Reasoning with Probabilities, Time And Space

Thesis Proposal
Alexander Dekhtyar

May 13, 1999

Abstract

Over the years *probability theory* has emerged as the leading way of dealing with uncertainty in AI and Databases. However, application of probabilistic methods to different frameworks has always been a problem as the two main goals of such an application — efficiency and correctness — are often at odds with each other. This is so because different dependencies between the events considered cause different formulae to be used in computing the probabilities of complex events. In this work we consider how probabilistic approaches to dealing with uncertainty can be applied to a variety of frameworks in the Database theory and Logic Programming. Our work is based on two generic probabilistic frameworks, Probabilistic Databases, developed previously by Lakshmanan, Leone, Ross and Subrahmanian ([36]), which first introduced different ways of computing probabilities of complex events and Hybrid Probabilistic Logic Programs, developed as a part of this proposal ([9]). While these frameworks provide a generic solution for reasoning with probabilistic information, these solutions may become inefficient for certain specific problems. E.g., reasoning about uncertainty in time is associated with a possibly *extremely large* probability space. The situation gets even worse when the uncertain reasoning is about the locations of objects in nDimensional space (for $n \geq 3$). In this work we plan to address these problems both in database and logic programming settings. Temporal Probabilistic databases and Temporal Probabilistic Logic Programs which we plan to study represent our proposed solution to the problem of reasoning with temporal probabilities. We also discuss how to extend these solutions to cover n-dimensional probabilistic reasoning in both settings.

Contents

1 Introduction:Probabilistic Reasoning in Logic Programming and Databases

There are a number of different ways to approach reasoning with uncertainty in logic programming and databases (see [55] for a survey). Frameworks based on fuzzy logic and multi-valued logics [FROM ALEX: **add appropriate citations**] have been known for years. Most such frameworks incorporate uncertain reasoning elegantly into the logic programming language or database design. However, there are some disadvantages to using non-probabilistic models of uncertain reasoning.

- Logics of reasoning about knowledge and belief are one of the basic ways to reason about uncertainty, which in these logics is in the incomplete information that the reasoning agent has about the state of the world. Reasoning about knowledge and belief based on modal logics provides only very general notions of uncertainty and is applicable only to the class of problems where the notions of *knowledge* and *belief* make sense. Uncertainty in such systems is not quantitative.

- When multivalued logics are used to simulate uncertain reasoning, each truth value in the logic will correspond to a certain *degree of uncertainty*. However, it may not be clear what exact meaning each degree of uncertainty has. The correspondence between events and their degrees of uncertainty may thus become purely subjective and dependent on who is evaluating the events.

- Fuzzy logic provides a somewhat more advanced way of assessing the uncertainty of events. But fuzzy logic possesses a number of properties which do not correspond to the properties of the events considered. E.g., if an event A has uncertainty value of 0.5, fuzzy logic will assign 0.5 also to be the uncertainty value of the event $\mathbf{not}A$. Then the uncertainty measure corresponding to the conjunction of the two events will be found as the minimum of the two values, i.e. 0.5. However, it is clear that both A and $\mathbf{not}A$ cannot occur.

These and other similar concerns about non-probabilistic uncertain reasoning lead researchers to develop more complex frameworks based on probability theory.

In 1854 G.Boole published his work *The Laws Of Thought* ([5]) in which he described, among other things the key concepts behind the idea of probabilistic inference. These ideas formed the foundations for most of the subsequent probabilistic logics.

T. Hailperin ([28]) was one of the first to develop such a very general logic for reasoning with probabilistic information. A very thorough study of probabilistic logics and their properties had been conducted by by Fagin, Halpern et al. ([21, 23, 29, 1]) The logics they proposed extended their modal logics of knowledge and belief ([22]), and it had been shown Abadi and Halpern ([1]) that they cannot be finitely axiomatized. This was due to the great expressive power of their logics. While interesting from theoretical point of view, this logic unfortunately could not serve as basis for realistic database and logic programming models.

Another version of probabilistic logic is due to Nilsson [45]. In his work he described the model theory for probabilistic inference and described a number of methods of computing the probabilities in his model. Nilsson also used possible worlds as a part of his model.

A direct precursor of the current work in logic programming was the work of R. Ng and V.S. Subrahmanian in [42, 43, 44] who introduced a Probabilistic Logic Programming language (PLP), based on General Annotated Programs (GAPs), ([32]) by M.Kifer and V.S. Subrahmanian. The language introduced became the base of the logic programming frameworks described in this proposal. We discuss it in more detail in Section 2.1.

Concurrently with the work of Ng and Subrahmanian, and with our current research, V.S. Lakshmanan and F. Sadri in [38] and later V.S. Lakshmanan and N. Shiri in [39] developed and advanced the theory of Probabilistic Deductive Databases. Their work addresses a very similar set of problems and provides an alternative approach to solving them.

The problem of managing uncertain information in databases arose at the same time as the problem of uncertain inference in logic-based systems. Most of the research had been directed towards probabilistic approaches. This can be explained by the fact that the uncertainty in most applications comes in the form of probabilities. In 1987, R. Cavallo and M. Pitarelli proposed a theory of probabilistic databases ([7]) which although quite restrictive (only projection and join operations have been described), became the basis for a number of of future frameworks. Another probabilistic model was developed by D. Barbara, H. Garcia-Molina and D. Porter in [3]. The key assumption in this model was that the probabilities of events can be always determined exactly. Together with that, their models assumed independence between the events in it.

D. Dey and S. Sarkar in [14] proposed a 1NF approach to handling probabilistic databases. This database model improves the model of Barbara et al., as it (i) supports having uncertainty about some objects but certain information about others, (ii) uses first normal form which is easy to understand and use, (iii) introduces new operations and (iv) removes some of the restrictions from previous approaches. However, they still use points to represent probabilities and because of that a number of operations in their relational algebra are rather counterintuitive. For example, if a relation r contains a data tuple t with probability attribute α and relation r' contains the same data tuple with probability attribute $\beta \leq \alpha$, relation $r - r'$ will contain data tuple t with probability attribute $\alpha - \beta$. While this makes difference operation very simple, it is not clear what probabilistic intuition lies behind it.

Dyreson and Snodgrass [20] were among the first who studied probabilities in temporal databases. Our work in that area correlates with [20] but is also different in many aspects. We included a special comparison between our two frameworks of probabilistic temporal databases in Section 8.

Last, but not least is the ProbView model of V.S. Lakshmanan, N. Leone, R. Ross and V.S. Subrahmanian described in [36]. In this model, most of the restrictions present in the previous models are removed. Events stored in the database are no longer considered to be independent - rather, the model provides a mechanism for specifying the exact dependence between the events in the database (or the lack of information about any such dependences). This leads to the use of intervals of probabilities, as under certain assumptions about the relationships between the events, the precise probabilities of compound events may not be obtained, even if the precise probabilities of the events that constitute them are known.

In our work on the extending probabilistic models into time and space, we use the ProbView approach as the starting point. Because of its importance to and influence on the content of the research described in this proposal, a short summary of the ProbView model is given in Section 2.2.

From here, this proposal is organized in the following way. Section 2 contains the descriptions of Probabilistic Logic Programs (Sec. 2.1) and ProbView probabilistic database model (Sec. 2.2). After that in Section 3 the Hybrid Probabilistic Logic Programming Language, created as a generalization of the PLP approach is described.

Section 4 then provides motivation for creating specific frameworks for temporal and spatial probabilistic reasoning in Logic Programming and Database settings. This section contains a discussion of the benefits of such approach and a number of motivating examples providing intuition about how such reasoning could be utilized.

The subsequent sections provide details on the particular frameworks developed and currently under development as parts of this proposal. Section 5 describes Temporal Probabilistic Databases; Section 6 describes Temporal Probabilistic Logic Programs and Section 7 provides information about the nDimensional Probabilistic Databases and Logic Programming.

We conclude by comparing the research outlined in this proposal to the related work in Section 8 and by giving a time frame for the remaining part of the research.

2 Background

In this section we briefly describe Probabilistic Logic Programs (PLPs) of Ng, Subrahmanian ([42, 43, 44]) and probabilistic database theory behind the ProbView ([36]). As mentioned above, our work in Logic Programming is an extension of the PLP framework and our work in the Database Theory owes much to the ProbView model.

2.1 Probabilistic Logic Programs

In [42], Ng and Subrahmanian proposed the following Probabilistic Logic Programming Language. They considered a function-free language L with Herbrand base B_L. Each element of B_L was thought to represent one particular *simple event*. *Compound events* could be constructed as conjunctions and disjunctions of simple events: $A_1 \wedge \ldots \wedge A_k$ and $A_1 \vee \ldots \vee A_k$. With each simple or compound event a probabilistic annotation of a form $\mu = [\alpha, \beta] \subseteq [0, 1]$ was associated. Formula $F : \mu$ represented the statement *"probability of the event represented by F lies in interval μ"*. With such formulas the rules of the form

$$A : \mu \longleftarrow F_1 : \mu_1 \wedge \ldots \wedge F_n : \mu_n.$$

where A was an atom were constructed. The rule could be read as *"Probability of simple event A lies in the interval μ **if** probability of event F_1 lies in the interval μ_1,... and probability of event F_n lies in the interval μ_n"*.

Later in [43, 44] Ng and Subrahmanian extended their language to allow

- Conjunctions and disjunctions of atoms to be the heads of the rules, such as

$$(a \wedge b) : [0.5, 0.7] \longleftarrow c : [0.2, 1] \wedge (e \wedge f) : [0.5, 1].$$

  and

- Variable annotations to be present in the rules such as in the example below:

$$a : [V, W] \longleftarrow (a \wedge b) : [V, 1] \wedge (a \vee c) : [0, W].$$

In the model theory for the Probabilistic Logic Programs (PLPs) described in their work, the authors chose to interpret the conjunctions and disjunctions of the events *under the assumption of **ignorance***. This means that *no information is assumed to be known about the relationship between the events*. Boole in [5] has shown that under such an assumption

$$\max(P(e_1)+P(e_2)-1,0) \leq P(e_1 \wedge e_2) \leq \min(P(e_1),P(e_2))$$

and

$$\max(P(e_1),P(e_2)) \leq P(e_1 \vee e_2) \leq \min(1,P(e_1)+P(e_2)).$$

Here $P(e)$ stands for the probability of event e. This fact also justifies the decision to represent the probabilities as intervals, rather than numbers (point probabilities). Indeed, even if all atomic events are assigned point probabilities, the probabilities of compound events cannot be computed exactly. And it is easy to see that point probabilities can be easily represented by intervals: if we know that the probability of an event represented by formula F is exactly $\alpha \in [0,1]$ we can represent it in PLP as a fact

$$F:[\alpha,\alpha] \longleftarrow .$$

Instead of providing a more formal description of model theory, fixpoint semantics and proof procedure for PLPs we will use a simple example below to illustrate the key ideas behind this framework.

Example 2.1 Let us look at the following simple Probabilistic Logic Program P.

a:$[0.5,0.6] \longleftarrow$ b$\wedge$c: $[0.6,0.8]$.
a:$[0.6,0.8] \longleftarrow$ b$\wedge$d: $[0.3,1]$.
a:$[0.8,1] \longleftarrow$ c$\wedge$e: $[0.9,1]$.
b:$[0.9,1] \longleftarrow$.
c:$[0.7,0.7] \longleftarrow$.
d:$[0.5,1] \longleftarrow$.
e:$[1,1] \longleftarrow$.

The program consists of three rules which define the knowledge about the probability of event a based on the knowledge of probability of the events b$\wedge$c, b$\wedge$d and c$\wedge$e respectively, and four facts which establish the known probability intervals for events b, c, d, e.

We now show how to determine the tightest probability interval for event a that can be successfully inferred from P. We do this in a number of steps:

1. First, we check which of the three rules for a will fire. For this we need to check which bodies of the rules hold and which do not.

   - For the first rule we need to check what is the best interval that can be successfully inferred about the event b$\wedge$c. There are no rules in P with this event as the head (such rules are not permitted by the language described in [42] but this restriction had been relaxed in [43, 44]). However, P contains a rule for b and a rule for c. These two rules have empty bodies, which means that their heads are always true. Thus we know that the probability of b according to P lies in the interval $[0.9,1]$ and the probability of c lies in the interval $[0.7,0.7]$.

     Now, we use the abovementioned formulas for computing the probability interval under the ignorance assumption. We get

     $$\max(0,0.9+0.7-1) \leq P(\mathsf{b} \wedge \mathsf{c}) \leq \min(0.7,1)$$

     i.e.

     $$0.6 \leq P(\mathsf{b} \wedge \mathsf{c}) \leq 0.7$$

or $P(\mathsf{b} \wedge \mathsf{c}) \in [0.6, 0.7]$.

But as $[0.6, 0.7] \subseteq [0.6, 0.8]$, it is now true that the probability of $\mathsf{b} \wedge \mathsf{c}$ lies in the interval $[0.6, 0.8]$ (this just weakens our result). Therefore the body of the first rule is true and this rule can fire.

Similar reasoning for the second and third rules leads us to the following two conclusions:

- For the second rule from the probability intervals for b ($[0.9, 1]$) and d ($[0.5, 1]$) we get $P(\mathsf{b} \wedge \mathsf{d}) \in [\max(0, 0.9 + 0.5 - 1), \min(1, 1)] = [0.4, 1]$. Since $[0.4, 1] \subseteq [0.3, 1]$, we conclude that the body of this rule is true.
- For the third rule, the probability intervals for c and e are $[0.7, 0.7]$ and $[1, 1]$ respectively. Then $P(\mathsf{c} \wedge \mathsf{e}) \in [\max(0, 0.7 + 1 - 1), \min(0.7, 1)] \in [0.7, 0.7]$. But as $[0.7, 0.7] \not\subseteq [0.9, 1]$, we cannot claim that the body of the third rule is true.

2. Now we know that the first and second rules of P can fire, as their bodies are true, while the third rule has body which is false and therefore will not fire. By firing the first rule we find that $P(\mathsf{a}) \in [0.5, 0.6]$. By firing the second rule we find that $P(\mathsf{a}) \in [0.6, 0.8]$. As both inclusions can be inferred from the program, we want them both to be true. This is possible if $P(\mathsf{a}) \in [0.5, 0.6] \cap [0.6, 0.8] = [0.6, 0.6]$. The latter interval will be the tightest probability intervals the program P allows to infer for a.

The example provides a general idea about the works of the fixpoint semantics and proof procedure for PLPs. It also highlights a few important features of the probabilistic logic programs, which we enumerated below.

- <u>Use of probability intervals</u> instead of point probabilities. This has been discussed above.
- <u>Use of ignorance assumption about the dependencies between events</u>. This too has already been discussed.
- <u>Different multiple derivations of the same formula.</u> In classic logic programming, **any** possible derivation of a formula from a program will suffice to establish its truth. In Probabilistic Logic Programming, as demonstrated by the example above, one needs to look for **all** possible derivations of the formula in order to find the tightest possible probability interval that can be inferred. In our example, firing only the first or only the second rule of the program would have given only partial solutions. In order to obtain the correct answer, we needed to consider two independent derivations of a and then combine them together.
- <u>Use of *intersection* to combine together the results of different derivations</u>. This approach is sometimes called "the expert system approach" because by taking the intersection of all the intervals derived for a formula from a program we are making an attempt to keep **all** the information at hand true. If such an intersection happens to be empty, we claim that our program contains *inconsistent information* about the formula in question. A classic example of an inconsistency is a program that contains two facts: $\mathsf{a} : [0, 0] \longleftarrow .$ and $\mathsf{a} : [1, 1] \longleftarrow .$ The first fact states that event a *does not occur*. The second fact states that event a *definitely occurs*. Clearly both statements cannot be true at the same time.

In the Hybrid Probabilistic Logic Programming language (Section 3), developed as a part of this work we free ourselves from the ignorance assumption about the dependencies between the events. However, most of the features of the PLP language described above will be preserved as the Hybrid Probabilistic Logic Programming framework is a generalization of the framework described in this section.

2.2 Probabilistic Databases (ProbView)

As mentioned in the Introduction, our research in the area of Probabilistic Databases originates from the work of V.S. Lakshmanan, N. Leone, R.Ross and V.S. Subrahmanian. In [36] these authors develop a new approach to storing and querying probabilistic data. The implementation of this approach – ProbView database system, is also described there. In this section we provide brief description of the theory developed in [36].

The key notion defined in [36] is that of a *probabilistic tuple*. Let $R = \{A_1, \ldots A_k\}$ be a relation scheme. A classic relational tuple over R would be a sequence of k values $(a_1, \ldots, a_k)$ such that $(\forall 1 \leq i \leq k)(a_i \in dom(A_i))$. A *probabilistic tuple* over R is defined as a sequence of k pairs $(\langle V_1, h_1\rangle, \ldots, \langle V_k, h_k\rangle)$ where each V_i is a subset of $dom(A_i)$ and h_i is a *probabilistic distribution function* from V_i to the set of subintervals of $[0, 1]$. In this definition the first element of each $\langle V_i, h_i\rangle$ pair represents the *set of all possible values* that the ith field of the tuple can take, while the second element of the pair represents the *distribution of probabilities* over this set of possible values. For reasons similar to the ones stated in previous section, intervals, rather than points, are used to represent probabilities in this framework.

A simple example demonstrates this definition.

Example 2.2 Consider a database of an image recognition system. Each image the system is dealing with is a photo of one person at some location and is stored in a unique file (e.g. IMAGE1.GIF). With each image the system associates a list of names of people who could be featured on the picture and a list of locations where the image could have been taken. Finally for each name/person and for each location, an interval $[a, b] \subseteq [0, 1]$ is given. This interval represents the probability that the person (location) appears on the image. A relation R which contains all this information may contain tuples that look as follows.

