theorems above follow immediately from the soundness and completeness theorems for HP-refutation and soundness and completeness theorems for HR_P w.r.t. CL(P). Before concluding this section, we briefly reiterate that HR_P refutations avoid compile-time construction of $CL(P)$ – an expensive and time/space consuming proce3ss.

5.4 B-Cache

We are now ready to study efficient tabled query processing techniques for HPPs. In this section, we will first define *caches* and *bounded caches*. Intuitively, a cache contains formulas with established probability ranges. As resolution based processing of a query occurs, we will gain information about certain basic formulas. These will need to be "added" to the cache. For this purpose, we will define in this section, a family of *updating strategies* and introduce several example strategies. Later, in Section 6, we will show how to use these tables and table update strategies for hand in hand with the resolution based proof procedure.

5.4.1 Definitions

Definition 33 *A cache is a finite set of annotated basic formulas. If b is an integer, a bounded b-cache is a finite set of annotated basic formulas containing at most b atoms each.*

Basically a b-cache is a collection of hybrid probabilistic basic formulas, where each formula's length is bounded by a constant b. Note that a b-cache may be considered to be a hybrid probabilistic logic program all of whose clauses are "facts".

Definition 34 *Let T be a b-cache, F be a basic formula (not necessarily ground). By $T[F]$ we denote the set of all such pairs $\{< \mu, \Theta >\}$, where Θ is a substitution for F and $\mu \subseteq [0,1]$ is the smallest interval such that $T \models \forall(F\Theta : \mu)$.*

Intuitively $T[F]$ represents what the b-cache T "thinks" about the possible probability ranges of instances of F. Note that if F is ground, then $\{\mu | < \mu, \Theta > \in T[F]\}$ is a singleton set. Without loss of generality we will abuse notation in this case and write $T[F] = \mu$.

5.4.2 *B*-Cache Update strategies

We fix an integer $b > 0$, a language L and a set S of p-strategies. Let $\mathcal{T}[b, L, \mathcal{S}]$ denote the set of all possible b-caches over $bf_S(B_L)$. Whenever b, L and $\mathcal{S}$ are clear from the context we may use $\mathcal{T}$ instead of $\mathcal{T}[b, L, \mathcal{S}]$.

We are interested in developing a resolution-based query processing procedure that is irredundant in the sense that it does not "re-infer" facts that it has already inferred. In the case of classical logic programs, caches and their utilization are relatively simple: caches contain facts; when performing resolution on an atom A in the query, we check to see if A is subsumed by the cache (Tamaki and Sato[29]). An alternative approach is due to Warren et. al. who check the cache for variants of A [8, 6]. However, in the case of probabilistic logic programs, b-caches are somewhat more complicated.

As the resolution triggered by a query proceeds, more and more information is being established and any time new information is obtained, we want to insert it to our b-cache. However simple addition of a new basic formula to T is not enough, because as we add new probabilistic information - we might be able to update the probability intervals for some other basic formulas already in T. Also, the way such an update can be defined is not unique - in fact, there is a variety of possible "intuitive" updates.

Rather than defining a specific update procedure, we first proceed by defining a notion of an *update strategy* - a function that takes a b-cache and a basic formula as input, and returns a new "improved"

b-cache. We will establish a number of basic properties of *any* update strategy. Later we will define a number of specific update strategies that are "natural" or "intuitive".

In the definition below $CN(S)$, where S is a set of hp-formulas denotes the set of all logical consequences of S.

Definition 35 *A function $f : \mathcal{T} \times bf_{\mathcal{S}}(B_L) \times \mathcal{C}[0,1] \longrightarrow \mathcal{T}$ is called a b-cache update strategy* **iff** *it satisfies the following conditions:*

1.
$$(\forall T \in \mathcal{T})(\forall F \in bf_{\mathcal{S}}(B_L))(\forall \mu \in \mathcal{C}[0,1]) CN(T) \subseteq CN(f(T,F,\mu)) \subseteq CN(T \cup \{F : \mu\})$$

2.
$$(\forall T \in \mathcal{T})(\forall F, G \in bf_{\mathcal{S}}(B_L))(\forall \mu, \nu \in \mathcal{C}[0,1]) f(f(T,F,\mu),G,\nu) = f(f(T,G,\nu),F,\mu)$$

We will use the $\uplus$ operator to denote b-cache update functions. *When more than one update function is considered, we will use the $\uplus_f$ notation and annotate F with μ. (so, $f(T,F,\mu) = T \uplus_f F : \mu$).*

Clause (1) in the above definition says that an update of a b-cache (i)should *not* decrease the amount of information that is contained in, or that can be deduced from the b-cache but at the same time (ii) may not increase the content of the table "unreasonably". Notice that b-caches, by their very definition, automatically pose certain restrictions on how complete the update is - if the length of an updating formula is greater than b - the formula itself cannot be stored in the b-cache.

