Proof Procedures for 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], it is only recently that probabilistic logic programming [20, 19, 22, 23, 24] has been studied. 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: if e_1, e_2 are *independent*, then $\mathbf{Prob}(e_1 \wedge e_2) = \mathbf{Prob}(e_1) \times \mathbf{Prob}(e_2)$, while 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: $[\max(0, \mathbf{Prob}(e_1) + \mathbf{Prob}(e_2) - 1), \min(\mathbf{Prob}(e_1), \mathbf{Prob}(e_2))]$ In fact, there is a wide range 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.*

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.

In [7], we proposed the concept of a *hybrid probabilistic program* (hp-program for short) that permits this and allows users to write hp-programmers that reflect different ways of computing probabilities,

*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

appropriate for different dependencies between events. First, we defined a general axiomatic notion of a *probabilistic strategy*. We showed that a number of well known probabilistic strategies are special cases of our definition. We then showed 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. Subsequently, we defined 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.*

In this paper, we build upon the above work and define a set of proof procedures for hybrid probabilistic programs, and show that these proof procedures all have some advantages/disadvantages. The resulting proof procedures are sound and complete.

2 Background

2.1 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 $[0,1]$. 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* $b \leq d$.

Definition 1 *A probabilistic strategy (p-strategy) is a pair of functions:* $i = \langle c, md \rangle$*, 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. $md : \mathcal{C}[0,1] \longrightarrow \mathcal{C}[0,1]$ *is called a probabilistic decomposition function or a maximal interval function.*

The function c above is a *composition* function that generates a new interval from two input intervals. There are no restrictions so far on the md function, but intuitively it will serve the following purpose: given a probability of a compound event, return "best estimate" for the probability of a part of this event.

Definition 2 Conjunctive and Disjunctive p-strategies

A *p-strategy* $< c, d >$ *is called* conjunctive (disjunctive) if it satisfies the following axioms:

Axiom	Conjunctive	Disjunctive
Bottomline	$c([a_1, b_1], [a_2, b_2]) \leq [min(a_1, a_2), min(b_1, b_2)]$	$c([a_1, b_1], [a_2, b_2]) \geq [max(a_1, a_2), max(b_1, b_2)]$
Identity	$c([a, b], [1, 1]) = [a, b]$	$c([a, b], [0, 0]) = [a, b]$
Annihilator	$c([a, b], [0, 0]) = [0, 0]$	$c([a, b], [1, 1]) = [1, 1]$
Max.Interval	$md([a, b]) = [a, 1]$	$md([a, b]) = [0, b]$

Intuitively, a composition function determines, given the probability ranges of two events, the probability range of their (either and- or or-composition). A max-interval function returns the best estimate for the probability of simple event given the probability of a compound event. For the discussion on why we specify max-interval functions as above see [7]. Examples of different p-strategies can also be found there.

2.2 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 3 *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 $\mathcal{CONJ} \cup \mathcal{DISJ}$ If $i \in \mathcal{CONJ}$ then a connective $\wedge_i$ is called an* i-annotated conjunction *If $i \in \mathcal{DISJ}$, 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 4 *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 $bf_{\mathcal{DISJ}} = \cup_{i \in \mathcal{DISJ}} bf_i(B_L)$.*

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 5 *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 $\mu \in \mathcal{C}[0, 1]$.*

Definition 6 *Let $B_0, B_1, \ldots, B_k$ be hybrid basic formulas. Let $\mu_0, \mu_1, \ldots, \mu_k \in \mathcal{C}[0, 1]$, 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.

Definition 7 *A hybrid probabilistic program (hp-program) over set $\mathcal{S}$ of p-strategies is a finite set of hp-clauses involving only connectives from $\mathcal{S}$.*

2.3 Model Theory for hp-programs

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

Definition 8 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).

For the purpose of defining probabilistic model theory fo hp-programs, hybrid basic formula functions will play the role of an "interpretation." The key inductive definition of satisfaction is given below.

Definition 9 Satisfaction. Let h be a 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** $(\forall\, 1 \leq j \leq n) h \models F_j : \mu_j$.
- $h \models F : \mu \longleftarrow F_1 : \mu_1 \wedge \ldots \wedge F_n : \mu_n$ **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 $h(F) \subseteq \mu$.

It is possible for a hybrid formula function h to 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 10 *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 1 *Let P be any hp-program. Then:*

1. *h is a model of P iff $T_P(h) \leq h$.*
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 $T_P^\omega(F) \subseteq \mu$.*

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.

3 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 procedure (Section 3.1), described in detail in [7] and [22] is based upon expanding the program P to a larger set of clauses and then resolving queries against that set.
- The second procedure (Section 3.2) describes a refutation of a query as a proof in a specially defined inference system.
- Finally, for the third procedure(Section 3.3) , we employ the tablulation methods. A notion of b-cache as a "storage" space for already computed answers to queries is introduced and a refutation of a query is now defined to accomodate for the support from b-cache.