Filename	h_1	**Person**	h_2	**Location**	h_3
IMAGE1.GIF	[1,1]	*John*	[0.2, 0.4]	*London*	$[0.5, 1]$
		Paul	[0.4, 1]	*Paris*	$[0.3, 0.3]$
		George	[0.3,0.5]		
IMAGE2.GIF	[1,1]	*John*	[0.9, 1]	*London*	$[0.2, 0.4]$
				Rome	$[0.5, 1]$
IMAGE3.GIF	[1,1]	*Paul*	[0.4, 0.4]	*Hamburg*	$[1, 1]$
		Ringo	[0.1,0.1]		

To specify the meaning of such probabilistic tuples the authors introduce *annotated representation* of probabilistic tuples. Intuitively, an annotated representation of a p-tuple is a collection of classical relational tuples, each augmented with a probability interval [1] which "represents" the same data as the p-tuple in a simpler form. Members of annotated representations are called *annotated tuples*.

Each annotated tuple represents by itself one possible assignment of values to all fields of the p-tuple and the probability interval associated it this assignment. E.g., annotated tuple (IMAGE1.GIF, *John*, *London*, $0.1, 0.4$) states that the probability that file IMAGE1.GIF contains a picture of John made in London lies between 0.1 and 0.4.

Using linear programming the authors define the conversion of probabilistic tuples into collections of annotated tuples which represent them. Then, relational algebra is defined on annotated

[1]In [36] and in the implementation, these tuples also contained a special *path* field which would allow to distinguish between the annotated representations which came from different "sources". As this nuance does not affect the key ideas behind the framework we do not address this issue here.

relations.

A few important features of the ProbView framework had direct influence on the development of our Temporal Probabilistic and nDimensional Probabilistic databases. These features are described briefly below.

- Dualism of probabilistic and annotated relations. While the notions of probabilistic tuples and probabilistic relations had been presented, the relational algebra and implementation were developed solely for annotated representations. In fact in the paper, the authors present the problem of developing a relational algebra for probabilistic tuples as open.

  In our work on Temporal Probabilistic Databases we solved the problem of developing a relational algebra for the relations that consist of large aggregate tuples (although as it will be described in Section 5 our temporal probabilistic tuples are a somewhat scaled down version of the probabilistic tuples). We elected a similar approach to creating the nDimensional Probabilistic databases.

- Use of Probabilistic Strategies in Relational Algebra. A tuple in an annotated relation represents an event. When the cartesian product of two annotated relations is taken the resulting relation will contain tuples that represent (all possible) conjunctions of events from the original annotated relations. But as we know, the probability of the conjunction of two events depends on the relationship between these events. Therefore, in order to compute probability intervals in the result of a cartesian product an assumption about the relationship between the events (represented by annotated tuples) has to be made.

  In [36] this problem is addressed by introducing a notion of *generic conjunction* function. Generic conjunction functions take as arguments two intervals and return an interval. As it is expected that given the probability intervals of two events, generic conjunctions produce the probability interval of their conjunction, these functions must satisfy certain axioms, which guarantee the latter. Examples of generic conjunctions for different assumptions about the relationship between the events (ignorance, independence, positive correlation) are given.

  Similarly, while defining the analog of duplicate elimination operation (called compaction in the ProbView framework), one needs to deal with the following problem. If an annotated relation R contains two tuples (d, l, u) and (d, l', u'), where d is the relational part of the tuple and $[l, u] \neq [l', u']$, how does one replace these two tuples in R with just one. Clearly the problem is in computing the "new" probability interval. In this [36] follows the works of Lakshmanan and Sadri ([38]) which take a "disjunctive" approach to compacting the information. In this approach, the tuple in the compacted relation will have the probability interval defined as the probability *that at least one of the tuples in the original relation is true.* But as probability of the disjunction of two (or more) events depends on the relationship between the events, the authors describe *generic disjunction* functions which can be used to compute these probabilities under a variety assumptions.

  We notice here, however, that the compaction approach in [36] ("compaction by disjunction") is different from the "expert system" compaction approach (or in other words "compaction by intersection") adopted by Ng and Subrahmanian in [42] and demonstrated in Example 2.1. While the former approach computes the probability that *at least one* of the compacted tuples is true, the latter computes it for the case when *all* tuples to be compacted are true. In our Temporal Probabilistic framework we generalized the notion of compaction to be capable of implementing either approach (as well as many more), however, our "preferred" (and to a certain extent, default) compaction strategy is the one of intersection.

- Incomplete Distributions. Returning back to the Example 2.2 let us take a look at the third p-tuple there. It states that the probability that the person depicted in IMAGE1.GIF is Paul is exactly 40% and the probability that this person is Ringo is exactly 10%. Combined together these two possibilities are responsible only for 50% of the probability. This suggests that there is missing information about the person on IMAGE3.GIF, i.e., this could be an image of some other person, currently unknown to the database.

3 Hybrid Probabilistic Logic Programs

As it mentioned in Section 2.1, the Probabilistic Logic Programming framework had a number of restrictive features. The main such feature was the forced assumption of ignorance about the relationships between the events. While this assumption is certainly safe, and produces sound results, in the cases when the actual relationship between the events is known, or in the case when we are willing to make certain assumptions about such relationship Probabilistic Logic Programs cannot help us.

This disadvantage of the PLP framework prompted us to look for a better, more general probabilistic logic programming solution. As we wanted to incorporate different dependencies between the events in our reasoning, we named this framework "Hybrid Probabilistic Logic Programs" (HPLPs). Our primary set of goals was defined as follows:

- Define a hybrid probabilistic logic programming framework which would allow for the dependencies between different events to be taken into account. Define structures necessary to support such framework. Allow the use of variable annotations in the program and allow the non-atomic heads in rules.
- Construct model and fixpoint theory for hybrid probabilistic logic programs.
- Construct a sound and complete query resolution procedure for HPLPs.
- Construct an efficient (and also sound and complete) query resolution procedure for HPLP. The first query resolution procedure constructed was proven to be sound and complete, but it required an exponential pre-compilation step.
- Construct a query resolution procedure for HPLPs using tabulation techniques. Tabulation techniques allow us to keep track of the subqueries that have already been proven during the query resolution procedure. Storing such subqueries in a table and consulting the table before proceeding with the resolution of the next subquery can eliminate repeated resolutions and speed up the resolution process.
- Study of algorithms for and complexity of fixpoint computation, entailment and consistency problems for HPLPs.

Below is a short summary of the current progress. The material of all sections except for Section 3.5 comes from [9]. The material of Section 3.5 comes from [13].

3.1 Probabilistic Strategies

The material in this section comes from [9].

We start the description of the Hybrid Probabilistic Logic Programming by defining the key notion of the framework : ***probabilistic strategies***.

As mentioned above, for any pair events e_1, e_2, the probability of their conjunction (i.e. that both of them occur) and their disjunction (i.e. that at least one of them occurs) depends on the relationship between the events. When we know that e_1 and e_2 are independent of one another, the probabilities can be computed precisely according to well-known formulae:

$$P(e_1 \wedge e_2) = P(e_1) \cdot P(e_2)$$

and

$$P(e_1 \vee e_2) = P(e_1) + P(e_2) - P(e_1) \cdot P(e_2).$$

When *nothing* is known about the dependencies between e_1 and e_2, as mentioned in Section 2.1 the precise probability of both conjunction and disjunction cannot be computed, but intervals in which these probabilities lie can be established. We repeat here the bounds found by Boole in [5] :

$$\max(P(e_1) + P(e_2) - 1, 0) \leq P(e_1 \wedge e_2) \leq \min(P(e_1), P(e_2))$$

and

$$\max(P(e_1), P(e_2)) \leq P(e_1 \vee e_2) \leq \min(1, P(e_1) + P(e_2)).$$

As the examples below show, there may be other dependencies between the events that influence the computation of the probabilities of conjunction and disjunction.

Our first goal is to generalize the notion of the dependency between the events in such a manner that would make it convenient to compute the probabilities of the compound events given the probabilities of the simple events. A similar goal had already been achieved by Lakshmanan *et al.*in the probabilistic database model ([36]).

However in our case the generalization has to be somewhat more complicated due to the fact that together with computing the probability of compound events (conjunction or disjunction) based on probabilities of simple events, we may want to solve the reverse problem as well. That is, given a compound event and its probability, find the probabilities of the simple events. This latter computation is needed because we want to allow the heads of the rules in our programs to be compound events. The examples below will illustrate this point fully.

Our choice to be able to express any dependency between events or the lack of thereof, suggests that probabilities have to be represented as intervals throughout our framework. In what follows we will implicitly assume that all intervals mentioned in the text are subintervals of $[0,1]$, and we will denote by $\mathcal{C}[0,1]$ the set of all such subintervals. We will use the notation $[a,b] \leq [c,d]$ to represent the fact that $a \leq c$ and $b \leq d$.

Because of the need to solve both problems described above, we proposed the following definition of a ***probabilistic strategy***.

Definition 3.1 *A* ***probabilistic strategy (p-strategy)*** *is a pair of functions:* $\rho = \langle c, d \rangle$*, 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) Inclusion-Monotonicity : $c([a_1,b_1],[a_2,b_2]) \subseteq c([a_3,b_3],[a_2,b_2])$ if $[a_1,b_1] \subseteq [a_3,b_3]$

(d) Separation: *There exist two functions* $c^1, c^2 : [0,1]\times[0,1] \rightarrow [0,1]$ *such that* $c([a,b],[c,d]) = [c^1(a,c), c^2(b,d)]$.

2. $d : \mathcal{C}[0,1] \longrightarrow 2^{(\mathcal{C}[0,1]\times\mathcal{C}[0,1])}$ *is called a probabilistic decomposition function.*

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

Among all possible probabilistic strategies, we are interested in two special classes: strategies that compute probabilities of conjunctions and strategies that compute probabilities of disjunctions. We axiomatize these two classes as follows:

Definition 3.2 Conjunctive and Disjunctive p-strategies

- A *p-strategy* $\rho = \langle c, d\rangle$ *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* $\rho = \langle c, d\rangle$ *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 : $c([a,b],[1,1]) = [1,1]$

Let us examine these definitions in detail. Out of the pair of functions $\langle c, d\rangle$, the first one, *composition* represents the function that returns the probability interval of a compound event, given the probability intervals of the events that constitute it. The second function, *decomposition*, performs the reverse task. Given a probability of a compound event, it returns the set of all pairs of probability intervals for the simple events which could have generated this probability.

Now we can look at each axiom.

- Commutativity and Associativity. As c is intended to be a function that computes probability of a compound event, it must obey some very basic principles of probability theory. We know that for two events e_1 and e_2, $P(e_1 * e_2) = P(e_2 * e_1)$ where $* \in \{\vee, \wedge\}$. This explains the Commutativity axiom. Similarly, the probability of a combination of three or more events will be the same *regardless* of how it is computed, i.e., if c is to be a function computing probability, it must be associative.

  These two axioms allow us to extend the definition of c onto more than 2 arguments in a natural way, as commutativity and associativity together guarantee that no matter in what order the computation of a cascade of c on more than 2 arguments is performed, the result will always be the same. For notational reasons it is also convenient to extend c to one argument as well setting $c([a,b]) = [a,b]$.

- Inclusion Monotonicity. The basic intuition behind this axiom is rather simple. As we look at the probability interval associated with an event e we can say that *the tighter the interval is, the more precise our information about the probability of e becomes.* Indeed, we can always say that the probability of e lies in the interval $[0,1]$, but this is really a vacuous truth. On the other hand if we know that the probability of e lies in the interval $[a,a]$, then we really know that it is exactly a as such interval contains only one point. Thus the intuition behind the inclusion monotonicity principle can be termed as the idea *"Information Preservation"*. The axiom states that the tighter (i.e. the "better") our information about the probabilities of initial events is, the tighter will be the probability interval of the compound event.

- Separation. This axiom states that function c which operates from pairs of intervals onto intervals can be broken into two *independent* functions c^1 and c^2, one of which takes two lower bounds of the initial intervals and produces the lower bound the resulting interval and the other one performing the same operation on the upper bounds. The real meaning of this axiom is the fact the lower bound of probabilistic interval for a compound event must depend *solely* on the lower bounds of the intervals of the events that constitute it, and similarly its upper bound must depend *solely* on the upper bounds of the arguments of c. Because of this axiom our composition functions generalize the functions that compute point probabilities.

- Coherence. This is the only axiom that involves the decomposition function. Basically we do not want an arbitrary combination of two functions to be called a probabilistic strategy. The coherence axiom provides the test that uniquely defines the decomposition function for each composition function. This axiom states that a pair $([a,b],[c,d])$ is in the set $d([\alpha,\beta])$ iff *it must be that case that* $c([a,b],[c,d]) = [\alpha,\beta]$. This guarantees us that d really decomposes the probability of the compound event soundly and completely into the probabilities of the events that constitute it.

- The three axioms defining a conjunctive or disjunctive p-strategy represent the well-known properties of the probabilities of conjunctions and disjunctions.

  For *conjunctive p-strategies*, Bottomline postulates that the probability of the compound event cannot exceed the minimum of the probabilities of the events that constitute it; Identity states that the the probability of a combination of any event e with a 100% *certain event* will be equal to the probability of e, and Annihilator states that the probability of a combination of any event e with an *absolutely impossible* event will be 0%.

  The three axioms for *disjunctive p-strategies* express properties dual to the properties of conjunctive p-strategies.

In the rest of the paper we will consider only conjunctive and disjunctive p-strategies. The example below illustrates conjunctive and disjunctive probabilistic strategies for the assumption of independence between the events.

Example 3.1 • *The conjunctive p-strategy $inc = \langle c_{inc}, d_{inc}\rangle$ is given by:*

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

- *The disjunctive p-strategy $ind = \langle c_{ind}, d_{ind}\rangle$ is given by:*

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

$$d_{ind}([a,b]) \text{ contains } \langle [a_1,b_1],[a_2,b_2]\rangle \in \mathcal{C}[0,1]\times\mathcal{C}[0,1]$$
$$\textbf{\textit{iff}}\ a_1+a_2-a_1a_2 = a \ \textbf{ and } \ b_1+b_2-b_1b_2 = b$$

It is easy to see, that composition function performs its duty exactly as planned, as given two intervals μ_1 and μ_2, it returns the interval $\mu = c(\mu_1,\mu_2)$, which as it was discussed above satisfies our intuition about the probabilities.

The situation is different when we look at the problem of finding the probabilities of simple events given the probability of a compound event they constitute. We would like to be able, given an interval representing the probability of a compound event, to obtain another interval which would be the probability of a simple event that is included in the compound event. However, our decomposition function d returns a *set of pairs of intervals*. This is so because there are many ways to set two probability intervals in such a way that their composition will give the same interval as a result. In general decomposition function is unable to select one pair of intervals out of the set it produces. So, we need to come up with a way to express the possible intervals for the simple events based on the entire content of the result of decomposition.

We do it in the following manner. Consider the set $d([a,b])$ for some interval $[a,b]$. We notice that as our composition function c is commutative, any interval μ that appears in the first place in some pair $\langle\mu,\mu'\rangle \in d([a,b])$ will also appear on the second place as the pair $\langle\mu',\mu\rangle$ must also be in $d([a,b])$. Now we concentrate on the set of intervals that appear in $d([a,b])$. In this set we find the smallest lower bound of all intervals and largest upper bound of all intervals. Clearly, the probability of a simple event that became a part of our compound event should be somewhere between these two numbers. We call this interval a *max-interval*. The formal definition goes as follows:

Definition 3.3 *Let $\rho = \langle c_\rho, d_\rho\rangle$ be a probabilistic strategy. As $\pi D_\rho([a,b])$ we will denote the set $\{\mu \subseteq [0,1] | (\exists\mu' \subseteq [0,1])(\langle\mu,\mu'\rangle \in d_\rho([a,b])\}$.*

We define the ***max-interval*** *of $d_\rho([a,b])$, denoted $md_\rho([a,b])$ as*

$$md_\rho([a,b]) = [\min_{[c,d]\in\pi D_\rho([a,b])}(c), \max_{[c,d]\in\pi D_\rho([a,b])}(d)]$$

As defined above, it may seem that finding md_ρ intervals is a hard problem. Indeed, in [43], in order to accommodate for the non-atomic heads of rules, Ng and Subrahmanian proposed a method of computing what essentially equals max-intervals for conjunctive and disjunctive p-strategies of ignorance that involved solving a linear program.

However as it turns out, computing the max-interval is a very simple task.

Theorem 1 *Suppose $\rho = \langle c,d\rangle$ is any* conjunctive *coherent p-strategy and $\rho' = \langle c',d'\rangle$ is any* disjunctive *coherent p-strategy. Then:*

1. $(\forall[a,b]\in\mathcal{C}[0,1])(md_\rho([a,b]) = [a,1])$.
2. $(\forall[a,b]\in\mathcal{C}[0,1])(md_{\rho'}([a,b]) = [0,b])$.

We explain why this is true informally on the following example.

Example 3.2 Consider conjunctive p-strategy for independence *inc* (see Example 3.1). Suppose we know that the probability of the conjunction of two events e_1 and e_2 under the assumption of independence lies in the interval $[a, b]$. Our goal is to find the interval in which the probability of each of the two simple events lies. We are looking at pairs of intervals $[a_1, b_1]$ and $[a_2, b_2]$ such that $a_1 a_2 = a$ and $b_1 b_2 = b$. Clearly we are interested in minimal possible values for a_1 and a_2 and maximal possible values of b_1 and b_2 such that the above equalities hold. It is easy to notice that both a_1 and a_2 can go as low as a while both b_1 and b_2 can reach 1 (not together). Just set $[a_1, b_1] = [a, b]$ and $[a_2, b_2] = [1, 1]$ or vice versa. So, 1 is the maximum number b_1 and b_2 can reach. Now we note that in order for a product of two numbers less than or equal to 1 to be equal to a number a, neither number can be less than a. This makes a the minimum a_1 and a_2 can reach. This suggests that the probability that each of the events e_1 and e_2 holds lies between a and 1. This interval is what will be returned by the md_{inc} function.