Clause (2) of the above definition says that the order in which we apply the update operator f should not matter. Updating a table T with $F : \mu$ first and then $G : \nu$ should be the same as doing it the other way around.

Definition 36 *Let P be an hp-program and T be a b-cache. We say that T is* <u>*sound*</u> *w.r.t. P ($P \models T$) iff for each formula $F : \mu \in T$, $P \models F : \mu$.*

Lemma 7 *(soundness of b-cache update). Let P be an hp-program, T be a b-cache and F be a basic formula. Let f be any b-cache update strategy. Then if $P \models T$ and $P \models F : \mu$ then also $P \models T \uplus_f \{F : \mu\}$.*

Proof. Let $F' : \mu' \in T \uplus_f F : \mu$. Two cases are possible.

1. $F' : \mu' \in T$. In this case, since $P \models T$, it has to be $P \models F' : \mu'$.
2. $F' : \mu' \notin T$. We know that $T \uplus_f F : \mu \models F' : \mu'$, hence $F' : \mu' \in CN(T \uplus_f F : \mu)$. We also know that $CN(T \uplus_f F : \mu) \subseteq CN(T \cup \{F : \mu\})$, therefore, we can obtain that $T \cup \{F : \mu\} \models F' : \mu'$. But, $P \models T$ and $P \models F : \mu$ implies that $P \models T \cup \{F : \mu\}$. Combining the obtained results together we get $P \models F' : \mu'$.

In order to simplify notation we define an update of a b-cache with a finite set of formulas as follows:

Definition 37 *Let $S = \{F_1 : \mu_1, \ldots, F_n : \mu_n\}$ be a finite set of annotated basic formulas and u- a b-cache update strategy. We define*

$$T \uplus_u S = (\ldots(T \uplus_u F_1 : \mu) \uplus_u \ldots) \uplus_u F_n : \mu_n)$$

The order in which we write F_is is irrelevant as by the second property of the b-cache update strategy (commutativity), the result of updating a b-cache with a sequence of basic formulas does not depend on the order of formulas. (Second property establishes it for a sequence of 2 basic formulas. It is easily extended onto the case of sequences of 3 or more formulas).

As the reader may notice, there are numerous functions that satisfy the definition of an update strategy. Some of these are intuitively "more complete" than others. The following definition captures this informal notion.

Definition 38 *Let u and w be two b-cache update strategies. We say that u is* <u>*more complete*</u> *than w (denoted $u \geq w$)* **iff** $(\forall T \in \mathcal{T})(\forall F \in bf_{\mathcal{S}}(B_L))(\forall \mu \in \mathcal{C}[0,1])CN(T \uplus_w F : \mu) \subseteq CN(T \uplus_u F : \mu)$.
Two update strategies u and w are <u>*equivalent*</u> **iff** *if both $u \geq w$ and $w \geq u$.*
An update strategy u is <u>*maximally complete*</u> **iff** $(\forall w)(u \geq w)$.

5.4.3 Examples of Update Strategies

In this section, we will provide examples of a number of different update strategies, and show how these strategies are related to one another w.r.t. the "more complete" relationship.

The first kind of update strategy we consider is a relatively simple "atomic update."

Definition 39 (Atomic Updates) *Let T be a b-cache and A be an atomic (not necessarily ground) formula. An* <u>*atomic update*</u> *of T by $A : \mu$, denoted $T \uplus_{at} \{A : \mu\}$ is defined as follows:*

1. *If T has no atomic formulas that* <u>*unify*</u> *with $A : \mu$, then $T \uplus_{at} \{A : \mu\} = T \cup \{A : \mu \longleftarrow\}$*
2. *Otherwise we proceed in a number of steps:*
   (a) *If there is a formula $A : \nu$ in T, we replace it with $A : \mu \cap \nu$.*
   (b) *For all B, such that $B : \nu \in T$ and $A\Theta = B$ for some substitution Θ, we replace $B : \nu$ with $B : \nu \cap \mu$.*
   (c) *Let $\mathcal{B} = \{\nu \mid B : \nu \in T \wedge (\exists \Theta) B\Theta = A\}$. We add $A : \mu \cap (\cap_{\nu \in \mathcal{B}}\{\nu\})$ to T.*
   (d) *For each B such that $B : \nu \in T$ and $A\Theta_1 = B\Theta_2$ for some substitutions Θ_1 and Θ_2 we add $A\Theta_1 : \mu \cap \nu$ to T.*
   (e) *If no clause for A had been added to T on previous steps, we add $A : \mu$ to T.*

An atomic update is not a "complete" b-cache update per se, but it will be at the core of a number of updates that we consider further. Informally, we can describe this process as follows: we check to see if T contains any formulas unifiable with A. If not, we just add $A : \mu$ to T. Otherwise, we look for formulas in T which have probabilities that can affect the probability of A, or vice versa (see example). Then we update probability ranges for all such formulas.