Unlike classical resolution, when dealing with annotated conjunctions and disjunctions, unifiers may not be unique, as noted by Ng and Subrahmanian[22]. However, as shown there, it is possible to define a notion of maximally general unifier in such a manner that if two formulas are unifiabe they have a (not necessarily unique) maximally general unifier max-gu.

3.1 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. The solution described below is to allow resolution against clauses of an extended version P, called closure of P.

Definition 11 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.

Definition 12 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 1 *For every clause $C \in CL(P)$, $P \models C$.*

We now present a refutation procedure for query processing.

Definition 13 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 14 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) $\lambda\Theta \subseteq \mu_i\Theta$

Definition 15 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.

Definition 16 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 2 (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 $P \models \forall((F_1 : \mu_1 \wedge \ldots \wedge F_n : \mu_n)\Theta)$.

Theorem 3 (Completeness of hp-refutation).
Let P be an hp-program, and Q be an initial query. If there exists a substitution θ such that $P \models \forall(Q\theta)$ then there exists an hp-refutation $< Q_1, C_1, \Theta_1 > \ldots < Q_r, C_r, \Theta_r >$ of Q and a substitution γ such that $\theta = \Theta_1 \cdots \Theta_r \gamma$.

3.2 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 17 *Let P be an hp-program. We define the 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* *R-Decomposition*
   $$\frac{(F *_i G) : \mu \longleftarrow Body}{F : md_i(\mu) \longleftarrow Body} \qquad \frac{(F *_i G) : \mu \longleftarrow Body}{G : md_i(\mu) \longleftarrow Body}$$
   - *Clarification:*
   $$\frac{F_1 : \mu_1 \longleftarrow Body_1 \qquad F_2 : \mu_2 \longleftarrow Body_2}{F_1\Theta : \mu_1 \cap \mu_2 \longleftarrow Body_1 \wedge Body_2}$$
   if F_1 and F_2 are unifiable via max-gu Θ
   - *Exchange: 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* **HR_P-derivation** 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 the result 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 theorem tells us that the system of axioms and inference rules describing HR_P precisely captures the closure, $CL(P)$, of P.

Theorem 4 *(1)* (soundness of HR_P) *For each hp-clause C, if $P \vdash C$ then $C \in CL(P)$.*
(2)(completeness of HR_P) *For each hp-clause C, if $C \in CL(P)$ then $P \vdash C$.*

Definition 18 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 $1 \le i \le r$
- Q_{i+1} is an hp-resolvent of Q_i and C_i with max-gu Θ_i, for all $1 \le i < r$.

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 does not require the construction of a program closure.

Theorem 5 (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 6 (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.

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.

3.3 B-Cache

We are now ready to study efficient tabled query processing techniques for HPPs.

3.3.1 Definitions

Definition 19 *A <u>cache</u> is a finite set of annotated basic formulas. If b is an integer, a <u>bounded b-cache</u> 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 20 *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 $\{\langle\mu, \Theta\rangle\}$, 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 $\{\mu | \langle\mu, \Theta\rangle \in T[F]\}$ is a singleton set. Without loss of generality we will abuse notation in this case and write $T[F] = \mu$.

3.3.2 *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. 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".

Definition 21 *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 22 *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 2 *(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 $P \models T \uplus_f \{F : \mu\}$.*

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

Definition 23 *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.

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 24 *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** $(\forall w)(u \geq w)$.

4 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.

Definition 25 *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 26 *We difine a b-cache supported resolvent and a b-cache update procedure 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** *:* $\phi_u(\hat{Q}, T, C, \Theta) = T \uplus_u G_1\Theta : \lambda \uplus_u S\Theta$
   (b) Body of C is **not** empty*:* $\phi_u(\hat{Q}, T, C, \Theta) = T$

Definition 27 *Let P be an hp-program, Q - a query and u - a b-cache update strategy.*
A b-cache supported refutation 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 $1 \leq i < r$, $T_{i+1} = \phi_u(\hat{Q}_i, T_i, C_i, \Theta_i)$*

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 7 (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 8 (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.

5 Related Work and Conclusions

Though there has been extensive work on probabilistic reasoning with uncertainty in AI and in logic programming [2, 9, 11, 12, 13, 14, 15, 16, 20, 25, 27, 31], till [7], there was no logic programming language that allowed reasoning with multiple probabilistic combination strategies. However, the probabilities of compound events cannot be computed by a single formula, as different probabilistic strategies must be used to compute the probability of $(a \vee b)$ and $(a \wedge b)$ from the probabilities of a and b. Lakshmanan et. al.[18] were the first to address this problem in the context of relational databases.

In [7], we addressed this issue for the first time in logic programming, and introduced the concept of hybrid probabilistic programs. A version of resolution was provided there, but this version was not very efficient. In this paper, we have presented two new resolution based query processing methods – HR_P-refutations and cache-based HR_P-refutations. The latter allows table driven evaluation methods, and allows arbitrary update strategies to be used to update caches as resolutions proceed.

6 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.