In what follows we will abuse notation and specify probabilistic strategies as pairs of *composition* and *max-interval* functions. There is no loss of generality here as the axiom of coherence guarantees that c uniquely determines d. However, since the decomposition function can be excluded from the actual computation of the max-interval, there is no need for it to be explicitly considered further. In fact, as max-interval functions depend only on the type of the p-strategy (conjunctive or disjunctive) in the examples below we specify only the composition functions.

Example 3.3

- p-strategies for ignorance assumption
  Conjunctive: (igc)
  $c_{igc}([a, b], [c, d]) = [\max(0, a + c - 1), \min(b, d)]$.
  Disjunctive: (igd)
  $c_{igd}([a, b], [c, d]) = [\max(a, c), \min(1, b + d)]$.
- p-strategies for positive correlation assumption
  Conjunctive:(pcc)
  $c_{pcc}([a, b], [c, d]) = [\min(a, c), \min(b, d)]$.
  Disjunctive:(pcd)
  $c_{pcd}([a, b], [c, d]) = [\max(a, c), \max(b, d)]$.
- p-strategy for negative correlation assumption
  Disjunctive: (ncd)
  $c_{ncd}([a, b], [c, d]) = [\min(a + c, 1), \min(b + d, 1)]$

Finally, we specify one more property of p-strategies.

Definition 3.4 *A composition function $c = [c^1, c^2]$ is called* continuous **iff** *both c^1 and c^2 are continuous in both their arguments. Similarly, a p-strategy $\rho = \langle c, d \rangle$ is continuous* **iff** *c is continuous.*

As it is easy to see that in all the examples of p-strategies that we have provided above the composition functions are continuous.

3.2 Syntax and Semantics of HPLPs

We can now define the syntax and semantics of Hybrid Probabilistic Logic Programs. These definitions fisrt appeared in [9].

3.2.1 Syntax

Let L be a language generated by finitely many constant and predicate symbols. Let B_L denote the set of constant symbols (atoms) in L. We assume that L has no ordinary function symbols, but it may contain *annotation function* symbols for a fixed family of functions (these symbols are interpreted below).

In hybrid probabilistic programs, we assume the existence of an arbitrary, but fixed set of conjunctive and disjunctive p-strategies $\mathcal{S}$. Each conjunctive p-strategy ρ has an associated conjunction operator $\wedge_\rho$ and each disjunctive p-strategy ρ' has an associated disjunction operator $\vee_{\rho'}$.

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 3.5 *Let ρ be a* conjunctive p-strategy, *ρ' be a* disjunctive p-strategy *and $A_1, \ldots, A_k$ be distinct atoms. Then $A_1 \wedge_\rho \ldots \wedge_\rho A_k$ and $A_1 \vee_{\rho'} A_2 \ldots \vee_{\rho'} A_k$ are called* hybrid basic formulas.

Suppose $bf_\rho(B_L)$ denotes the set of all ground hybrid basic formulas for the $\vee_\rho$ and $\wedge_\rho$ connectives. Let $bf_{\mathcal{S}}(B_L) = \cup_{\rho \in \mathcal{S}} bf_\rho(B_L)$.

We define the notion of *annotations* inductively: (1) Any real number or variable over real numbers is an annotation term. (2) If f is an interpreted function over the reals of arity k and $t_1, \ldots, t_k$ are annotation terms, then $f(t_1, \ldots, t_k)$ is an annotation term. An *annotation* is pair $[at_1, at_2]$ where at_1, at_2 are annotation terms. Thus, for instance, $[0.5, 0.6]$, $[0..5, \frac{V+1}{2}]$ are both annotations.

Definition 3.6 *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.*

Definition 3.7 *Let $B_0, B_1, \ldots, B_k$ be hybrid basic formulas. Let $\mu_0, \mu_1, \ldots, \mu_k$ be annotations. A hybrid probabilistic clause (hp-clause) is a construction of the form: $B_0 : \mu_0 \leftarrow B_1 : \mu_1 \wedge \ldots \wedge B_k : \mu_k$.*

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

An hp-program is *ground* iff its every clause is ground, i.e. all its clauses do not contain neither variables nor variable annotations.

Example 3.4 STOCK PROGRAM

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)):$[0.3, 1]$.
buy-stock(C):$[0.7, 1]$ $\longleftarrow$ (price-drop(C) $\wedge_{inc}$ stable(C)):$[0.3, 1]$.
sell-stock(C):$[0.5, 1]$ $\longleftarrow$ (price-drop(C) $\wedge_{inc}$ unstable(C)):$[0.4, 1]$ $\wedge$ have-stock(C):$[1, 1]$.
stable(c):$[0.8, 1]$ $\longleftarrow$.
strike(c):$[0.4, 0.5]$ $\longleftarrow$.
unstable(C):$[V1, V2]$ $\longleftarrow$ 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 the 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 "lowest common denominator" in terms of the 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 the premises of the company is over 30%, then the probability that the stock of the company will drop is *at least* 50%. If we 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 company 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 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 extend 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 $1-V1$.

3.2.2 Model Theory

Now we can proceed to define the model theory for the HPLPs. First we define *hybrid formula functions*, which will serve as interpretations in our semantics.

Recall that in our language L we have constant, predicate and variable symbols and B_L denotes the Herbrand base of L. An *atomic formula function* is a function $f : B_L \longrightarrow \mathcal{C}[0,1]$. Intuitively, an atomic formula function assigns probabilistic intervals to all ground atoms of our language.

We want to extend atomic formula functions in such a way that it allows us to specify the probability interval for any formula, atomic or not. The definition of an atomic formula function first appeared in [42]. In describing the model theory for PLPs, Ng and Subrahmanian then define *formula functions* which are just the extensions of atomic formula functions to non-atomic formulas. For PLPs of [42] there is a one-to-one correspondence between atomic formula functions and formula functions, i.e. atomic function uniquely defines its extension onto non-atomic formulas.

As it turns out this is not the case for HPLPs. In our model theory atomic formula functions will not uniquely define their non-atomic extensions.

Before defining *hybrid formula functions*, which will play the roles of interpretations in our model theory we introduce the following notation: let $F = F_1 *_\rho \ldots *_\rho F_n$, $G = G_1 *_\rho \ldots *_\rho G_k$, $H = H_1 *_\rho \ldots *_\rho H_m$ where $* \in \{\wedge, \vee\}$. We write $G \oplus_\rho H = F$ (or $G \oplus H$ if the p-strategy ρ is irrelevant) **iff:** (i) $\{G_1, \ldots, G_k\} \cup \{H_1, \ldots, H_m\} = \{F_1, \ldots, F_n\}$, (ii) $\{G_1, \ldots, G_k\} \cap \{H_1, \ldots, H_m\} = \emptyset$ and (iii) $k > 0$ and $m > 0$.

Definition 3.9 *A* hybrid formula function *is a function $h_f : bf_{\mathcal{S}}(B_L) \longrightarrow \mathcal{C}[0,1]$ which satisfies the following properties:*

1. Commutativity. *If $F = G_1 \oplus_\rho G_2$ then $h(F) = h(G_1 *_\rho G_2)$.*
2. Composition. *If $F = G_1 \oplus_\rho G_2$ then $h(F) \subseteq c_\rho(h(G_1), h(G_2))$.*
3. Decomposition. *For any basic formula F, $h(F) \subseteq md_\rho(h(F *_\rho G))$ for all $\rho \in \mathcal{S}$ and $G \in bf_{\mathcal{S}}(B_L)$.*

If h is a hybrid formula function and f is an atomic formula function, we say that h is *based on* f iff **iff** $(\forall A \in B_L)(f(A) = h(A))$.

Let us now look at the axioms that define hybrid formula functions.

First, remember that we want to interpret $h(F)$ as a probability interval which h associates with F. The axiom of Commutativity says that h has to be "honest" and assign the same probability *to all possible permutations of the same formula*. So, e.g., if B_L contains three atoms a, b and c, any hybrid formula function must assign the same intervals to $a \wedge_{inc} b \wedge_{inc} c$, $a \wedge_{inc} c \wedge_{inc} b$ as well as to all the other possible permutations.

The axiom of Composition states that the probability of a compound event has to be bounded by the probability given by an appropriate composition function. Continuing with the same example, any hybrid formula function h must satisfy the following condition: $h(a \wedge_{inc} b) \subseteq c_{inc}(h(a), h(b))$.

Finally, the axiom of Decomposition provides "feedback" from more complicated events to the simpler ones. Using the same example, any hybrid formula function h must satisfy the following property: $h(a) \subseteq md_{inc} h(a \wedge_{inc} b)$.

We can now specify the satisfaction.

Definition 3.10 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** $(\forall 1 \leq j \leq n) h \models F_j : \mu_j$.
- $h \models F : \mu \longleftarrow F_1 : \mu_1 \wedge \ldots \wedge F_n : \mu_n$ **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 $h(F) \subseteq \mu$.

3.2.3 Fixpoint Semantics

In order to define a fixpoint semantics, we first need to establish some properties of hybrid formula functions. First, some notation. A hybrid formula function is called *fully defined* iff $(\forall F \in bf_{\mathcal{S}}(B_L))h(F) \neq \emptyset$. If h and g are two hybrid formula functions, we will write $h \leq g$ to represent the fact that $(\forall F \in bf_{\mathcal{S}}(B_L))(h(F) \supseteq g(F))$.

In the ordering that we gave, the "greater" functions are the ones that assign tighter intervals to basic formulas. Now we are ready to state the result which will allow us to proceed with the design of the fixpoint procedure.

Theorem 2 *Let $\mathcal{S}$ contain* only continuous p-strategies. *Let $H = h_1, h_2, \ldots$ be an infinite sequence of* fully defined *hybrid formula functions over $bf_{\mathcal{S}}(B_L)$, such that $h_i \leq h_{i+1}$ (we can call this* an ascending sequence*). H has a* **least upper bound**, *i.e., there exists such hybrid formula function h^* such that $(\forall i)(h_i \leq h^*)$ and for any other function h which is an upper bound of H, $h^* \leq h$.*

The proof of this theorem uses the Separation axiom for p-strategies and the fact that all p-strategies considered are continuous. [2]

Now we are ready to state the fixpoint procedure. With each hp-program P we associate two operators S_P and T_P. T_P will be our immediate consequence operator, while S_P will be an intermediate operator.

Definition 3.11 *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 $M = \emptyset$ $S_P(h)(F) = [0, 1]$.*

Definition 3.12 *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*

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

[2] The proof consists of two parts: (1) construction of h^* and proof that it is an lub of our sequence and (2) proof that h^* is indeed a *hybrid formula function*. Proving the first part is easy, it is the second part that requires certain effort and the Separation and continuity of the p-strategies.

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

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

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

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

The operator S_P is very simple. Given a hybrid formula function h 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 $F_j : \mu_j$ in the body, $h(F_j) \subseteq \mu_j$, i.e. h says that F_j's probability does in fact lie within the interval μ_j. (ii) It then takes the intersection of the intervals associated with the heads of all rules identified in the preceding step (being true to our manifested allegiance to the "expert system approach").

The intuition underlying the T_P operator is as follows: (i) Consider an atomic formula F: if $S_P(h)(F) = \emptyset$, then we know than an inconsistency 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 head of the rule must be of the form $(F *_\rho G)$ where $*$ is either $\wedge$ or $\vee$. As the rule's body is satisfied by h, it means that the head of this rule, viz. $(F *_\rho G)$ has probability in the interval μ. The rule in question thus allows us to conclude that F's probability ranges anywhere in $md_\rho(\mu)$ which is the "maximal interval" associated with F w.r.t. the connective $*_\rho$. 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. The first source taken care of by the S_P operator are the rules with F as their head. The second source consists of information that can be inductively obtained by computing T_P for every pair G, H of formulas such that $G \oplus_\rho H = F$ (notice that we require both G and H to be non-empty), and using c_ρ to combine these values. Finally, some heads of the 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_ρ function. Combining (intersecting) the ranges obtained from all three sources we obtain the final value of T_P operator.

The following examples illustrate the works of the S_P and T_P operators.

Example 3.5 Consider our stock example. Let h assign the following values to the atoms:

$h(\text{ch-sells-stock(c)}) = [0.8, 0.8]$

$h(\text{ch-retires(c)}) = [0.1, 0.1]$

$h(\text{strike(c)}) = [0.4, 0.5]$

$h(\text{price-drop(c)}) = [0.7, 0.9]$

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

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

Computing S_P.

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((ch-sells-stock(c) $\vee_{igd}$ ch-retires(c))) $= c_{igd}$(ch-sells-stock(c),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, $S_P(h)$(price-drop(c)) $= [0.4, 0.9] \cap [0.5, 1] = [0.5, 0.9]$.

Computing T_P.

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)) $= [0.5, 0.9]$. $T_P(h)$(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)):$[0.3, 1]$.

As we know, 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 turns out that both operators are defined to be monotonic.

Theorem 3 *S_P and T_P operators are both monotonic.*

We define iterations of T_P as follows:

Definition 3.13 *1. $T_P^0 = \perp$ where $\perp$ is the function that assigns $[0, 1]$ to all ground formulas $F \in bf_{\mathcal{S}}(B_L)$.*

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

3. $T_P^\gamma = lub(\{T_P^\alpha | \alpha < \gamma\})$, where γ is limit ordinal. [3]

[3] The monotonicity of T_P operator means that $\{T_P^\alpha | \alpha < \gamma\}$ is an ascending sequence as defined by Theorem 2. Therefore, according to the conclusion of this theorem there indeed exists the lowest upper bound of this sequence.

Definition 3.14 *A hybrid probabilistic program P is said to* <u>satisfy the fixpoint reachability condition</u> iff

$$(\forall F \in bf_{\mathcal{S}}(L))(\exists n < \omega)(lfp(T_P)(F) = T_P^n(F)).$$

Theorem 4 *Let P be an hp-program with only* ***constant annotations*** *in it. Then P satisfies the fixpoint reachability condition.*

This, however, is not true for programs with variable annotations as shown in the following example.

Example 3.6 *(borrowed from [42])* Consider the program
$A : [0, V/2] \longleftarrow A : [0, V]$
$B : [0, 0] \longleftarrow A : [0, 0]$
The second rule of the program states that if the probability of A is known to be $[0, 0]$ then the probability of B is also $[0, 0]$. The first rule of the program states that if we know that the probability of A lies between 0 and some V, we should conclude that the probability of A lies in fact in the bottom half ($[0, V/2]$) of the interval $[0, V]$.

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. A T_P^ω assigns A the intersection of all T_P^l, $l < \omega$, it will assign $[0, 0]$ interval to A. Then $T_P^{\omega+1}$ will finally assign $[0, 0]$ to B.

Now we state our main result about the relationship between model theory and fixpoint semantics.

Theorem 5 *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 $lfp(T_P)(F) \subseteq \mu$.*

3.3 Proof Procedures for HPLPs

The material of this section is from [9].

As mentioned in Section 2.1 one of the key features of Probabilistic Logic Programs (which we inherited in our framework) is the fact that during resolution, more than one resolution path needs to be explored in general in order to find the correct answer. The reason for that is the fact that rather than deducing the truth or falsehood of a formula from a probabilistic program we are deducing the probability interval associated with it. And it is quite possible that a program P has two or more ways of of deducing two or more probabilistic intervals for a formula. Until *all* possible deductions of a formula from a program have been examined we cannot really claim to know if we found the best solution (although in order to answer *some* questions it may suffice to check less than all such possibilities).

Consider the following example.

Example 3.7 Let program P consist of two facts:
$a : [0.5, 0.8] \longleftarrow .$
$a : [0.4, 0.6] \longleftarrow .$

Suppose we know beforehand that this program is consistent. If asked "Is it true that the probability of a according to P lies in the interval $[0.5, 0.9]$?" we can produce the affirmative answer only by verifying the first fact of the program as it states that the probability of a lies in the interval $[0.5, 0.8] \subseteq [0.5, 0.9]$.

However if asked "Is it true that the probability of a lies in the interval $[0.45, 0.7]$ according to P ?" examining neither of the two facts alone will produce an affirmative answer as $[0.5, 0.8] \not\subseteq [0.45, 0.7]$ and $[0.4, 0.6] \not\subseteq [0.45, 0.7]$. But if we examine *both* of the facts together we will come to a conclusion that according to P the probability of a lies in the interval $[0.5, 0.8] \cap [0.4, 0.6] = [0.5, 0.6]$. And as $[0.5, 0.6] \subseteq [0.45, 0.7]$ we will be able to answer "yes" to the query above.

Before proceeding with the description of our proof procedures we briefly describe unification in HPLPs.

If $*$ is either $\wedge$ or $\vee$, then we say that two formulas $A_1 *_\rho \ldots *_\rho A_k$ and $B_1 *_\rho \ldots *_\rho B_m$ are unifiable iff there is a substitution θ such that $\{A_1\theta, \ldots, A_k\theta\} = \{B_1\theta, \ldots, B_m\theta\}$. This notion of unification was introduced in [42] and it was proved that though most general unifiers are not necessarily unique, a corresponding notion of a *maximally* general unifier (max-gu) exists.