Example 7 Suppose our b-cache $T = \{p(a,Y) : [0.4, 0.7], p(b,Y) : [0.6, 0.9], p(X,a) : [0.5, 1]\}$ Below we show the results of $T \uplus_{at} A$ for a number of given atoms (we consider variables in all the formulas to be standardized apart).

A	$T \uplus_{at} A$
$p(X,Y) : [0.5, 0.95]$	$\{p(a,Y) : [0.5, 0.7], p(b,Y) : [0.6, 0.9], p(X,a) : [0.5, 0.95], p(X,Y) : [0.5, 0.95]\}$
$p(a,a) : [0.3, 0.6]$	$\{p(a,Y) : [0.4, 0.7], p(b,Y) : [0.6, 0.9], p(X,a) : [0.5, 1], p(a,a) : [0.4, 0.6]\}$
$p(b,Z) : [0.4, 0.8]$	$\{p(a,Y) : [0.4, 0.7], p(b,Y) : [0.6, 0.8], p(X,a) : [0.5, 1], p(b,a) : [0.5, 0.8]\}$

Atomic updates do not update annotated basic formulas that are not atomic, and hence the cache that results from an atomic update may not be maximally complete, i.e. it may be the case that $T \cup \{F : \mu\} \models G : \mu'$, but $(T \uplus_{at} F : \mu) \not\models G : \mu'$ for a non-atomic G. An alternative update strategy that propagates such updates is given below.

Definition 40 (Propagated Atomic Update - *pat*) Let T be a b-cache, F be a basic formula. A <u>*Propagated Atomic Update*</u> strategy (*pat*) is defined as follows:

1. *F is atomic. $T \uplus_{pat} F : \mu = T \uplus_{at} F : \mu$.*
2. *Let $F = (F_1 *_i \ldots *_i F_m)$. $T \uplus_{pat} F : \mu = (\ldots(T \uplus_{at} F_1 : md_i(\mu)) \uplus_{at} \ldots) \uplus_{at} F_m : md_i(\mu))$*

The Propagated Atomic Update strategy extends atomic updates onto complex formulas. Among the advantages of this strategy are its relative simplicity and the fact that it works for any bound b on a b-cache. However it is a rather weak strategy in the sense that because every updating formula gets broken into the atoms that constitute it, some information about the probability ranges of associated formulas is lost, i.e. it is not maximally complete. The following example demonstrates this fact.

Example 8 *Let $T = \emptyset$ and $F = (p(a) \wedge_{inc} q(a)) : [0.3, 0.6]$. By definition $T' = T \uplus_{pat} F = \{q(a) : [0.3, 1], p(a) : [0.3, 1]\}$. Now we have $T'[(p(a) \wedge_{inc} q(a)] = [0.3 * 0.3, 1] = [0.09, 1] \supseteq [0.3, 0.6]$. However, if the bound b is greater than 1, we could try to store F itself in T', and preserve information about its probability range.*

The above example suggests how the PAT strategy can be modified to be able to be more complete.

Definition 41 (Elementary b-cache update) *Let T be a b-cache, F be a basic formula. We define an <u>elementary b-cache update</u> strategy (denoted $\uplus_{eb}$ as follows:*

1. **Case 1.** *$|F| = 1$. (F is atomic). $T \uplus_{eb} \{F : \mu\} = T \uplus_{at} \{F : \mu\}$*
2. **Case 2.** *$1 < |F| \leq b$. Let $F \equiv F_1 *_i \ldots *_i F_m$. We proceed in a number of steps.*
   (a) *Let $T' = T \uplus_{pat} F : \mu$*
   (b) *Let $\{< \nu_1, \Theta_1 >, \ldots < \nu_k, \Theta_k > \subseteq T'[F]$ be all pairs from $T'[F]$, s.t., $\nu \not\subseteq \mu$. We proceed in steps. Let $T^0 = T'$. Consider T^i $(0 \geq i < k)$ constructed. We now construct T^{i+1}.*
      - *If $F\Theta_{i+1} : \nu \in T'$ we replace it with $F : \mu \cap \nu$ and declare the new b-cache to be the result of an update operation, i.e. $T^{i+1} = (T^i - \{F\Theta_{i+1} : \nu\}) \cup \{F\Theta_{i+1} : \mu \cap \nu\}$*
      - *If $F\Theta_{i+1} : \nu \notin T'$ we declare $T^{i+1} = T^i \cup \{F\Theta_{i+1} : \mu \cap \nu\}$.*

   (c) *Now we declare $T \uplus_{eb} F : \mu = T^k$.*
3. **Case 3.** *$|F| > b$. Let $F = F_1 *_i \ldots *_i F_m$ Let $B_1, B_2, \ldots B_k$ be <u>all</u> subformulas of F of size b. Then $T \uplus_{eb} \{F : \mu\} = (T \uplus_{pat} F) \uplus_{eb} B_1 : md_i(\mu) \uplus_{eb} \ldots \uplus_{eb} B_k : md_i(\mu)$.*

It is easy to notice that

Proposition 2 *(i) $(\forall b > 0) eb \geq pat$ (ii) $e1 \equiv pat$*

Elementary updates allow us to capture more information about the updating formula, but these updates still allow for the loss of information as is shown in the following example.

Example 9 *Let $T = \emptyset$ and $F = (A \wedge_{inc} B \wedge_{inc} C) : [0.4, 0.6]$ (A,B,C are ground atoms). Let $T' = T \uplus_{e3} F$. By definition above $T' = \{(A \wedge_{inc} B \wedge_{inc} C) : [0.4, 0.6], A : [0.4, 1], B : [0.4, 1], C : [0.4, 1]\}$. We notice that $T'[(A \wedge_{inc} B)] = [0.16, 1]$. However, it is clear that $F \models (A \wedge_{inc} B) : [0.4, 1]$.*

The following strategy is more complete than elementary b-cache updates, but is also more difficult to compute.

Definition 42 (Full b-cache update) *Let T be a b-cache, F be a basic formula. We define a full b-cache update strategy (denoted $\uplus_{fb}$) as follows:*

1. **Case 1.** *$|F| = 1$. (F is atomic). $T \uplus_{fb} \{F : \mu\} = T \uplus_{at} \{F : \mu\}$*
2. **Case 2.** *Let $F \equiv F_1 *_i \ldots *_i F_m$, $m \leq b$. Let $B_1, \ldots B_k$ be all the subformulas of F of size $< m$. We declare $T \uplus_{fb} \{F : \mu\} = T \uplus_{eb} F : \mu \uplus_{eb} B_1 : md_i(\mu) \uplus_{eb} \ldots \uplus_{eb} B_k : md_i(\mu)$.*
3. **Case 3.** *$|F| > b$. Let $F = F_1 *_i \ldots *_i F_m$ Let $B_1, B_2, \ldots B_k$ be all subformulas of F of size $\leq b$. Then $T \uplus_{fb} \{F : \mu\} = (T \uplus_{pat} F) \uplus_{eb} B_1 : md_i(\mu) \uplus_{eb} \ldots \uplus_{eb} B_k : md_i(\mu)$.*

The following result tells us that not only is the full b-cache update strategy more complete than the elementary b-cache update strategy, it is also maximally complete when b is taken to be the size of the Herbrand base of our underlying logical language L.

Proposition 3 *(i) $(\forall b > 0) fb \geq eb$ (ii) $f1 \equiv e1 \equiv pat$
(iii) Suppose $k = |B_L|$. Then: fk is maximally complete update strategy.*

As the reader may easily notice from the definitions, implementing atomic updates is easy, however, PAT is more efficient than the elementary b- cache strateges eb, which get less efficient as b gets larger – and finally, implementing the full b-cache strategies is hardest of all, with the efficiency of these updates degrading as b increases. This will become apparent from the examples shown in the next section.

6 Proof Procedure for HP-Programs with b-cache

In the previous section, we presented a query refutation procedure for hybrid probabilistic programs. We now modify that refutation procedure for the case of query resolution from an hp-program with b-cache.

Informally the desired resolution procedure works as follows. Initially we have query Q, program P, a b-cache update strategy u and our b-cache T is (initially) empty. On each resolution step, we select a basic formula $F : \mu$ from current query and perform a lookup for the probability range of this formula in our current b-cache. To do this we have to compute $T[F]$. Once $T[F]$ is computed we compare it to μ. In case $T[F] \subseteq \mu$ we consider current resolution step done. Otherwise, we use refutation procedure described above to perform one resolution step. If we decide that this resolution step resulted in proving new basic formula, we use b-cache update strategy u to update current b-cache with one or possible more new proven formulas.