The first procedure follows the footsteps of Ng and Subrahmanian [42]. The procedure, called hp-resolution strives to imitate the computation of the fixpoint of the program. Because of it, this procedure will be very inefficient as it involves a giant pre-computation step on which we will replace program P with the program that contains all possible rules that can be derived from P (We will call such program a *closure of* P). Although inefficient, this procedure is provably sound and complete, and can be used as a benchmark against which we check the soundness and completeness results for other proof procedures (instead of proving a full-blown soundness and completeness theorems for every other proof procedure, it will suffice to show that the results of each such procedure will coincide with the results of hp-resolution, which is a much shorter proof). The closure is built as follows:

Definition 3.15 Let P be an hp-program. Then $\mathrm{CL}(P)$ (closure of P) is defined as follows

- $\mathrm{CL}^0(P) = REDUN(P) = P \cup \{A : [0, 1] \longleftarrow | A \in B_L\}$.
- 1. For each pair of clauses $F_1 : \mu_1 \longleftarrow Body_1$ and $F_2 : \mu_2 \longleftarrow Body_2 \in \mathrm{CL}^j(P)$, such that their heads F_1 and F_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 $F_1 *_\rho F_2 : \mu \longleftarrow Body \in \mathrm{CL}^j(P)$ add the following two clauses to $\mathrm{CL}^{j+1}(P)$:
     - $F_1 : md_\rho(\mu) \longleftarrow Body$
     - $F_2 : md_\rho(\mu) \longleftarrow Body$
  3. For each two clauses $(A_1 *_\rho \ldots *_\rho A_k) : \mu_1 \longleftarrow Body_1$ and $(B_1 *_\rho \ldots *_\rho B_l) : \mu_1 \longleftarrow Body_2 \in \mathrm{CL}^j(P)$, $k > 1, l \geq 1$, add the clause
  $$(A_1 *_\rho \ldots *_\rho A_k *_\rho B_1 *_\rho \ldots *_\rho B_l) : c_\rho(\mu_1, \mu_2) \longleftarrow Body_1 \wedge Body_2$$
  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

$$(A *_\rho B) : c_\rho(\mu_1, \mu_2) \longleftarrow Body_1 \wedge Body_2$$

for each $\rho \in \mathcal{S}$ to $\mathrm{CL}^{j+1}(P)$.

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

We show that all rules in $\mathrm{CL}(P)$ are indeed the consequences of P:

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

Definition 3.16 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.

An hp-resolution procedure of query Q will do the following: on each step of the resolution one of the conjuncts $F_i : \mu_i$ of Q will be resolved against some rule C in $\mathrm{CL}(P)$. The body of C (with the appropriate substitution) will then replace $F_i : \mu_i$ in the query and the resolution of the new query will continue onto the next step. This will go on until the query becomes empty (successful resolution) or until on some step no conjunct of the current query can be resolved against any rule from $\mathrm{CL}(P)$ (failed resolution). Here, a formula $F : \mu$ is said to be resolvable with a rule $G : \mu' \longleftarrow Body$ via a substitution Θ iff $F : \mu$ and $G : \mu'$ are unifiable via max-gu Θ and $\mu\Theta \supseteq \mu'\Theta$.

The soundness and completeness results for hp-resolution are summarized below.

Theorem 6 (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 7 (Completeness of hp-refutation).
Let P be a consistent hp-program which satisfies the fixpoint reachability condition (see Def. 3.14) and Q' be a query. Then, if $P \models \exists(Q')$ then there exists an hp-refutation of Q' from P.

We elect not to give any examples of this proof technique as even trying to do resolution against a small program will entail describing a large closure of it. Instead we proceed directly to the second proof procedure, which is more derivation-driven and will provide examples for it.

The idea behind the second proof procedure is simple. Instead of *precomputing* the entire closure before a query has even been asked, we wait until we get the query and then dynamically reconstruct the part of the closure which is directly needed for the resolution of the query. We expect that each such dynamic reconstruction will require creating only a tiny fraction of the actual closure and this will make our second proof procedure quite efficient (at least compared to the first one).

Our second procedure, called HR_P-resolution is presented in a form of a Hilbert style inference system HR_P. We define it in the following manner.

Definition 3.17 *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 $A_1, A_2 \in B_L$*

   $$\frac{A_1 : \mu_1 \longleftarrow Body_1 \qquad A_2 : \mu_2 \longleftarrow Body_2}{(A_1 *_\rho A_2) : c_\rho(\mu_1, \mu_2) \longleftarrow Body_1 \wedge Body_2}$$

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

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

   - *Decomposition:* *L-Decomposition* *R-Decomposition*

   $$\frac{(F *_\rho G) : \mu \longleftarrow Body}{F : md_\rho(\mu) \longleftarrow Body} \qquad \frac{(F *_\rho G) : \mu \longleftarrow Body}{G : md_\rho(\mu) \longleftarrow Body}$$

   - *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

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

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

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 $P \vdash C$.*

If one compares the definition of HR_P with the definition of closure of P, clear parallels will be found. Each of the four key rule schemas of HR_P (A-Composition, F-Composition, Decomposition, Clarification) corresponds *exactly* to one of the four ways in which the closure of P can be augmented by new clauses (the Exchange rule schema is needed in HR_P to account for the commutativity of the formulas, it does not really create any new rules by itself).

Now we can see how HR_P can be used to answer queries. Assume a query Q has been asked to an hp-program P. Each step of the HR_P resolution will consist of two stages:

1. One conjunct $F : \mu$ from Q is selected. Then using HR_P inference system a clause C is derived from P which is resolvable with $F : \mu$. If no such rule can be derived, the negative answer to the query is returned. Otherwise, resolution proceeds to the next stage.

2. $F : \mu$ is resolved with C and is replaced in Q by the body of C. The query resolution process loops back to stage 1.

The soundness and completeness results for HR_P are based on the following theorems.

Theorem 8 *(soundness of HR_P w.r.t $CL(P)(P)$) For each hp-clause C, if $P \vdash C$ then $C \in CL(P)$.*

Theorem 9 *(completeness of HR_P w.r.t. $CL(P)$) For each hp-clause C, if $C \in CL(P)$ then $P \vdash C$.*

From these two theorems, and the soundness and completeness results for the hp-refutation procedure, we immediately derive the soundness and completeness of HR_P proof procedure.

Theorem 10 (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 11 (Completeness of HR_P)-Refutations).
Let P be a consistent hp-program which satisfies the fixpoint reachability conditions and Q' be a query. Then, if $P \models \exists(Q')$ then there exists an hp-refutation of Q' from P via HR_P.

The example below demonstrates how HR_P resolution works on practice.

Example 3.8 (**HR_P refutations**) Consider the following hp-program P:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$Q_{13} = \Box$

3.4 Tabulation Techniques for HPLPs

The material in this section is taken from [9].

The main idea behind the so called *OLD-Resolution* for logic programs is that while a query is evaluated, keeping track of the subqueries that have already succeeded can speed up the resolution process. OLD-Resolution and related techniques have been extensively studied, among other works, in [8, 15, 52]. All these works are done for classic Logic Programming.

In this section we describe how tabulation techniques can be applied to the proof procedure in HPLP setting. Before we proceed with our description of the tabulation model, we state one key difference between classic Logic Programs and HPLPs as far as it applies to the tabulation.

In classical LP we infer collections of facts from the program. However, the immediate influence of a newly inferred fact on possible deduction (or the impossibility of one) is not very large. For the most part, it only provides answers to all queries unifiable with the fact inferred.

In Hybrid Probabilistic Logic Programs, the situation is quite different. A derived fact $F : \mu$ can influence the probabilities of

1. all formulas that are unifiable with $F : \mu$;
2. all subformulas of F;
3. all superformulas of F;
4. all formulas which have a non-empty atom intersection with atoms of F.

For example, inferring new information about the probability of an event $a \wedge_\rho b$ may influence the probabilities of the following formulas : a, b, $a \wedge_{\rho'} b$, $a \vee_{\rho''} b$, $a \wedge_\rho b \wedge_\rho c$ and many more.

In developing tabulation techniques for HPLPs, we will need to be aware of this fact and be able to handle it.

3.4.1 Bcache and $b$cache Updates

Definition 3.18 *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 3.19 *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 $T[F] = \mu$ (as Θ is the identity substitution).

We fix an integer $b > 0$, a logical language L as defined in Lloyd[40], and a set $\mathcal{S}$ of p-strategies. Let $\mathcal{T}[b, L, \mathcal{S}]$ denote the set of all possible b-caches over $bf_{\mathcal{S}}(B_L)$. 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 into our b-cache. However, as noticed above simply appending 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 are 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 3.20 *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. $(\forall T \in \mathcal{T})(\forall F, G \in bf_{\mathcal{S}}(B_L))(\forall \mu, \nu \in \mathcal{C}[0,1]) f(f(T, F, \mu), G, \nu) = f(f(T, G, \nu), F, \mu)$
3. $(\forall T \in \mathcal{T})(\forall F \in bf_{\mathcal{S}}(B_L))(\forall \mu \in \mathcal{C}[0,1]) f(f(T, F, \mu), F, \mu) = f(T, F, \mu)$

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.

Finally, Clause (3) states that "redundant" updates should not change the b-cache.

Definition 3.21 *Let P be an hp-program and T be a b-cache. We say that T is sound 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 3.22 *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 the 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. [4]

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 3.23 *Let u and w be two b-cache update strategies. We say that u is more complete 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 equivalent **iff** *if both $u \geq w$ and $w \geq u$.*
An update strategy u is maximally complete **iff** $(\forall w)(u \geq w)$.

As pointed out earlier, Clause (3) of the definition of a b-cache update strategy postulates that no change in b-cache should occur when an update is repeated. However, this is not the only possible *redundant update*. The following proposition tells us how b-cache update strategies handle some other *redundant* updates:

Proposition 1 *For any b-cache T, and b-cache update strategy f, any basic formula F and any interval $\mu \in \mathcal{C}[0,1]$ the following holds: if $T \models F : \mu$ then $CN(T) = CN(T \uplus_f F : \mu)$.*

[4] the second property establishes it for a sequence of 2 basic formulas. It is easily extended to the case of sequences of 3 or more formulas.

3.4.2 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 3.24 (Atomic Updates) *Let T be a b-cache and A be an atomic (not necessarily ground) formula. An atomic update of T by $A:\mu$, denoted $T \uplus_{at} \{A:\mu\}$ is defined as follows:*

1. *If T has no atomic formulas that unify 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 3.9 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 3.25 (Propagated Atomic Update - *pat*) Let T be a b-cache, F be a basic formula. A *Propagated Atomic Update* strategy (*pat*) is defined as follows:

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

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 3.10 *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 3.26 (Elementary b-cache update) *Let T be a b-cache, F be a basic formula. We define an elementary b-cache update 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 $F \equiv F_1 *_\rho \ldots *_\rho F_m$. We proceed in a number of steps.*
   (a) *Let $T' = T \uplus_{pat} F : \mu$*
   (b) *Let $\{\langle \nu_1, \Theta_1\rangle, \ldots \langle \nu_k, \Theta_k\rangle\} \subseteq T'[F]$ be all pairs from $T'[F]$, s.t., $\nu \not\subseteq \mu$. We proceed in steps. Let $T^0 = T'$. Consider T^i $(0 \leq i < k)$ 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 $F = F_1 *_\rho \ldots *_\rho F_m$ Let $B_1, B_2, \ldots B_k$ be all subformulas of F of size b. Then $T \uplus_{eb} \{F : \mu\} = (T \uplus_{pat} F) \uplus_{eb} B_1 : md_\rho(\mu) \uplus_{eb} \ldots \uplus_{eb} B_k : md_\rho(\mu)$.*

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 3.11 *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 3.27 (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* $F \equiv F_1 *_\rho \ldots *_\rho F_m$, $m \leq b$. *Let* $B_1, \ldots B_k$ *be all the subformulas of F of size* $< m$. *We declare* $T \uplus_{fb} F : \mu = T \uplus_{eb} F : \mu \uplus_{eb} B_1 : md_\rho(\mu) \uplus_{eb} \ldots \uplus_{eb} B_k : md_\rho(\mu)$.
3. **Case 3.** $|F| > b$. *Let* $F = F_1 *_\rho \ldots *_\rho F_m$ *Let* $B_1, B_2, \ldots B_k$ *be all subformulas of F of size* $\leq b$. *Then* $T \uplus_{fb} F : \mu = (T \uplus_{pat} F) \uplus_{eb} B_1 : md_\rho(\mu) \uplus_{eb} \ldots \uplus_{eb} B_k : md_\rho(\mu)$.

The following result tells us that the full b-cache update strategy is more complete than the elementary b-cache update strategy.

Proposition 3 *(i)* $(\forall b > 0) fb \geq eb$ *(ii)* $f1 \equiv e1 \equiv pat$

As the reader may easily notice from the definitions, implementing atomic updates is easy, however, PAT is more efficient than the elementary b- cache strategies 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.

3.4.3 Proof Procedure for HP-Programs with b-cache

In Section 3.3, 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 the 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 the current b-cache with one or more newly proven formulas.

Definition 3.28 *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* $F_{i_1} : \mu_{i_1} \ldots F_{i_n} : \mu_{i_n}$ *be 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 3.29 *We define a b-cache supported resolvent and a b-cache update procedure simultaneously. 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_ρ *is a set (possibly empty) of annotated basic formulas (not necessarily 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 3.30 *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* $1 \leq i < r$, $T_{i+1} = \phi_u(\hat{Q}_i, T_i, C_i, \Theta_i)$

Example 3.12 (2-cache supported hp-refutation with elementary update strategy) Let us return to the hp-program shown in Example 3.8 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. $Q_1 =< (a : [0.9, 1], \emptyset), (e : [1, 1], \emptyset) >$
   $T_1 = \emptyset; P \ni C_1 = a : [1, 1] \longleftarrow (b \wedge_{ind} c \wedge_{ind} d) : [0.25, 1] \wedge f : [0.5, 0.9]$.

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 straightforward, as we know that HR_P is sound and complete, and the b-cache supported hp-refutation via HR_P is just its conservative extension.

Theorem 12 (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 13 (Completeness of b-cache supported hp-refutation).
Let P be a consistent hp-program which satisfies the fixpoint reachability conditions let 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.

3.5 Algorithms and Complexity Results

The material of this section is taken from [13].

In this section we briefly summarize the results, obtained in [13]. In that paper, HPLPs had been divided into a number of subclasses and the problems of *fixpoint model computation*, *entailment* and *consistency* have been studied on these classes of programs/ We notice that **all** results of this section are given for ***ground HPLPs***.

The classes, that HPLPs are divided into are organized by the sizes of the formulas in the heads of rules and lengths of bodies of the rules.

Given a basic formula G, we define *width*(G) to be the number of atoms in G.
Given an hp-clause $C = B_0 : \mu_0 \leftarrow B_1 : \mu_1 \wedge \ldots \wedge B_k : \mu_k$, we say that the *head-width* of C is the width of B_0, and the *body-width* of C is $\max\{width(B_1), \ldots, width(B_k)\}$.

Definition 3.31 *Let $HPP_{k,r}$ denote the class of HPP-programs P such that for all clauses $C \in P$, the head-width of C is less than or equal to k and the body-width of C is less than or equal to r. Let $HPP_k = \bigcup_{r \geq 0} HPP_{k,r}$.*

Given an arbitrary ground program P from the class $HPP_{k,r}$ with m clauses and a different atoms found in it, we show that it is possible to compute the least fixpoint model of the program for all formulas of size $N \geq \max(k, r)$ in the time $O(2m(2sa^N)^2) = O(m(sa^N)^2)$ (here s is the number of different P-strategies in our current framework).

Then, we focus on two other problems for HPLPs: entailment and consistency. For these problems, we separate our results into three categories as presented in the table below.

Table 1: Complexity results

Problem	HPP_1	HPP_2	HPP_r $r \geq 3$
Entailment	polynomial	polynomial for $c_\rho \in \{inc, igc, pcc, igd, pcd, ncd\}$	NP-complete
Consistency	polynomial	polynomial for $c_\rho \in \{inc, igc, pcc, igd, pcd, ncd\}$	co-NP-complete

We would like to point out the interesting results that we have obtained for class HPP_2 of hybrid probabilistic programs in which heads of rules contain at most two atoms. Entailment problem for such programs was shown to be polynomially reducible to a generalization of a weighted matching problem.

Generalized weighted matching problem

Given an edge-weighted, undirected graph $G = \langle V, E, w \rangle$ and a goal weight combination function c, find a complete matching for which the goal function on weights of selected edges is maximized (minimized).

More formally, we define two classes of "yes-no" matching problems:
$GWM_{max}(c) = \{G = \langle V, E, w : E \rightarrow [0,1] \rangle, B \in [0,1] || V| = 2m\}$ and there exists a complete matching $\{e_1, \ldots, e_m\} \subseteq E$ such that $c(w(e_1), \ldots, w(e_m)) \geq B\}$ and
$GWM_{min}(c) = \{G = \langle V, E, w : E \rightarrow [0,1] \rangle, B \in [0,1] || V| = 2m\}$ and there exists a complete matching $\{e_1, \ldots, e_m\} \subseteq E$ such that $c(w(e_1), \ldots, w(e_m)) \leq B\}$.

Basically the difference between our GWM problems and the weighted matching problem ([41]) is in the fact that weighted matching combines the weights of edges using +, while GWM problems may use different, more sophisticated functions.

Given a p-strategy ρ with composition function $c_\rho = [c^1_\rho, c^2_\rho]$ we will use c^1_ρ and c^2_ρ as goal functions of GWM problems in our reduction of GWM to entailement for HPP_2. Remember that composition functions of p-strategies must satisfy certain axioms, specified in Defs. 3.1 and 3.2.

Theorem 14

(1) Let $P \in HPP_2$ use probabilistic strategies with combination functions $c = \langle c^1, c^2 \rangle$ where $c^1 \in C^1, c^2 \in C^2$ for some sets of functions C^1 and C^2. Then the entailment problem for P and annotated basic formula $F : \mu$ is polynomially reducible to the problems $GWM_{max}(c^1)$ and $GWM_{min}(c^2)$, where $c^1 \in C^1, c^2 \in C^2$.

(2) Any generalized weighted matching problem for goal functions, satisfying axioms (a)-(d) of Definitions 3.1, is reducible in polynomial time to entailment problems for hp-programs of $HPP_{2,0}$.