Definition 43 *Let $Q \equiv \exists(F_1 : \mu_1 \wedge \ldots \wedge F_n : \mu_n)$ be an initial query to hp-program P. A b-cache supported initial query $\hat{Q}$ is defined as follows: Let $F_{i_1} : \mu_{i_1} \ldots F_{i_n} : \mu_{i_n}$ is an arbitrary permutation of $F_1 : \mu_1 \wedge \ldots \wedge F_n : \mu_n$. Then $\hat{Q} \equiv < (F_{i_1}\mu_{i_1}, \emptyset), \ldots, (F_{i_n} : \mu_{i_n}, \emptyset) >$. Any initial b-cache supported query is a b-cache supported query.*

It is clear from the definition above that one query to P of size n can generate $n!$ different b-cache supported queries.

Definition 44 *We difine a* <u>*b-cache supported resolvent*</u> *and a* <u>*b-cache update procedure*</u> *simulteneously. Let P be an hp-program, T - a b-cache and u - a b-cache update strategy. Let $\hat{Q} \equiv <(G_1 : \mu_1, S_1), \ldots, (G_m : \mu_m, S_m)>$, where for each $1 \leq i \leq m$, S_i is a set (possibly empty) of annotated basic formulas (not necesserily ground). Two cases have to be considered:*

1. *There exists $<\mu, \Theta> \in T[G_1]$, such that, $\mu \subseteq \mu_1$. Let $C \equiv G_1\Theta : \mu \longleftarrow$. Then*

$$\hat{Q}' \equiv <(G_2\Theta : \mu_2, S_2\Theta), \ldots, (G_m\Theta : \mu_m, S_m\Theta)>$$

is a b-cache supported resolvent *of $\hat{Q}$ and C.*
A b-cache update procedure ϕ_u for this case can be defined as follows: $\phi_u(\hat{Q}, T, C, \Theta) = T \uplus_u S_1\Theta$.

2. *There is* **no** *$<\mu', \Theta'> \in T[G_1]$, such, that $\mu' \subseteq \mu_1$. In this case, let $C \equiv G : \lambda \longleftarrow F_1 : \lambda_1 \wedge \ldots \wedge F_n : \lambda_n$, G_1 unifies with G via max-gu Θ and $\lambda \subseteq \mu_1$. Let $F_{i_1} : \lambda_{i_1} \ldots F_{i_n} : \lambda_{i_n}$ be any arbitrary permutation of $F_1 : \lambda_1 \wedge \ldots \wedge F_n : \lambda_n$.*
We define a b-cache supported resolvent of $\hat{Q}$,C and T to be:

$$\hat{Q}' \equiv <(F_{i_1}\Theta : \lambda_{i_1}, \emptyset), \ldots, (F_{i_n}\Theta : \lambda_{i_n}, S_1\Theta \cup \{G_1\Theta\}), (G_2\Theta : \mu_2, S_2\Theta), \ldots, (G_m\Theta : \mu_m, S_m\Theta)>$$

.

A b-cache update procedure for this case is defined as follows:

   (a) Body of C is **empty** .

$$\phi_u(\hat{Q}, T, C, \Theta) = T \uplus_u G_1\Theta : \lambda \uplus_u S_1\Theta$$

   (b) Body of C is **not** empty.

$$\phi_u(\hat{Q}, T, C, \Theta) = T$$

Definition 45 *Let P be an hp-program, Q - a query and u - a b-cache update strategy. A* <u>*b-cache supported refutation*</u> *of Q from P via HR_P is a* finite *sequence*

$$<\hat{Q}_1, C_1, \Theta_1, T_1> \ldots <\hat{Q}_r, C_r, \Theta_r, T_r>$$

, where

- *$\hat{Q}_1$ is b-cache supported initial version of Q.*
- *$T_1 = \emptyset$*
- *$\hat{Q}_r$ is empty.*
- *for each $1 \leq i \leq r$ either $P \vdash C_i$ or $T_i \vdash C_i$.*
- *for each $1 \leq i < r$, $\hat{Q}_{i+1}$ is a b-cache supported resolvent of $\hat{Q}_i$ and C_i with max-gu Θ_i.*
- *for each $1 \leq i < r$, $T_{i+1} = \phi_u(\hat{Q}_i, T_i, C_i, \Theta_i)$*

Example 10 (2-cache supported hp-refutation with elementary update strategy) Let us return to the hp-program shown in Figure 6 and the query considered there. We present below, a refutation using a 2-cache (i.e. $b = 2$) using the strategy $e2$, i.e. elementary 2-cache update. The reader will notice that using this strategy cuts the number of steps in the resolution by 3 steps, leading to an over 20% reduction in the length of a proof. Note that had we used a different update strategy, the reduction may have been different.

1. $Q_1 =< (a : [1,1], \emptyset), (e : [1,1], \emptyset) >$
   $T_1 = \emptyset; P \ni C_1 = a : [1,1] \longleftarrow (b \wedge_{ind} c \wedge_{ind} d) : [0.25,1] \wedge f : [0.5,0.9]$.