It is well-known [41] that the weighted matching problem is solvable in polynomial time. Hence, for a number of probabilistic strategies for which the weight functions are easily reduced to addition or computations of min and max, we can obtain polynomial time-bound on the entailment problem. Such p-strategies include all but one the example in Section 3.1. GWM problems with *disjunctive p-strategy of independence* as goal function are not immediately reducible to weighted matching problem. For all other p-strategies we get

Corollary 1 *The entailment problem for the class of HPP_2 programs over strategies $\mathcal{S} = \{inc, igc, pcc, igd, pcd, ncd\}$ is solvable in polynomial time.*

We still do not know the complexity of the Generalized Weighted Matching Problem, neither were we able to find the this version of weighted matching problem described before.

Going back to assessing the obtained complexity results for HPLPs we notice that despite the open problem of the general complexity of entailment for HPP_2, we were able to specify a non-trivial class of hybrid probabilistic programs for which entailment problem is solvable in polynomial time. In fact, even if it turned out that GWM problems are NP-complete, we could still hope that for particular probabilistic p-strategies (such as *ind*) we can find a reduction of GWM to an instance of weighted matching solvable in polynomial time.

Another interesting complexity result that we were able to obtain states that even for most general versions of HPPs, if a formula $F : \mu$ is a consequence of a (ground) program P then there exists a *polynomially bounded (in size) derivation* of $F : \mu$ from P. This does not contradict the fact that entailment in general is NP-complete, as there may be exponentially many polynomial-sized derivations of $F : \mu$ from P.

More formally, we define a an inference system HGR_P, which is similar in spirit to our HR_P inference system (see Section 3.3), only instead of clauses, as HR_P did, this new system infers basic formulas. HGR_P is shown to be sound and complete w.r.t. model theory for HPLPs and we show that if $P \vdash_{HGR_P} F : \mu$ then there exists a derivation of $F : \mu$ in HGR_P of polynomial size.

The proof of this fact involves careful construction of such derivation and estimation of its size.

4 Towards Probabilistic Reasoning With Time And Space

In the preceding sections, we described one model that deals successfully with generic probabilistic uncertainty in databases (ProbView model of Lakshmanan *et al.*[36]) and one model that deals with probabilistic uncertainty in Logic Programming (HPLP). Both models can work with any kind of probabilistic data. So, the question arises, why do we need to construct special database and logic programming frameworks for dealing with temporal and spatial (n-dimensional) probabilities. And is there a dire need for temporal and spatial probabilistic reasoning at all ?

In this section we will provide a number of examples intended to support our claim that such reasoning is indeed useful. We will then look at the drawbacks of ProbView and HPLP w.r.t. dealing with temporal and spatial probabilities, and will specify the benefits we want to reap from creating database and logic programming models specifically intended for dealing with temporal and spatial probabilities.

4.1 Motivating Examples

Example 4.1 Scheduling.

Consider the databases maintained by a transportation provider such as CSX or Federal Express. When a package is delivered to such an organization for shipping, a tentative shipping *schedule* is created for the package. The transportation provider must maintain such schedules for millions of packages. Such schedules specify which flight (or truck) the shipment is scheduled to leave on, when the shipment will reach a waypoint, and so on. However, there is uncertainty about how long a particular part of the schedule will actually take. For example, Federal Express may ship a package from Boston to Chicago via Albany, NY. They have reliable statistics on how long the Boston-Albany leg takes, and how long the Albany-Chicago leg takes. A user who wants to know when his shipment is likely to reach him usually gets an *uncertain* answer of the form "Either today (36%) or tomorrow (64%)." A database system used by such a transportation vendor must have the ability to handle temporal modes of uncertainty.

Example 4.2 Weather Applications.

Consider a weather database that tracks the weather at a fixed location (e.g. Washington). Such a weather database contains not only information about the weather in Washington in the past, but also contains projections for the future. Needless to say, any prediction about the future is liable to be uncertain. How often have we heard a TV newscaster say "There is a 39% probability of rain this afternoon"?

Example 4.3 Time-Series Stock Applications.

There are a wide variety of programs that analyze the behavior of stocks, and predict their rise and fall in the future. Most such programs associate with their predictions a level of uncertainty. Such programs may say "We expect, with 60-70% certainty, that IBM stock will fall by 30% sometime in the next 2 weeks." When the output of such programs is to be stored in a relational database system, we must have the ability to represent and manipulate such statements.

Example 4.4 Scouting robots.

Reconnaissance without the use of human scouts is now an integrated part of mobile warfare. Special small scout robots are sent into the territory for which the reconnaissance is needed. Such robots have a number of external sensors which allow them to obtain information about the environment. Depending on this information, the program that guides the robot's behavior chooses what to do. In such a setting, a lot of information received by the robot is very uncertain. The uncertainty may be due to various reasons, such as errors in sensor readings (any time a distance to some object is computed, there is always a certain error in the measurement), dated information (e.g., 20 minutes ago a car had been spotted in a location that is currently invisible to the robot; whether or not the car is still in the same location is not known, but the robot may assume that there is a high probability that the car is indeed there because there were no humans sighted near it and it had not been seen leaving the street), inability to distinguish certain properties of sighted objects (e.g., a tank is sighted at a certain location but it is unknown whether it is a friendly or hostile tank). In the first case the uncertainty is purely spatial, in the second, it is both temporal and spatial as the reasoning has to be made about the changes of location in time. In the third case the uncertainty itself may not be spatial or temporal, but (probabilistic) knowledge about locations of friendly tanks may allow the robot to infer the probability that the sighted tank is hostile.

Example 4.5 Network Administration

Administration of large computer networks can be performed in part by a so called policy server. A policy server analyzes the events that take occur in the network and tries to manage the network. The server can use very simple inference rules, such as "If a crash occurred, switch system to overload mode" to draw inferences that are intended to manage the the network.

However, this is also a very natural place to apply uncertain reasoning. By running the network and logging all transactions one can get a lot of data about the behavior of the network. This data can be statistically analyzed and the results of this analysis can be used by the policy server in making decisions. Clearly ,some of such data is of probabilistic nature [5]. For example, the data may indicate that in 91% of all cases when computer *jedi* is used by more than 15 users at the same time, a system crash occurs on that computer within 40 minutes. Then, policy server running under the instructions that anytime the probability of a crash on exceeds 50%, warning messages should be issued to all people currently working with the affected computer can use this information to decide when to post such warnings. It may also decide not to let anyone else log on that computer until the problem is cleared or some user of the computer in question leaves.

Probabilistic temporal reasoning is important here, as many actions manifest their consequences not immediately but rather with a delay which may or may not be predictable.

4.2 The Need For Specific Frameworks

The examples from the previous section provide, in our opinion, a sufficient justification of the need to reason with temporal and spatial probabilities. In this section we will discuss the drawbacks of using ProbView database model and/or HPLPs for this purpose. We will also formulate the goals that the new frameworks, tailored to deal with these particular kinds of data should strive to achieve.

Drawbacks of ProbView

- **Use of annotated representation to store probabilistic data.** This really was the main reason why the new approach to managing temporal (and later, spatial) probabilistic data had

[5] Assuming that we agree that statistically computed frequencies of occurrence of certain phenomena extracted from a large sample can be safely used as the probabilities.

been considered. Indeed, annotated representations will work just fine as long as the number of possible annotated tuples in an annotation of one probabilistic tuple is relatively small. This may be okay for examples like the ones we gave for probabilistic databases, where the possible space of uncertain values for each attribute was rather small. However, when time is considered, this may quickly become a problem. Depending on application we may wish to have rather small units of time as our "chronons" [6] in a temporal probabilistic database.

Let revisit our network example from the previous section. For the policy server application the chronon can be as small as one processor tick or the time, that the processor takes to perform a certain operation. Such granularity may be needed to be able to tell in what exact order some instructions/operations had been executed. Suppose now that we want to express the fact that computer a crash has occurred sometime within the last two hours. Even considering that our chronon is in the order of a millisecond, the probabilistic tuple expressing the statement about crash will be annotated by $1000 \cdot 60 \cdot 60 \cdot 2 = 240000$ annotated tuples: one tuple represents the probability that crash occurred at a particular (up to a millisecond) timepoint.

Consequently, the probabilities we will be dealing with in this case for each annotated tuple will be very small.

Clearly, a better way to represent such temporal probabilistic data is desirable. The ProbView data model suggests one possible solution (which in fact we adopted for our TP Databases). This solution is to store tuples in their compressed format, not unlike the Probabilistic tuple format (see next point on discussion about the data format issue). To match the storage method, we would need to develop the relational algebra on such compressed representation of data, something that [36] fell short of doing.

- **ProbView data format is too general.** While hardly a drawback by itself, the generality of the ProbView data format turns out to be excessive for storing probabilistic temporal data. We want to reason about the events whose composition is known for sure, with the **only uncertainty** being in the time when the event occurred. This means that we only need *one* probabilistic attribute in the tuple which represents temporal probabilistic information. Probabilistic tuples defined in [36] allow us to specify as many uncertain attributes in a relation as needed.

  So, as there are other reasons why developing a specific temporal probabilistic database framework is desirable, we will also want the specificity of data format to be one of the features of any such framework designed.

- **Generality of some relational algebra operations.**

  One of our key goals in trying to find a good representation of temporal probabilistic data is to make relational algebra operations on such data *intuitively meaningful*. In the light of this, it may be preferable for us to opt for some operations to be defined in a manner which makes them different from classical relational algebra operations, but does correspond to certain reasonable ways of managing the temporal probabilistic data. Examples of such operations include *cartesian product* and *difference*. Jumping a little bit ahead of ourselves, we mention that in one case (cartesian product) we decided to define and implement a more sophisticated version of an operation because the more traditionalistic approach to it lacked the temporal intuition. In another case (difference) we actually simplified the operation, as

[6] "chronon" is the smallest nondivisible period of time in a calendar.

ProbView difference had been found lacking certain intuitions, consistent with how we defined other operations in our algebra. More explanations about our choices of relational algebra operations can be found throughout Section 5.3.

In general, while evaluating possible approaches to dealing with temporal probabilistic data we decided that an approach based on a *specific, tailored to the properties of time and temporal probabilities* data manipulation would be most beneficial in this case.

As the case of spatial, and in general any nDimensional probabilistic reasoning is even more complicated than pure temporal probabilistic reasoning, we should expect that the same drawbacks of the general model, such as ProbView would apply, and hence similar assessments about the relative goodness of general vs. specific data management approach remain intact.

5 Temporal Probabilistic Databases

The material in this section comes from [11], except for Section 5.4. The material of that section is from [12].

The Temporal Probabilistic (TP) Database model, proposed in this section is based on the ProbView database model. However, a number of features, key to ProbView were modified in TP Database model. We will address the differences as we describe our model.

[FROM ALEX: **Need to state that this is joint work with Rob**]

The structure of this section is as follows. First, we briefly describe a few general notions that are used in Temporal Probabilistic Databases. Then, in Section 5.2 we describe the structure of Temporal Probabilistic relations (TP-relations, for short). In the same section we also describe Temporal Probabilistic Annotated Relations (we will refer to them below simply as *annotated relations*) which are intended to represent the *meaning* of TP-relations. We also describe the connection between annotated and TP-relations and the operation of annotation.

Section 5.3 describes the relational algebras for both TP-relations (we call it TPA) and annotated relations (we call it TATA). In this section we will explain what it means for an operation of TPA to *correctly implement* an operation of TATA and will state the results showing that all relational operations of TPA correctly implement their respective analogs in TATA.

In Section 5.4 we will formulate currently known equivalences between the queries of TPA and will discuss the plans for the development of a cost model (or cost model*s*) for TPA.

Temporal Probabilistic Databases are being designed to deal with temporal uncertainty, and because of this, first thing we need to specify is our notion of *time* which lies in the base of the theory.

5.1 Preliminary definitions

5.1.1 Calendars and Temporal Constraints

Out of many possible representations of time (among which one can count various temporal logics and segments of natural real numbers) we elected to use the *calendars* as described in [35]. In choosing this particular approach we were guided by our desire to have a representation of time which (i) would allow for simple conversion of the "real life" dates and times in to our representation of them and (ii) provide enough flexibility for us to be able to select the desired *level of granularity* of time.

More formally, *calendars* are defined as as follows.

Definition 5.1 *A* time unit *consists of a* name *and a* linearly ordered value set*.*

A linear hierarchy of time units *consists of a finite set H of time units and a linear order $\sqsubseteq$ defined on H. We require that the value sets for all time units in the hierarchy, with the possible exception of the "greatest time unit" w.r.t. $\sqsubseteq$ are* **finite.**

Suppose $T_1 \sqsubseteq \cdots \sqsubseteq T_n$ is a linear hierarchy H of time units. A time point *t in H is an n-tuple $(v_1, \ldots, v_n)$ such that for all $1 \leq i \leq n$, v_i is a time-value in the time-value set of T_i.*

A calendar *τ consists of a linear hierarchy H of time units and a* validity predicate *denoted* valid_H *(or simply* valid *if H is clear from context). A validity predicate specifies a non-empty set of valid time points;* $\mathrm{valid}_H(t)$ *is true iff t is a valid time point. The* set of all time points over calendar τ, *denoted S_τ, is defined as $\{t \mid t$ is a time point in H and* $\mathrm{valid}_H(t)$ *is true$\}$*

Here is an example of a calendar.

Example 5.1 *Let time unit hierarchy H = "month" $\sqsubseteq$ "day" $\sqsubseteq$ "hour" $\sqsubseteq$ "minute", where the sets of values for the time intervals are set to be respectively $\{1, \ldots 12\}$, $\{1, \ldots 31\}$, $\{0, \ldots 23\}$, $\{0, \ldots 59\}$. Let also predicate $valid_H$ be defined in a "standard" way, following the Gregorian calendar (i.e., $valid_H$ correctly defines the length of every month, e.g., stating that all tuples that feature "month" = 2 and "day" > 28 are invalid). Then $\tau = \langle H, valid_H \rangle$ is a calendar which enumerates all timepoints in the duration of one year with th e granularity of 1 min.*

Next we define *temporal constraints* over an arbitrary but fixed calendar τ.

Definition 5.2 (atomic temporal constraint) *Suppose $T_1 \sqsubseteq \cdots \sqsubseteq T_n$ is a linear hierarchy H of time units over calendar τ. An* atomic temporal constraint over calendar τ *is either $(T_i$ op $v_i)$ where op is a member of the set $\{\leq, <, =, \neq, >, \geq\}$ and v_i is a time-value in the time-value set of time unit T_i or $(t_1 \sim t_2)$ where $t_1, t_2 \in S_\tau$ and $t_1 \leq_H t_2$. We let (t_1) be an abbreviation for $(t_1 \sim t_1)$.*

A temporal constraint is either an atomic constraint or conjunction or disjunction *of already known constraints, or* negation *of an already known constraint.*

Now we can define what a solution of a temporal constraint is.

Definition 5.3 (solution set to an atomic temporal constraint) Suppose $T_1 \sqsubseteq \cdots \sqsubseteq T_n$ is a linear hierarchy H of time units over calendar τ. Then an atomic temporal constraint C over τ is of the form $(T_i\ op\ v_i)$ or $(t_1 \sim t_2)$. The *solution set to an atomic temporal constraint over calendar* τ is the set $S = sol(C)$ which is defined in the following way:

Case	$sol(C)$
$op = (\leq)$	$\{t \mid t \in S_\tau \wedge t.T_i \leq_{T_i} v_i\}$
$op = (<)$	$\{t \mid t \in S_\tau \wedge t.T_i <_{T_i} v_i\}$
$op = (=)$	$\{t \mid t \in S_\tau \wedge t.T_i = v_i\}$
$op = (\neq)$	$\{t \mid t \in S_\tau \wedge t.T_i \neq v_i\}$
$op = (>)$	$\{t \mid t \in S_\tau \wedge v_i <_{T_i} t.T_i\}$
$op = (\geq)$	$\{t \mid t \in S_\tau \wedge v_i \leq_{T_i} t.T_i\}$
$op = (\sim)$	$\{t \mid t \in S_\tau \wedge t_1 \leq_H t \leq_H t_2\}$

□

Let C now be a temporal constraint. Then the solution set of C can be found as follows:

$C = (C_1 \wedge C_2)$	$sol(C) = sol(C_1) \cap sol(C_2)$
$C = (C_1 \vee C_2)$	$sol(C) = sol(C_1) \cup sol(C_2)$
$C = \neg C_1$	$sol(C) = S_\tau - sol(C)$

□

5.1.2 Probabilistic Distribution Functions

We use probabilistic distribution functions to compress the information about the distribution of the probabilities of events in time.

Definition 5.4 (probability distribution function) Let D be a temporal constraint over calendar τ such that $|\mathrm{sol}(D)| \geq 1$. Then a *probability distribution function (PDF) over calendar* τ, denoted $\mathrm{pdf}(D, t_j)$, is a function which takes D and a time point $t_j \in S_\tau$ as input, and returns as output a probability ρ_j which satisfies the following conditions:

1. For each $t_j \in S_\tau$, $0 \leq \rho_j \leq 1$.
2. For all $t_j \in S_\tau$ where $t_j \notin \mathrm{sol}(D)$, $\rho_j = 0$.
3. $\sum_{t_j \in S_\tau}(\rho_j) \leq 1$. This implies that $\sum_{t_j \in \mathrm{sol}(D)}(\mathrm{pdf}(D, t_j)) \leq 1$.

If the sum $\sum_{t_j \in S_\tau}(\rho_j)$ is strictly less than one, the function is called a *partial PDF*; if the sum is exactly equal to one, the function is called a *complete PDF*. A PDF is *determinate* if $\sum_{t_j \in S_\tau}(\rho_j)$ is computable in constant time. □

Notice that PDFs are both discrete and finite. Complete PDFs tell us what percentage of the total probability mass (i.e., 1.0) is associated with each $t_j \in \mathrm{sol}(D)$. Partial PDFs are useful when modeling infinite distributions; here, we are considering only a finite portion of the total probability mass. Here are some examples of pdfs. Determinate PDFs tell us up-front that a fixed percentage of the probability mass is unassigned. Thus, every complete PDF is determinate. In addition, a partial PDF which is known to allocate only a total of 0.9 to the values in S_τ is determinate.