2. $Q_2 =< ((b \wedge_{ind} c \wedge_{ind} d) : [0.25,1], \emptyset), (f : [0.5,0.9], \{a : [1,1]\}), (e : [1,1], \emptyset) >$
   $T_2 = \emptyset; P \vdash C_2 = (b \wedge_{ind} c \wedge_{ind} d) : [0.3,1] \longleftarrow (c \wedge_{ind} d) : [0.3,1]$.

3. $Q_3 =< ((c \wedge_{ind} d) : [0.3,1], \{(b \wedge_{ind} c \wedge_{ind} d) : [0.3,1]\}), (f : [0.5,0.9], \{a : [1,1]\}), (e : [1,1], \emptyset) >$
   $T_3 = \emptyset; P \vdash C_3 = (c \wedge_{ind} d) : [0.3,1]) \longleftarrow$

4. $Q_4 =< (f : [0.5,0.9], \{a : [1,1]\}), (e : [1,1], \emptyset) >$
   $T_4 = (T_3 \uplus_{2e} (c \wedge_{ind} d) : [0.3,1])) \uplus_{2e} (b \wedge_{ind} c \wedge_{ind} d) : [0.3,1] = \{c : [0.3,1], d : [0.3,1], b : [0.3,1], (c \wedge_{ind} d) : [0.3,1], (b \wedge_{ind} c) : [0.3,1], (b \wedge_{ind} d) : [0.3,1]\}$
   $P \vdash C_4 = f : [0.7,0.9] \longleftarrow b : [1,1]$.

5. $Q_5 =< (b : [1,1], \{f : [0.7,0.9], a : [1,1]\}), (e : [1,1], \emptyset) >$
   $T_5 = T_4; T[b] = [0.3,1] \not\subseteq [1,1]$;
   $P \ni C_5 = b : [1,1] \longleftarrow (c \wedge_{ind} d) : [0.3,1]$.

6. $Q_6 =< ((c \wedge_{ind} d) : [0.3,1], \{b : [1,1], f : [0.7,0.9], a : [1,1]\}, (e : [1,1], \emptyset) >$
   $T_6 = T_5 = T_4; T_6[(c \wedge_{ind} d)] = [0.3,1] \subseteq [0.3,1]$

7. $Q_7 =< (e : [1,1], \emptyset) >$
   $T_7 = (T_6 \uplus_{2e} (c \wedge_{ind} d) : [0.3,1]) \uplus_{2e} \{b : [1,1], f : [0.7,0.9], a : [1,1]\} = \{c : [0.3,1], d : [0.3,1], b : [1,1], (c \wedge_{ind} d) : [0.3,1], (b \wedge_{ind} c) : [0.3,1], (b \wedge_{ind} d) : [0.3,1], f : [0.7,0.9], a : [1,1]\}$
   $P \ni C_7 = e : [1,1] \longleftarrow (b \wedge_{ind} c \wedge_{ind} d) : [0.25,1] \wedge f : [0.5,0.1]$.

8. $Q_8 =< ((b \wedge_{ind} c \wedge_{ind} d) : [0.25,1], \emptyset), (f : [0.5,0.1], \{e : [1,1]\}$
   $T_8 = T_7; T_8[(b \wedge_{ind} c \wedge_{ind} d)] = [0.3,1] \subseteq [0.25,1]$

9. $Q_9 =< (f : [0.5,0.1], \{e : [1,1]\}$
   $T_9 = T_8 = T_7; T_9[f] = [0.7,0.9] \subseteq [0.5,0.9]$

10. $Q_{10} = \Box$

The following two important results state that *irrespective* of which update strategy is used, b-cache supported hp-refutations are guaranteed to be sound and complete. (Completeness assumes that the program P is consistent). The proofs are straighforward, as we know that HR_P is sound and comlete, and the b-cache supported hp-refutation via HR_P is just its conservative extention.

Theorem 10 (Soundness of b-cache supported hp-refutation via HR_P).
Let P be an hp-program, Q be an initial query, and $\uplus$ be any update strategy. If there exists a b-cache supported refutation via HR_P of $Q \equiv \exists(F_1 : \mu_1 \wedge \ldots \wedge F_n : \mu_n)$ from P with the answer substitution Θ then $P \models \forall((F_1 : \mu_1 \wedge \ldots \wedge F_n : \mu_n)\Theta)$.

Theorem 11 (Completeness of b-cache supported hp-refutation).
Let P be a consistent hp-program, Q' be a query, and $\uplus$ be any update strategy. Then, if $P \models \exists(Q')$ then there exists a b-cache supported hp-refutation of Q' from P via HR_P.

7 Related Work and Conclusions

Logic knowledge bases have been extended to handle fuzzy modes of uncertainty since the early 70's with the advent of the MYCIN and Prospector systems [10]. Shapiro was one of the first to develop results in fuzzy logic programming [27]. Baldwin [1] was one of the first to introduce *evidential* logic programming and a language called FRIL. Van Emden [31] was the first to provide formal semantical foundations for logic programs that was later extended by Subrahmanian [28] and then completely generalized in a succession of papers by Blair and Subrahmanian[3], and Fitting[12], Ginsberg [13], and applied to databases by Kifer and Li [16] and Kifer and Subrahmanian [17]. *All the above works did not obey the laws of probability.*

The first works in the area of probabilistic logic programming were due to Ng and Subrahmanian who, in a series of papers [22, 24], developed techniques for probabilistic logic programming under the assumption of *ignorance*. Their work built upon earlier work on probabilistic logics due to Fagin, Halpern and and others [11] and Nilsson [25].

In contrast, Kiessling and his group [14, 30, 26] have developed a framework called DUCK for reasoning with uncertainty. They provide an elegant logical axiomatic theory for uncertain reasoning in the presence of rules, and using the *independence assumption*.

Perhaps the most significant related work is the elegant recent work of Lakshmanan's group [20, 19]. There are several differences between our approach, and theirs. First, the syntax is different. We associate probabilities with atoms, while [20] associates two intervals with any rule. Second, their work can be viewed as a *refinement* of [23] and [17]. In contrast, in this paper, we propose a generalization of [22]. Third, our semantics is different. For instance, even if none of the hybrid p-strategies in our paper are present, and we just have two facts $a : [1,1]$ and $a : [0,0]$ in P, our semantics would declare an inconsistency, while theirs would imply $a : [1,1]$, but would not be inconsistent. This seems quite counter-intuitive to us. They only allow atoms in the head and in the body, at most one conjunction (corresponding to one of our basic formulas) is allowed. In contrast, we allow arbitrary basic formulas in rule heads, and allow arbitrary conjunctions of basic formulas reflecting different probabilistic strategies in the rule body. Fourth, our semantics uses arbitrary and new definitions of conjunction and disjunction strategies in logic programs. Fifth, we have developed multiple alternative ways of processing queries to probabilistic logic programs, include the use of arbitrary update strategies in cache-based query processing. Lakshmanan and his colleagues complement our results with elegant query optimization results. Developing such results in the general setting of hp-programs remains a significant challenge, and will need to build upon the foundation laid by them in that arena.

There has been a substantial body of work on probabilistic extensions of relational databases, which we do not discuss here as their relation to logic programming is not immediate. For the sake of completeness, such works include[2, 5, 15, 18]. In particular, [18] proposes a set of axioms governing probabilistic strategies. This differs from our work here in that the axioms are different and there is no notion of a "decomposition" function associated with p-strategies.

In sum, our paper's goal was to provide a flexible probabilistic logic programming framework. Past approaches to logic programming with probabilities assumed that knowledge about all events in the real world represented by propositional symbols or predicate symbols took one single form – either we assumed ignorance of all dependences between such events (e.g. [22]) or we assumed independence (e.g. in most AI expert systems). In practice however, a probabilistic logic programming system must be flexible enough to allow the logic programmer to explicitly specify any domain specific knowledge he has about dependences (or lack thereof) between events. Our approach allows this, through the use of syntactic connectives that represent generalized conjunction/disjunction strategies. We have provided a formal model theoretic and fixpoint semantics for such hp-programs and shown that they are equivalent. We have further proposed three alternative execution paradigms for hp-programs.

In future work, we plan to build an hybrid probabilistic deductive database system that incorporates many of the ideas proposed in this paper. This system will be built on top of our ProbView[18] probabilistic relational database system. We hope to use this implementation, when complete, to experiment with different probabilistic query evaluation algorithms such as those described here, as well as probabilistic query optimization techniques that we hope to develop in the future.

8 Acknowledgements

This work was supported by the Army Research Office under Grants $DAAH-04-95-10174$, $DAAH-04-96-10297$, and $DAAH04-96-1-0398$, by the Army Research Laboratory under contract number $DAAL01-97-K0135$, and by an NSF Young Investigator award $IRI-93-57756$.

References

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

[2] D. Barbara, H. Garcia-Molina and D. Porter. (1992) *The Management of Probabilistic Data*, IEEE Trans. on Knowledge and Data Engineering, Vol. 4, pps 487–502.

[3] H. Blair and V.S. Subrahmanian. Paraconsistent Logic Programming, THEORETICAL COMPUTER SCIENCE, Vol. 68, pp 135–154, 1989.

[4] G. Boole. (1854) *The Laws of Thought*, Macmillan, London.