To see how specific PDFs may be defined, let us examine some examples.

Example 5.2 (PDF; uniform) The *PDF for the uniform distribution over calendar* τ, denoted $\mathrm{pdf}_u(D, t_j)$, is defined as $\rho_j = \frac{1}{|\mathrm{sol}(D)|}$ if $t_j \in \mathrm{sol}(D)$ or $\rho_j = 0$ otherwise. pdf_u is a complete PDF.

For the following PDF examples, let D be a temporal constraint and let $t_0, \ldots, t_n$ be a list of distinct time points in S_τ where $\mathrm{sol}(D) = \{t_0, \ldots, t_n\}$ and t_i *occurs before* t_{i+1} for all $0 \leq i < n$, i.e. $sol(D)$ is enumerated in ascending order of time points.

Example 5.3 (PDF; geometric) Let p be a probability where $(0 < p < 1)$. Then the *PDF for the geometric distribution with parameter p over calendar τ*, denoted $\mathrm{pdf}_{g,p}(D, t_j)$, is defined as $\rho_j = p \cdot (1-p)^i$ if $t_j = t_i \in \mathrm{sol}(D)$ or $\rho_j = 0$ otherwise. pdf_g is a partial PDF. Note that if $|\mathrm{sol}(D_j)|$ is fixed (or constant time computable), then pdf_g is a determinate PDF.

Example 5.4 (PDF; binomial) Let p be a probability where $(0 < p < 1)$. Then the *PDF for the binomial distribution with parameter p over calendar τ*, denoted $\mathrm{pdf}_{b,p}(D, t_j)$, is defined as $\rho_j = \binom{n}{i} \cdot p^i \cdot (1-p)^{n-i}$ if $t_j = t_i \in \mathrm{sol}(D)$ or $\rho_j = 0$ otherwise. pdf_b is a complete PDF.

Example 5.5 (PDF; Poisson) Let $(\lambda > 0)$ be a rate and let e be the base of the natural logarithm (i.e., $e \simeq 2.71828$). Then the *PDF for the Poisson distribution with parameter λ over calendar τ*, denoted $\mathrm{pdf}_{po,\lambda}(D, t_j)$, is defined as $\rho_j = e^{-\lambda} \cdot \frac{\lambda^i}{i!}$ if $t_j = t_i \in \mathrm{sol}(D)$ or $\rho_j = 0$ otherwise. pdf_{po} is a partial PDF. When $|\mathrm{sol}(D)|$ is known, then pdf_{po} is a determinate PDF.

PDF can be specified by simpler means as well.

Example 5.6 (PDF; "User-defined") Let $sol(D) = \{t_1, t_2, t_3, t_4\}$. We specify a probabilistic distribution function *pdf_D as follows:*
$pdf_D(t_1) = 0.2$
$pdf_D(t_2) = 0.3$
$pdf_D(t_3) = 0.1$
$pdf_D(t_4) = 0.4$

5.1.3 Probabilistic Strategies

Probabilistic strategies used in our algebra are the same *probabilistic conjunction operators* $\otimes_\rho$ and *probabilistic disjunction operators* $\oplus_\rho$ as in ProbView.

Other than being presented with different syntax $\otimes_\rho$ and $\oplus_\rho$ operations can be thought of as the *composition functions* from *probabilistic strategies* defined in Section 3.1 (See Def. 3.1 and 3.2).

The use of probabilistic strategies in our relational algebra will be similar to their use in ProbView (see [36] and Section 2.2 for description). We will use probabilistic conjunctions during the computation of a cartesian product of two temporal probabilistic relations. And although we do not explicitly use probabilistic disjunctions during the compaction operation, as such use would contradict our adopted principle of assuming that *all* data in a temporal probabilistic relation is true, we still left the door open for its use in compaction operation for those people who prefer the less strict approach to the data.

5.2 Temporal Probabilistic Relations (TP-relations)

We are now ready to define Temporal Probabilistic (TP) relations.

In this definition lies an important difference between the TP Databases and the ProbView databases. Remember (Example 2.2) that **every** attribute in a tuple in a ProbView-style probabilistic relation can be uncertain. This is a very nice feature of ProbView framework: in the same relation, information about many different "reasons" for the data to be uncertain can be kept easily. This is also a major drawback of ProbView, as the increase in expressive power is paid for by a potentially exponential blow up of annotations of probabilistic relations.

For our Temporal Probabilistic framework we chose a more restrictive approach. Our main assumption here is that ***the only uncertainty we are dealing with in our data is the uncertainty about <u>when</u> particular events occurred in time.*** This allows us to consider the composition of actual events to be certain (i.e., we always know *what the event is*, we just are not sure *exactly when* it occurred (or will occur)). What it means in terms of the syntax of our TP-relations, is that each tuple in them will contain **only one** probabilistic attribute - associated with time.

Here is how we define our TP-relations formally. The definition consists of 3 parts. First we define the structure that allows us to keep information about the probability of one event at many timepoints in one tuple. Once defined, we will combine this structure with the regular relational database structure which describes the actual event to obtain a Temporal Probabilistic tuple (tp-tuple). Then a TP-relation is defined as a collection of tp-tuples.

Definition 5.5 (TP-case statement over calendar τ) A *TP-case statement over calendar* τ, denoted γ, is an expression of the form $\{\langle C_1, D_1, L_1, U_1, \delta_1\rangle, \ldots, \langle C_n, D_n, L_n, U_n, \delta_n\rangle\}$ where $n \geq 1$, C_i and D_i are a temporal constraints over τ, L_i and U_i are probabilities, δ_i is a distribution function over τ, and the following conditions are satisfied for all $1 \leq i \leq n$:

1. $(0 \leq L_i \leq U_i \leq 1)$.
2. $\mathrm{sol}(C_i) \subseteq \mathrm{sol}(D_i)$. This ensures that $\delta_i(D_i, t)$ is defined for each time point $t \in \mathrm{sol}(C_i)$.
3. $|\mathrm{sol}(C_i)| \geq 1$. In other words, C_i and D_i each have at least one solution in S_τ.
4. For all $1 \leq j \leq n$, $i \neq j \Rightarrow \mathrm{sol}(C_i) \cap \mathrm{sol}(C_j) = \emptyset$. In other words, $(C_i \wedge C_j)$ is always inconsistent. This ensures that each TP-case statement specifies at most one probability interval for each $t \in S_\tau$. Note that we do not have a similar requirement for $(D_i \wedge D_j)$.

For each $1 \leq i \leq n$, $\gamma_i = \langle C_i, D_i, L_i, U_i, \delta_i\rangle$ is called *a TP-case* of γ. On occasion, we may want to assign probabilities to every time point in S_τ. Here, $\mathrm{sol}(C_n) = \mathrm{sol}(\neg C_1 \wedge \neg C_2 \wedge \ldots \wedge \neg C_{n-1})$ and γ_n is called the *catch-all case*. For brevity, when γ_n is a catch-all case, we may use "(∗)" to represent C_n.

Note: If $\mathrm{sol}(C_i) = \mathrm{sol}(D_i)$, we let "(#)" be an abbreviation for C_i. □

Each tp-tuple will contain a so called "hidden field" which is introduced here to keep separate tuples referring to different events. The definition below briefly explains its format.

Definition 5.6 (hidden field) A *hidden field* holds a lexicographically sorted *hidden list* of *field-value pairs* (i.e., "<field_1>:<value_1>, …, <field_n>:<value_n>"). If there are no pairs to store, the hidden list will be EMPTY. □

Now we are ready to proceed onto the definition of tp-tuples.

Definition 5.7 (TP-tuple) Let $T_1 \sqsubseteq \cdots \sqsubseteq T_m$ be the linear hierarchy of time units over calendar τ and suppose $A = (A_1, \ldots, A_k)$ is a relational schema where for all $1 \leq i < k$, $A_i \notin \{$"C", "D", "L", "U", "δ", "L_t", "U_t", "H"$\}$, and for all $1 \leq j \leq m$, $A_i \neq T_j$. Furthermore, let A_k be the *hidden field* "H", let $d = (d_1, \ldots, d_k)$ be a (data) tuple over A, and let γ be a TP-case statement over τ. Then $tp = (d, \gamma)$ is a *TP-tuple over relational schema A and calendar τ*. Intuitively, γ gives the probability for each $t \in S_\tau$ that d occurs at time t. □

Definition 5.8 (TP-relation) A *TP-relation over relational schema A and calendar τ*, denoted r, is a multiset of TP-tuples over relational schema A and calendar τ. Intuitively, a *base TP-relation* is a TP-relation which did not result from a query. If r is a base TP-relation, then for each TP-tuple $tp = (d, \gamma) \in r$, (i) d.H = EMPTY and (ii) for each TP-case $\langle C_i, D_i, L_i, U_i, \delta_i \rangle \in \gamma$, $C_i = D_i$.

We associate with each TP-relation a *primary key.*

Definition 5.9 (TP-database) A *TP-database over calendar τ* is a pair $(Base, MView)$ where $Base$ is a set of base TP-relations over τ and $MView$ is a set of non-base TP-relations over τ. □

Let us now go back to the definition of a TP-case statement as see why it had to be specified in this manner.

The first thing that may make one wonder is the presence of *two* constraints (C_i and D_i) in a TP-case $\gamma_i = \langle C_i, D_i, L_i, U_i, \delta_i \rangle$. Intuitively, sol($C_i$) is the set of time points which γ_i is "interested in" while sol(D_i) is the set of time points used when distributing the probability interval $[L_i, U_i]$ according to δ_i. When the above TP-case is associated with a data-tuple d, it says that d is in some relation at some time point t that is a solution of the constraint C_i. The probability that d is true in the relation at such a t is $\delta_i(D_i, t)$. In other words, the constraint D_i is used to specify the set of time points used when distributing the probability interval $[L_i, U_i]$ according to δ_i. **This is an important distinction which is critically necessary.** Why ? Suppose that originally, sol(C_i) = sol(D_i) = $S = \{1, 2, 3, 4\}$ and δ_i = "$b, 0.5$". Thus, the probabilities associated with time points 1,2,3,4 are 0.125, 0.375, 0.375, 0.125. Now suppose we perform a *selection* operation [7] which only asks for time points in the set $S' = \{2, 3\} \subset S$. If we had no D_i field in our TP-cases, then we would merely carry over the fact that S' has the binomial distribution on it. But applying the binomial distribution to this set yields a probability of 0.5 to both 2 and 3 which is incorrect because selections should not change the probabilities assigned to time points $t \in S'$. Thus, some mechanism is needed to correctly compute the probabilities of relations resulting from algebraic operations executed.

Thus, in order to accurately compute probabilities, we must do one of two things: either (i) *carry with us the original set of values over which a probability distribution was defined*, or (ii)*Determine how to accurately refine an* arbitrary *distribution to apply to a subset of the set to which the distribution was originally applicable.*

The latter option requires a complex algebraic theory of distributions and its implementation is likely to be extremely expensive. For this reason, we have chosen the first option above.

For another (simpler) example, consider a TP-case statement with one TP-case γ_1:
$\{\langle (1/8/1997 \sim 5/8/1997), (1/8/1997 \sim 10/8/1997), 0.4, 0.8, u \rangle\}$ Intuitively, γ_1 says that some event occurred during the first five days of August 1997 (in other words, it occurred during one of the

[7]selections are described formally below but the basic intuition about selection operation the same as in the classical relational algebra

time points in $\mathrm{sol}(C_1)$). Since $\delta_1 =$ "u", the probability that it occurred on any of these days is the same. Specifically, this probability is $[\frac{1}{10} \cdot 0.4, \frac{1}{10} \cdot 0.8] = [0.04, 0.08]$ since we are uniformly distributing the probability mass $[0.4, 0.8]$ between all of the (10) time points in $\mathrm{sol}(D_1)$. In general, the probability interval for some time point $t \in \mathrm{sol}(C_i)$ is $[L_i \cdot \delta_i(D_i, t), U_i \cdot \delta_i(D_i, t)]$. Here, we see that a TP-case $\langle C_i, D_i, L_i, U_i, \delta_i \rangle$ is simply an extension to the P-tuple $\langle D_i, L_i, U_i, \delta_i \rangle$.

5.2.1 Annotated Relations

Annotated relations are the flat, relational equivalents of TP-relations. Let TP-case statement $\gamma = \{\langle C_1, D_1, L_1, U_1, \delta_1 \rangle, \ldots, \langle C_n, D_n, L_n, U_n, \delta_n \rangle\}$. Then we can *"flatten"* TP-tuple $tp = (d, \gamma)$ by creating an *annotated tuple* for each time point $t \in \mathrm{sol}(C_1) \cup \ldots \cup \mathrm{sol}(C_n)$.

Each annotated tuple (at) provides probabilistic information ($[L_t, U_t]$) for one data tuple (d) at one point in time (t). According to the definition for a TP-case statement, $t \in \mathrm{sol}(C_i)$ for at most one $\gamma_i = \langle C_i, D_i, L_i, U_i, \delta_i \rangle \in \gamma$ since $(i \neq j) \Rightarrow (\mathrm{sol}(C_i) \cap \mathrm{sol}(C_j) = \emptyset)$. If $t \notin \bigcup_{\gamma_i \in \gamma} \mathrm{sol}(C_i)$, then we do not create an annotated tuple for t. We denote the set of all annotated tuples derived from tp by $\mathrm{ANN}(tp)$. In the worst case, $\mathrm{ANN}(tp)$ may contain $|S_\tau|$ tuples.

Definition 5.10 (annotated relation for a TP-tuple) Let $tp = (d, \gamma)$ be a TP-tuple over relational schema $(A_1, \ldots, A_k)$ and calendar τ where $d = (d_1, \ldots, d_k)$. Suppose γ contains n TP-cases of the form $\gamma_i = \langle C_i, D_i, L_i, U_i, \delta_i \rangle$ $(1 \leq i \leq n)$ and suppose τ consists of a linear hierarchy H containing m time units $T_1 \sqsubseteq \cdots \sqsubseteq T_m$. Here, each $t \in S_\tau$ will be of the form $t = (v_1, \ldots, v_m)$.

Then the *annotated relation for TP-tuple tp over calendar* τ, denoted $\mathrm{ANN}(tp)$, is defined as $\{(d, t, L_t, U_t) \mid t \in \mathrm{sol}(C_i)$ for some $\gamma_i \in \gamma$ and $[L_t, U_t] = [L_i \cdot x, U_i \cdot x]$ where $x = \delta_i(D_i, t)\}$. □

Intuitively, in the definition above, x represents the percentage of $\mathrm{sol}(D_i)$'s probability which is associated with time point t according to δ_i. Note that when we explicitly show all fields of an annotated tuple, $at = (d_1, \ldots, d_k, v_1, \ldots, v_m, L_t, U_t)$ is over the schema $(A_1, \ldots, A_k, T_1, \ldots, T_m, L_t, U_t)$. Here, A_1 to A_{k-1} are *manifest data fields*, A_k is the *hidden field*, T_1 to T_m are *temporal fields*, and L_t, U_t are *probabilistic fields.*

Definition 5.11 (annotated relation for a TP-relation) Let r be a TP-relation over τ containing n TP-tuples $tp_1 \ldots tp_n$. Then the *annotated relation for TP-relation r over calendar* τ, denoted $\mathrm{ANN}(r)$, is defined as the multiset $(\mathrm{ANN}(tp_1) \uplus \ldots \uplus \mathrm{ANN}(tp_n))$ over τ.

We associate with each $\mathrm{ANN}(r)$ the *primary key* which is associated with r. This key will be used when we describe the TATA's projection operation.

5.2.2 Consistency of TP- and Annotated relations

We are now ready to define the formal semantics of TP-relations. In order to provide such a semantics, we will extend classical logic [48] to the case of TP-relations, by extending the concept of an interpretation in classical logic [48] to handle TP-relations. Before doing this, a preliminary definition is needed.

Definition 5.12 (data-identical TP- and annotated tuples) TP-tuples $tp = (d, \gamma)$ and $tp' = (d', \gamma')$ are *data-identical* iff $(d = d')$. Note that tp and tp' may come from different TP-relations as long as both TP-relations have the same schema. Also, note that $(d = d')$ only if $(d.\mathrm{H} = d'.\mathrm{H})$. □

Annotated tuples $at = (d, t, L_t, U_t)$ and $at' = (d', t', L'_t, U'_t)$ are *data-identical* iff $(d = d')$.

Recall that without loss of generality, we interpret both TP-relations and annotated relations under the assumption that all data-identical TP-tuples (annotated tuples) refer to the same, unique event. If tp and tp' are data-identical, we assume that they provide complementary information for the same event. If tp and tp' are not data-identical, we assume that they refer to different events. Let $tp \in r$. Then $r[tp]$ denotes the multiset of all TP-tuples in r which are data-identical to tp. Since "data-identical" is a reflexive, symmetric, transitive relation on TP-tuples, it is also an equivalence relation on r where each $r[tp]$ corresponds to an equivalence class in this relation. The $ANN(r)[at]$ notation is defined in a similar manner.

A TP-relation r is *compact* iff for each data tuple d and each time point t there is at most one TP-tuple $tp = (d, \gamma) \in r$ where $t \in \mathrm{sol}(C_1 \vee \ldots \vee C_n)$. Otherwise, since r contains at least two TP-tuples which refer to the same event at the same time, r is an *uncompact* TP-relation. Similarly an annotated relation $ANN(r)$ is *compact* iff any two data-identical tuples in it have *different* timepoints. Otherwise $ANN(r)$ is *uncompact* Later, we will describe a variety of *compaction* operators which convert uncompact TP- and annotated relations into compact ones by consolidating probabilistic information for each $r[tp] \subseteq r$ or $ANN(r)[at] \subseteq ANN(r)$.

The following theorem states that the concept of "compact relation" for TP-relations and annotated relations coincide.

Theorem 15 *A TP-relation r is compact iff its annotated counterpart,* $\mathrm{ANN}(r)$*, is compact.*

Intuitively, a TP-tuple $tp = (d, \gamma)$ is *consistent* if there exists a satisfying assignment of probabilities for each TP-case $\gamma_i \in \gamma$. This is given formal "teeth" through the following definition.

Definition 5.13 (TP-interpretation) Let $A = (A_1, \ldots, A_k)$ be a relational schema, let τ be a calendar, and let $dom(A) = dom(A_1) \times \cdots \times dom(A_k)$ be the domain of A. Then a *TP-interpretation* over the pair A, τ is a function $I_{A,\tau} : dom(A) \times S_\tau \mapsto [0, 1]$ such that $(\forall d \in dom(A))(\sum_{t \in S_\tau} I_{A,\tau}(d, t) \leq 1)$. $\square$

Let e be the event represented by data tuple d. Then $I_{A,\tau}(d, t) = p$ says that according to TP-interpretation $I_{A,\tau}$, the probability that e is true at time point t is p. Let D be a temporal constraint over τ. Then the *probability assigned by $I_{A,\tau}$ to D*, denoted $I_{A,\tau}(d, D)$, is equal to $\sum_{t \in \mathrm{sol}(D)} I_{A,\tau}(d, t)$. This intuition may be used to explain what it means for a TP-interpretation to satisfy a TP-tuple.

Definition 5.14 (satisfaction) Let d be a tuple in relational schema A and let $\gamma_i = \langle C_i, D_i, L_i, U_i, \delta_i \rangle$ be a TP-case. Then $I_{A,\tau}$ *satisfies* $\langle d, \gamma_i \rangle$, denoted $I_{A,\tau} \models \langle d, \gamma_i \rangle$, iff the following conditions hold:

1. $L_i \leq I_{A,\tau}(d, D_i) \leq U_i$, i.e. the probability that $I_{A,\tau}$ assigns to D_i lies in the interval $[L_i, U_i]$.
2. $(\forall t \in \mathrm{sol}(C_i))(I_{A,\tau}(d, D_i) \cdot \delta_i(D_i, t) = I_{A,\tau}(d, t))$, i.e. $I_{A,\tau}$ distributes probabilities for each $t \in \mathrm{sol}(C_i)$ according to δ_i.

TP-interpretation $I_{A,\tau}$ *satisfies* TP-tuple $tp = (d, \gamma)$, denoted $I_{A,\tau} \models tp$, iff $I_{A,\tau} \models \langle d, \gamma_i \rangle$ for all $\gamma_i \in \gamma$. $\square$

Definition 5.15 (consistency and mutual consistency) A TP-tuple tp is *consistent* iff there exists a TP-interpretation $I_{A,\tau}$ where $I_{A,\tau} \models tp$. A TP-relation r is *consistent* iff $(\exists I_{A,\tau})(\forall tp \in r)(I_{A,\tau} \models tp)$. TP-relations r and r' are *mutually consistent* iff $(\exists I_{A,\tau})((\forall tp \in r)(I_{A,\tau} \models tp) \wedge (\forall tp' \in r')(I_{A,\tau} \models tp'))$. Note that if consistent TP-relations r, r' have different schemas, then r and r' must be mutually consistent. $\square$

Later in this paper, we will provide algorithms to convert any TP-relation into a *compact* TP-relation. When a TP-relation is compact, there are no two TP-tuples that are data-identical, and hence, we can check consistency of a TP-relation by individually checking consistency of each TP-tuple. Suppose a TP-tuple $tp = (d, \gamma)$ has $\gamma = \{\langle C_1, D_1, L_1, U_1, \delta_1\rangle, \ldots, \langle C_n, D_n, L_n, U_n, \delta_n\rangle\}$ as its TP-case statement. Then it suffices to check that

$$(L_1 + \cdots + L_n) \quad \leq \quad \min(1, \Sigma_{i=1}^{n} L_i \cdot \Sigma_{t \in S_\tau} \delta_i(D_i, t)).$$

If this condition holds, then the TP-tuple is consistent. This forms the basis for the following claim. When the distribution function used is determinate, then the quantity on the right side of the above inequality can be computed in linear time because the determinacy condition guarantees constant time computation of the sum $\Sigma_{t \in S_\tau} \delta_i(D_i, t))$.

Proposition 4 *Checking consistency of a compact TP-relation which uses determinate PDFs is linear in the size of the TP-relation.*

Now we define what is means for an annotated relation to be satisfied by a TP-interpretation.

Definition 5.16 (satisfaction of annotated tuples) Let d be a tuple in relational schema A, let t be a time point in S_τ, and let $[L, U]$ be a probability interval. Then a TP-interpretation $I_{A,\tau}$ *satisfies* annotated tuple $at = (d, t, L_t, U_t)$, denoted $I_{A,\tau} \models at$, iff $L_t \leq I_{A,\tau}(d, t) \leq U_t$. $\square$

Definition 5.17 (consistency of annotated relations) An annotated tuple at is *consistent* iff $(\exists I_{A,\tau})(I_{A,\tau} \models at)$. An annotated relation $\mathrm{ANN}(r)$ is *consistent* iff $(\exists I_{A,\tau})(\forall at \in \mathrm{ANN}(r))(I_{A,\tau} \models at)$. Annotated relations $\mathrm{ANN}(r)$ and $\mathrm{ANN}(r')$ are *mutually consistent* iff $(\exists I_{A,\tau})((\forall at \in \mathrm{ANN}(r))(I_{A,\tau} \models at) \wedge (\forall at' \in \mathrm{ANN}(r'))(I_{A,\tau} \models at'))$. $\square$

The following theorem tells us that if r is a consistent TP-relation, then $\mathrm{ANN}(r)$ is also consistent.

Theorem 16 *Let r be a TP-relation. If $I_{A,\tau}$ satisfies r, then $I_{A,\tau}$ also satisfies* $\mathrm{ANN}(r)$. *Hence if r is consistent, so is* $\mathrm{ANN}(r)$.

The converse of this theorem is not true, i.e., it may be the case that a TP-interpretation satisfies $\mathrm{ANN}(r)$, but does not satisfy r. This is shown in the following example.

Example 5.7 (satisfaction) Let r consist of one TP-tuple (d, γ) where $\gamma = \{\langle(\#), (1 \sim 2), 0.4, 0.8, u\rangle\}$. Then $\mathrm{ANN}(r) = \{at_1, at_2\}$ where $at_1 = (d, 1, 0.2, 0.4)$ and $at_2 = (d, 2, 0.2, 0.4)$.

Now consider the TP-interpretation $I_{A,\tau}$ such that $I_{A,\tau}(d, 1) = 0.3$ and $I_{A,\tau}(d, 2) = 0.4$. Clearly, $I_{A,\tau}$ satisfies $\mathrm{ANN}(r)$, but $I_{A,\tau}$ does not satisfy r because every TP-interpretation $J_{A,\tau}$ that satisfies r must have $J_{A,\tau}(d, 1) = J_{A,\tau}(d, 2)$.

This occurs because the details of the distribution get lost when annotating a relation — this is not surprising as annotated relations have no fields for including information about distributions.

Instead, we can show that if r is compact, if $(d, \gamma) \in r$, and if $(d, t, L_t, U_t) \in \mathrm{ANN}(r)$, then there must be a TP-interpretation $I_{A,\tau}$ of r such that $I_{A,\tau}(d, t) = L_t$. A similar statement applies to U_t. This means that the bounds contained in ANN(r) are tight, and hence, ANN(r) correctly captures the implied probability intervals for data tuple d at time t.

Theorem 17 *Let r be a compact TP-relation containing a TP-tuple (d, γ), and suppose $(d, t, L_t, U_t) \in$ ANN(r). Then there is a TP-interpretation $I_{A,\tau}$ satisfying r such that $I_{A,\tau}(d, t) = L_t$.*

Theorems 16 and 17 jointly tells us that as far as lower bounds are concerned, r and ANN(r) are equivalent when r is known to be compact. Later, we will describe mechanisms to make compact a TP-relation r.

Note that the above result does not hold for upper bounds — the reason for this is that in a TP-tuple, the upper bounds may often be loose (i.e., not tight). For instance, consider the following TP-tuple:

Data	H	C	D	L	U	δ
D1		(#)	(5/1/1998)	0.6	1	u
		(#)	(6/1/1998)	0.4	1	u

It is easy to see that the upper bounds of the TP-cases above can be tightened to 0.6 and 0.4 respectively. Hence, these upper bounds are "loose" and need to be tightened if a theorem similar to Theorem 17 is to hold.

Definition 5.18 (tightening) Let $\gamma = \{\gamma_1, \ldots, \gamma_n\}$ be a TP-case statement where each TP-case $\gamma_i = \langle C_i, D_i, L_i, U_i, \delta_i \rangle$. A *tightening* of γ returns a TP-case statement $\gamma'' = \{\gamma_1'', \ldots, \gamma_n''\}$ where each $\gamma_i'' = \langle C_i'', D_i'', L_i'', U_i'', \delta_i'' \rangle$ and each $U_i'' \leq U_i$ for all $1 \leq i \leq n$.

A TP-tuple (d, γ) is said to be *tight* iff there is no other TP-tuple (d, γ') such that: (i) γ' is a tightening of γ and (ii) for all TP-interpretations $I_{A,\tau}$, $I_{A,\tau} \models (d, \gamma)$ iff $I_{A,\tau} \models (d, \gamma')$.

A TP-relation is *tight* iff every TP-tuple in it is tight. □

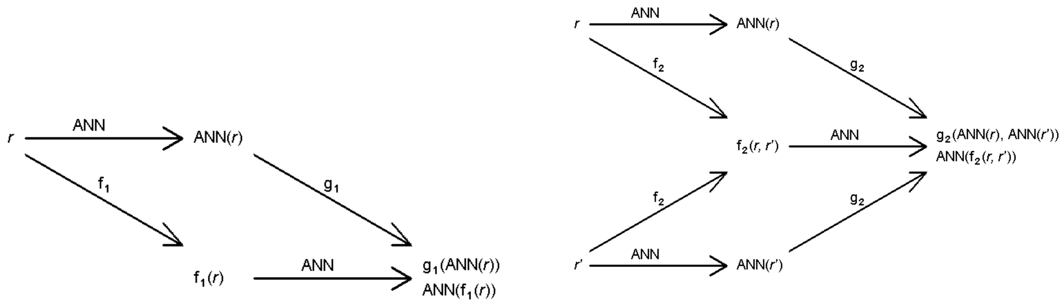
The following theorem tells us that for tight and compact TP-relations, Theorem 17 holds for upper bounds as well.

Theorem 18 *Let r be a compact, tight, TP-relation containing a TP-tuple (d, γ), and suppose $(d, t, L_t, U_t) \in$ ANN(r). Then there is a TP-interpretation $I_{A,\tau}$ satisfying r such that $I_{A,\tau}(d, t) = U_t$.*

Theorems 17 and 18 jointly tell us that the conversion of a TP-relation r to annotated form preserves bounds when r is tight and compact.

5.3 Relational Algebras (TATA and TPA)

The key difference between the TP Databases and ProbView framework is in the fact that in ProbView model, the relational algebra had been described only for the annotated relations. In



(a) Unary Operators (b) Binary Operators

Figure 1: Correct Implementation of TATA operations by TPA operations

fact at that time the creators of ProbView wrote "... *it is unclear how a generic algebra not dependent on any specific probabilistic strategy can be defined based on probabilistic relations...*" ([36], p. 429).

On the other hand, one of key purposes for developing TP Databases was to keep information stored in compressed way and try to manipulate it in this compressed (TP-relations) format. To a certain extent, our work on relational algebra for TP Databases solves that problem stated in the quote above, only for a somewhat modified syntax.

It should be noted that as annotated relations were introduced to represent the meaning of TP-relations, it makes sense to develop two relational algebra: one on annotated relations, and one – on TP-relations. Then our goal would be to show that there is a commutative diagram between the two algebras: i.e. no matter in which order a relational algebra operation and annotation are applied to a TP-relation, the result will not change.

This is exactly the course of action we are taking. We will define two Relational algebras: Theoretical Annotated Temporal Algebra (TATA) and Temporal Probabilistic Algebra (TPA). We should stress again that the operations of TATA are only of theoretical importance, as they are meant to provide the semantics for the operations of TPA - the Relational Algebra which is intended to be implemented (and is indeed implemented).

First we formalize our notion of a "commutative diagram" between the two algebras proposed.

Definition 5.19 (correct implementation) Unary TPA operator op^T *correctly implements* the semantics for unary TATA operator op^A iff $\mathrm{ANN}(op^T(r)) = op^A(\mathrm{ANN}(r))$ for every TP-relation r.

Furthermore, binary TPA operator op^T *correctly implements* the semantics for binary TATA operator op^A iff $\mathrm{ANN}(r\ op^T\ r') = \mathrm{ANN}(r)\ op^A\ \mathrm{ANN}(r')$ for every pair of TP-relations r, r'. □

These relationships are best illustrated by the diagrams on Figure 1.

We start our description of both algebras with the *compaction* operation, as this operation will become a part of a number of other operations in both TATA and TPA.

5.3.1 Compaction

Compaction can be thought of as the TP analog or extension of duplicate elimination in the relational algebra. The difference between the two operations is in the fact that in relational algebra two tuples are either equivalent or the are not. In TP-case we separate between *data-identical* and *not data-identical tuples* (both annotated and TP). As it follows from Def. 5.12 two data-identical tuples can have different probabilities associated with the same timepoints. The task of *compaction* operation will be to "reconcile" all data-identical tuples in the relation.

While we have a *preferred* way of performing such reconciliation (via intersection of all the intervals from all data-identical tuples at the same timepoint) we understand that this may not be the method of choice for some users and some applications. Another issue that arises when intersection is considered is the treatment of cases when it returns $\emptyset$ (i.e. when there are non-intersecting intervals given as probabilities of the same event at a certain time). Recall from Section 3.2 that in this case in our HPLP framework we would just declare an inconsistency.

A different approach, however, should be taken for the database environment. This approach consists in defining a class of functions that satisfy certain properties. Each function of such a class would be an appropriate "core" of a compaction operation, i.e., when compaction is performed, the resulting interval is obtained from the intervals from all data-identical tuples by applying such a function. We call these functions *combination functions.*

Definition 5.20 (combination function) Let $S = \{[L_1, U_1], \ldots, [L_k, U_k]\}$ be a non-empty multiset of probabilistic intervals. Then a *combination function* χ is a function which takes S as input, and returns as output a probabilistic interval $[L, U]$ which satisfies the following axioms:

1. Identity: If $[L_1, U_1] = \ldots = [L_k, U_k]$, then $\chi(S) = [L_1, U_1]$. In other words, when all input intervals are equal, the output interval is also equal to all of the input intervals.
2. Bottomline: $L \leq \max\{L_i \mid [L_i, U_i] \in S\}$. In other words, the lower bound of the result cannot exceed the largest lower bound of the intervals in S. □

The axiom of Identity tells us that whenever all intervals in the multiset S are the same, the value returned by χ should not be different. This property will assure that our compaction operation will be a proper extension of duplicate elimination operation from classical relational algebra.

The axiom of Bottomline states that the lower bound of the returned interval cannot exceed the maximum lower bound of the intervals in S. This property will guarantee us that if a TP (Annotated) relation was consistent before compaction operation, the compaction will not change that.

We can now describe our "preferred class" of combination functions.

Definition 5.21 (conflict) A multiset S of probability intervals *conflict* iff $\bigcap_{[L,U]\in S}[L, U] = \emptyset$. □

Note that all combination functions must find a way to remove (resolve) conflicts. A class of combination functions called *equity combination functions* prescribe to the view that if $S = \{[L_1, U_1], [L_2, U_2]\}$ does not conflict, then $\chi(S)$ should equal $[L_1, U_1] \cap [L_2, U_2]$. However, if these intervals conflicted, then different equity combination functions may resolve the conflict in different ways.

Definition 5.22 (equity combination function) An equity combination function χ_e is a combination function where $(\bigcap_{[L,U]\in S}[L,U] \neq \emptyset) \Rightarrow (\chi_e(S) = \bigcap_{[L,U]\in S}[L,U])$. □

The following example shows a variety of equity combination functions.

Example 5.8 (example equity combination functions) :

Name	Interval Returned when $\bigcap_{[L,U]\in S}[L,U] = \emptyset$
Optimistic Equity	$\chi_{eq}(S) = [\max(\{L_i \mid [L_i,U_i] \in S\}), \max(\{U_i \mid [L_i,U_i] \in S\})]$
Enclosing Equity	$\chi_{ec}(S) = [\min(\{L_i \mid [L_i,U_i] \in S\}), \max(\{U_i \mid [L_i,U_i] \in S\})]$
Pessimistic Equity	$\chi_{ep}(S) = [\min(\{L_i \mid [L_i,U_i] \in S\}), \min(\{U_i \mid [L_i,U_i] \in S\})]$
Rejecting Equity	$\chi_{er}(S) = [0,0]$
Skeptical Equity	$\chi_{esk}(S) = [0,1]$
Quasi-independence Equity	$\chi_{eqi}(S) = [\Pi_{[L_i,U_i]\in S} L_i, \Pi_{[L_i,U_i]\in S} U_i]$

Note that when $\bigcap_{[L,U]\in S}[L,U] \neq \emptyset$, all of the functions above return $\bigcap_{[L,U]\in S}[L,U]$.

Example 5.9 *Consider a equity function $chi_{eac}(S) = [1,1]$ whenever $\bigcap_{[L,U]\in S}[L,U] = \emptyset$. It is easy to see that this function is not a valid combination function, because it violates the* Bottomline *axiom.*

Now we can describe the use of combination functions in compaction operations.