[5] R. Cavallo and M. Pittarelli. (1987) *The Theory of Probabilistic Databases*, Proc. VLDB 1987.

[6] W. Chen and D.S. Warren. (1992) *A Goal-Oriented Approach to Computing Well-Founded Semantics*, Proc. 1992 Intl. Conf. on Logic Programming (ed. K.R. Apt), MIT Press.

[7] A. Dekhtyar and V.S. Subrahmanian. (1997) *Hybrid Probabilistic Logic Programs*, Proc. 1997 Intl. Conf. on Logic Programming (ed. L. Naish), MIT Press.

[8] S. Dietrich and D.S. Warren. (1986) *Extension Tables: Memo Relations in Logic Programming*, SUNY Stonybrook Tech. Report 86/18.

[9] D. Dubois and H. Prade. Certainty and Uncertainty of Vague Knowledge and Generalized Dependencies in Fuzzy Databases. In *Proceedings International Fuzzy Engineering Symposium*, pp. 239–249, Yokohama, Japan, 1988.

[10] R. O. Duda, P. E. Hart and N. J. Nilsson. (1976) *Subjective Bayesian Methods for Rule-based Inference Systems*, Proceedings of National Computer Conference, pp 1075–1082.

[11] R. Fagin and J. Halpern. (1988) *Uncertainty, Belief and Probability*, in Proc. IJCAI-89, Morgan Kaufman.

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

[13] M. Ginsberg. (1988) *Multivalued Logics: A Uniform Approach to Reasoning in Artificial Intelligence*, Computational Intelligence, vol. 4, 1988, pp. 265–316.

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

[15] W. Kiessling, H. Thone and U. Guntzer. (1992) *Database Support for Problematic Knowledge*, Proc. EDBT-92, pps 421–436, Springer LNCS Vol. 580.

[16] M. Kifer and A. Li. (1988) *On the Semantics of Rule-Based Expert Systems with Uncertainty*, 2-nd Intl. Conf. on Database Theory, Springer Verlag LNCS 326, (eds. M. Gyssens, J. Paredaens, D. Van Gucht), Bruges, Belgium, pp. 102–117.

[17] M. Kifer and V. S. Subrahmanian. (1992) Theory of Generalized Annotated Logic Programming and its Applications, JOURNAL OF LOGIC PROGRAMMING, 12, 4, pps 335–368, 1992.

[18] V.S. Lakshmanan, N. Leone, R. Ross and V.S. Subrahmanian. ProbView: A Flexible Probabilistic Database System. ACM TRANSACTIONS ON DATABASE SYSTEMS, Vol. 22, Nr. 3, pps 419–469, Sep. 1997.

[19] V.S. Lakshmanan and F. Sadri. (1994) *Modeling Uncertainty in Deductive Databases*, Proc. Int. Conf. on Database Expert Systems and Applications, (DEXA'94), September 7-9, 1994, Athens, Greece, Lecture Notes in Computer Science, Vol. 856, Springer (1994), pp. 724-733.

[20] V.S. Lakshmanan and F. Sadri. (1994) *Probabilistic Deductive Databases*, Proc. Int. Logic Programming Symp., (ILPS'94), November 1994, Ithaca, NY, MIT Press.

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

[22] R. Ng and V.S. Subrahmanian. (1993) Probabilistic Logic Programming, INFORMATION AND COMPUTATION, 101, 2, pps 150–201, 1993.

[23] R. Ng and V.S. Subrahmanian. A Semantical Framework for Supporting Subjective and Conditional Probabilities in Deductive Databases, JOURNAL OF AUTOMATED REASONING, 10, 2, pps 191–235, 1993.

[24] R. Ng and V.S. Subrahmanian. (1995) Stable Semantics for Probabilistic Deductive Databases, INFORMATION AND COMPUTATION, 110, 1, pps 42-83.

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

[26] H. Schmidt, W. Kiessling, U. Guntzer and R. Bayer. (1987) *Combining Deduction by Uncertainty with the Power of Magic*, Proc. DOOD-89, pps 205–224, Kyoto, Japan.

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

[28] V.S. Subrahmanian. (1987) *On the Semantics of Quantitative Logic Programs*, Proc. 4th IEEE Symp. on Logic Programming, pps 173-182, Computer Society Press. Sep. 1987.

[29] H. Tamaki and T. Sato. (1986) *OLD Resolution with Tabulation*, Proc. 3rd Intl. Conf. on Logic Programming (ed. E. Shapiro), pps 84–98, Springer.

[30] H. Thone, W. Kiessling and U. Guntzer. (1995) *On Cautious Probabilistic Inference and Default Detachment*, Annals of Operations Research, 55, pps 195–224.

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