Definition 5.23 (Compaction of an annotated relation (TATA)) A function κ from annotated relations to annotated relations is called a *compaction operation* if it satisfies the following axioms:

- Compactness : $\kappa(\text{ANN}(r))$ is *compact* for all annotated relations $\text{ANN}(r)$.
- No Fooling Around (NFA) : If $\text{ANN}(r)$ is compact, then $\kappa(\text{ANN}(r)) = \text{ANN}(r)$.
- Conservativeness : If $at = (d,t,L_t,U_t) \in \kappa(\text{ANN}(r))$, then $\exists at' = (d,t,L'_t,U'_t) \in \text{ANN}(r)$. □

Definition 5.24 (Compaction of a TP-relation (TPA)) A function κ from TP-relations to TP-relations is called a *compaction operation* if it satisfies the following axioms:

- Compactness : $\kappa(r)$ is *compact* for all TP-relations r.
- No Fooling Around (NFA) : If r is compact then $\text{ANN}(\kappa(r)) = \text{ANN}(r)$.
- Conservativeness : If $at = (d,t,L_t,U_t) \in \text{ANN}(\kappa(r))$, then $\exists at' = (d,t,L'_t,U'_t) \in \text{ANN}(r)$. □

The Compactness axiom assures us that the result of a compaction operation will be a compact relation. The NFA axiom states that applying compaction operation to a compact relation should not change the relation. The Compactness and NFA axioms jointly guarantee that compaction operations are idempotent, i.e. $\kappa(\kappa(\text{ANN}(r))) = \kappa(\text{ANN}(r))$. The Conservativeness axiom says that any information which appears in the result of a compaction has to originate from information in the initial relation; no information about "new" events, or events at "new" time points gets added during compaction.

It should be clear that there are many possible ways to compact a relation. One possible way is to use the combination functions to do the computation of the resulting intervals.

Definition 5.25 (χ-compaction of an annotated relation (TATA)) Let χ be a combination function. Then the
χ-*compaction of annotated relation* $\mathrm{ANN}(r)$, denoted $\kappa_\chi(\mathrm{ANN}(r))$, is defined as $\kappa_\chi(\mathrm{ANN}(r)) =$
$\{at = (d,t,L_t,U_t) \mid [L_t,U_t] = \chi(\{[L_1^{(d,t)}, U_1^{(d,t)}], \ldots, [L_k^{(d,t)}, U_k^{(d,t)}]\})\}$ where
$\mathrm{ANN}(r)[d,t] = \{at_1^{(d,t)}, \ldots at_k^{(d,t)}\}$ and $at_i^{(d,t)} = (d,t,L_i^{(d,t)}, U_i^{(d,t)})$. □

Definition 5.26 (χ-compaction of a TP-relation (TPA)) Let χ be a combination function. Then the
χ-*compaction of TP-relation* r, denoted $\kappa_\chi(r)$, is defined as $\mathrm{ANN}(\kappa_\chi(r)) =$
$\{at = (d,t,L_t,U_t) \mid [L_t,U_t] = \chi(\{[L_1^{(d,t)}, U_1^{(d,t)}], \ldots, [L_k^{(d,t)}, U_k^{(d,t)}]\})\}$ where
$\mathrm{ANN}(r)[d,t] = \{at_1^{(d,t)}, \ldots at_k^{(d,t)}\}$ and $at_i^{(d,t)} = (d,t,L_i^{(d,t)}, U_i^{(d,t)})$. □

We notice that TPA definitions of compaction directly emulate TATA. Because these definitions are declarative, while TPA is thought to be the operational semantics of the TP Databases, we need to provide an algorithm that would correctly compute compaction on the TP-relations.

Algorithm Compute-Compaction(r,f):
Input: TP-relation r and combination function f
Output: TP-relation $r'' = \kappa_f(r)$
$r'' := \emptyset$; `// Initialize the resulting relation`
$r' := r$; `// Obtain a working copy of the initial relation`
`// For each (maximal) multiset S of data-identical TP-tuples in r`
while $(r' \neq \emptyset)$ **do** {
  Select a TP-tuple $tp \in r'$;
  $S := r'[tp]$; `// Extract the next equivalence class from r'`
  $r' := r' - S;\quad \gamma'' := \emptyset$;
  $\Gamma_S := \biguplus_{(d,\gamma)\in S} \gamma;\quad \Gamma_I := \Gamma_S$;
  foreach $\gamma_i = \langle C_i, D_i, L_i, U_i, \delta_i\rangle \in \Gamma_I$ {
    Remove γ_i from Γ_I and Γ_S; $C_S := \bigvee_{\langle C,D,L,U,\delta\rangle \in \Gamma_S} C$;
    if $\mathrm{sol}(C_i \wedge \neg C_s) \neq \emptyset$ **then** Add $\langle (C_i \wedge \neg C_s), D_i, L_i, U_i, \delta_i\rangle$ to γ'';
    if $\mathrm{sol}(C_i \wedge C_s) \neq \emptyset$ **then** Add $\langle (C_i \wedge C_s), D_i, L_i, U_i, \delta_i\rangle$ to Γ_S; }
  $C_S := \bigvee_{\langle C,D,L,U,\delta\rangle \in \Gamma_S} C$;
  `// Note: Each` $t \in \mathrm{sol}(C_S)$ `will refer to more than one TP-case`
  foreach $t \in \mathrm{sol}(C_S)$ {
    $X := \emptyset$; `//` X `will contain the probability intervals to be combined`
    $\Gamma_t := \{\langle C, D, L, U, \delta\rangle \in \Gamma_S \mid t \in \mathrm{sol}(C)\}$;
    foreach $\langle C, D, L, U, \delta\rangle \in \Gamma_t$ {
      $x_t := \delta(D,t)$; $[L_t, U_t] := [L \cdot x_t, U \cdot x_t]$;
      $X := X \cup \{[L_t, U_t]\}$; }
    $\boxed{[L''_t, U''_t] := f(X);}$
    Add TP-case $\langle (\#), (t), L''_t, U''_t, u\rangle$ to γ''; }
  Add TP-tuple (d, γ'') to r''; }
return r'';
End-Algorithm

The following states that this algorithm correctly computes a $\kappa_\chi(r)$ compaction.

Theorem 19 *Let χ be a combination function. Then algorithm* **Compute-Compaction**(r, χ) *correctly computes the $\kappa_\chi(r)$ compaction operation.*

5.3.2 Intersection and Union

Both intersection and union operations will consist of two suboperations: multiset intersection (union) and compaction.

While the general intuition behind the union pretty much defines the TATA and TPA operations, things are a little bit more tricky with intersection. We want to think of intersection as an attempt to combine together the *common knowledge* about events contained in two relations. Such common knowledge in our setting would be the information that event described by the data part of tp-tuple (or annotated tuple) could happen at a certain timepoint.

With this approach the definitions of intersection operation will look as follows.

Definition 5.27 (multiset intersection of two annotated relations (TATA)) The *multiset intersection of annotated relations ANN(r) and ANN(r′)*, denoted $\mathrm{ANN}(r) \cap \mathrm{ANN}(r')$, is defined as $\mathrm{ANN}(r'') = \{at \in \mathrm{ANN}(r) \mid (\exists at' \in \mathrm{ANN}(r'))(d = d' \wedge t = t')\} \cup \{at' \in \mathrm{ANN}(r') \mid (\exists at \in \mathrm{ANN}(r))(d = d' \wedge t = t')\}$. □

$\mathrm{ANN}(r'')$ will contain all $at \in \mathrm{ANN}(r)$ and all $at' \in \mathrm{ANN}(r')$ where at and at' refer to the same event at the same point in time.

Definition 5.28 (multiset intersection of two TP-relations (TPA)) The *multiset intersection of TP-relations r and r'*, denoted $r \cap r'$, can be constructed in the following way: Initially, let $r'' = \emptyset$. Then for each $tp = (d, \gamma) \in r$ and each $tp' = (d', \gamma') \in r'$ where $(d = d')$,

1. Let $\Gamma = \Gamma' = \emptyset$.
2. For each $\gamma_i \in \gamma$ and each $\gamma'_j \in \gamma'$ where $|\mathrm{sol}(C_i \wedge C'_j)| \geq 1$, add TP-case $\langle (C_i \wedge C'_j), D_i, L_i, U_i, \delta_i \rangle$ to Γ and add TP-case $\langle (C_i \wedge C'_j), D'_j, L'_j, U'_j, \delta'_j \rangle$ to Γ'. Note that $(C_i \wedge C'_j)$ is shared by both TP-cases.
3. If $\Gamma \neq \emptyset$, add TP-tuples (d, Γ) and (d, Γ') to r''. Note that $\Gamma \neq \emptyset \Rightarrow \Gamma' \neq \emptyset$. Γ will be empty if there are no overlapping time points. □

Multiset intersection operations will produce relations which are not compact as it is easy to see (as with each tuple from one original relation included in the result of this operation a data-identical tuple from another relation is also included). Applying compaction operation after the multiset intersection will solve this problem.

Definition 5.29 (intersection of two annotated relations (TATA)) The *intersection of annotated relations ANN(r) and ANN(r′) under the χ combination function*, denoted $\mathrm{ANN}(r) \cap_\chi \mathrm{ANN}(r')$, is defined as $\kappa_\chi(\mathrm{ANN}(r) \cap \mathrm{ANN}(r'))$. □

Definition 5.30 (intersection of two TP-relations (TPA)) The *intersection of TP-relations r and r' under the χ combination function*, denoted $r \cap_\chi r'$, is defined as $\kappa_\chi(r \cap r')$. □

The definitions for multiset union and union operations are similar in spirit.

Definition 5.31 (multiset union of two annotated relations (TATA)) The *multiset union of annotated relations ANN(r) and ANN(r′)*, denoted $\mathrm{ANN}(r) \cup \mathrm{ANN}(r')$, is defined as $\mathrm{ANN}(r'') = \mathrm{ANN}(r) \uplus \mathrm{ANN}(r')$. $\square$

Definition 5.32 (multiset union of two TP-relations (TPA)) The *multiset union of TP-relations r and r'*, denoted $r \cup r'$, is defined as $r'' = r \uplus r'$. $\square$

Definition 5.33 (union of two annotated relations (TATA)) The *union of annotated relations ANN(r) and ANN(r′) under the χ combination function*, denoted $\mathrm{ANN}(r) \cup_\chi \mathrm{ANN}(r')$, is defined as $\kappa_\chi(\mathrm{ANN}(r) \cup \mathrm{ANN}(r'))$. $\square$

Definition 5.34 (union of two TP-relations (TPA)) The *union of TP-relations r and r' under the χ combination function*, denoted $r \cup_\chi r'$, is defined as $\kappa_\chi(r \cup r')$. $\square$

5.3.3 Selection

There are three distinctive types of selects on our model. First, an analog of the select in classical relational algebra is a select based on the conditions which apply to the data part of the relation. For TP-relations this operation will select complete tp-tuples. Second, is a temporal select, which selects parts of a relation which satisfy certain temporal constraints. When such an operation is applied to TP-relations, only parts of tp-tuples may be selected. Finally there are probabilistic selects which look for the data which has the specified probabilistic properties (such as the probability being in a certain range, or some other such conditions). Again, only parts of tp-tuples may be selected as a result of such operation. There are also mixed selects which combine the conditions of two or more types, but we will first address the three basic types of selections and then show how they are combined.

Definition 5.35 (select condition)

- Any atomic temporal constraint (see Def. 5.2) is an *atomic temporal select condition.*
- Any expression of the form $A\mathbf{op}v$ where A is a name of some attribute in a TP-database, and $v \in dom(A)$ is an *atomic data select condition.*
- Any expression of the form $L\mathbf{op}x$ or $U\mathbf{op}x$ where $x \in [0,1]$ and L and U are two reserved attribute names is an *atomic probabilistic select condition*

An *atomic select condition* is either a temporal atomic select condition, or data atomic select condition or probabilistic atomic select condition.

A select condition is any boolean combination of atomic select conditions. We distinguish between (i) *data select conditions* which consist only of atomic data select conditions, (ii) *temporal select condition*, which consist only of atomic temporal select conditions, (iii) *probabilistic select conditions* which consist only of atomic probabilistic select conditions, and, finally (iv) *mixed select conditions*, which contain atomic conditions of more than one type.

We first define selection on atomic conditions.

Definition 5.36 (selection on an annotated relation; atomic condition (TATA)) The *selection of atomic condition $\mathcal{C}$ on annotated relation ANN(r)*, denoted $\sigma_{\mathcal{C}}(\text{ANN}(r))$, is defined in the following way:

- If $\mathcal{C}$ is a data condition, $\text{ANN}(r'') = \{at \in \text{ANN}(r) \mid d \text{ satisfies } \mathcal{C}\}$.
  In this case, our selection is based on the classical relational algebra.
- If $\mathcal{C}$ is a temporal condition, $\text{ANN}(r'') = \{at \in \text{ANN}(r) \mid t \in \text{sol}(\mathcal{C})\}$.
- If $\mathcal{C}$ is a probabilistic condition, $\text{ANN}(r'') = \{at \in \text{ANN}(r) \mid ([L, U] = [L_t, U_t]) \text{ satisfies } \mathcal{C}\}$. □

Before defining the TPA version of selection we need to define one more function which will be used for probabilistic select.

Definition 5.37 (TP-filter) Let $\gamma_i = \langle C_i, D_i, L_i, U_i, \delta_i \rangle$ be a TP-case, let $\mathcal{C}$ be a probabilistic condition, and let $x = L_i$ if $F =$ "L" or let $x = U_i$ otherwise. Then a *TP-filter* is a function which takes γ_i and $\mathcal{C}$ as input, and returns as output a temporal constraint C_i'' where

1. $\text{sol}(C_i'') \subseteq \text{sol}(C_i)$
2. For each time point $t \in \text{sol}(C_i'')$, $\delta_i(D_i, t) \cdot [L_i, U_i]$ *satisfies* $\mathcal{C}$.
3. There is no temporal constraint C_i' where $(\text{sol}(C_i') \supset \text{sol}(C_i''))$ for which conditions 1 and 2 hold.

Intuitively, a TP-filter returns a temporal constraint whose solution set consists of all time points $t \in \text{sol}(C_i)$ where $[L_t, U_t] = [L_i \cdot \delta_i(D_i, t), U_i \cdot \delta_i(D_i, t)]$ satisfies $\mathcal{C}$. If no $t \in \text{sol}(C_i)$ satisfies this condition, $\text{TP-filter}(\gamma_i, \mathcal{C})$ returns an inconsistent temporal constraint. □

Definition 5.38 (Selection on a TP-tuple; atomic condition (TPA)) The *selection of atomic condition $\mathcal{C}$ on TP-tuple $tp = (d, \gamma)$*, denoted $\sigma_{\mathcal{C}}(tp)$, can be constructed in the following way: Initially, let $\gamma'' = \emptyset$.

- If $\mathcal{C}$ is a data condition, let $\gamma'' = \gamma$ if d satisfies $\mathcal{C}$.
- If $\mathcal{C}$ is a temporal condition, then for each $\gamma_i \in \gamma$ where $C_i'' = (C_i \wedge \mathcal{C})$ is consistent, add TP-case $\langle C_i'', D_i, L_i, U_i, \delta_i \rangle$ to γ''.
- If $\mathcal{C}$ is a probabilistic condition, then for each $\gamma_i \in \gamma$ where $C_i'' = \text{TP-filter}(\gamma_i, \mathcal{C})$ is consistent, add TP-case $\langle C_i'', D_i, L_i, U_i, \delta_i \rangle$ to γ''.

If $\gamma'' = \emptyset$ then $\sigma_{\mathcal{C}}(tp) = \emptyset$. Otherwise, $\sigma_{\mathcal{C}}(tp) = (d, \gamma'')$. □

Definition 5.39 (Selection on a TP-relation; atomic condition (TPA)) The *selection of atomic condition $\mathcal{C}$ on TP-relation $r = \{tp_1, \ldots, tp_n\}$*, denoted $\sigma_{\mathcal{C}}(r)$, is defined as $(\sigma_{\mathcal{C}}(tp_1) \uplus \ldots \uplus \sigma_{\mathcal{C}}(tp_n))$. □

Definition 5.40 (Selection on a TP-relation) The *selection of condition $\mathcal{C}$ on TP-relation r*, denoted $\sigma_{\mathcal{C}}(r)$, is defined inductively in the following way:

- If $\mathcal{C}$ is an atomic condition, then $r'' = \sigma_{\mathcal{C}}(r)$ by way of our previous definition.
- If $\mathcal{C}$ is of the form $(\mathcal{C}_1 \wedge \mathcal{C}_2)$, then $r'' = \sigma_{\mathcal{C}_1}(\sigma_{\mathcal{C}_2}(r))$.
- If $\mathcal{C}$ is of the form $(\mathcal{C}_1 \vee \mathcal{C}_2)$, then $r'' = \sigma_{\mathcal{C}_1}(r) \cup_{eq} \sigma_{\mathcal{C}_2}(r)$. (Note that as long as r is compact, it follows by the Identity axiom that irrespective of which combination function is used, we obtain the same results, i.e. "eq" in the above definition can be replaced with any other combination without the result being changed.)
- If $\mathcal{C}$ is of the form $(\neg\mathcal{C}_1)$, then
  - If $\mathcal{C}_1$ is of the form $(\mathcal{C}_2 \wedge \mathcal{C}_3)$, $(\mathcal{C}_2 \vee \mathcal{C}_3)$, or $(\neg\mathcal{C}_2)$, then $r'' = \sigma_{\mathcal{C}_4}(r)$ where $\mathcal{C}_4 = (\neg\mathcal{C}_2 \vee \neg\mathcal{C}_3)$, $\mathcal{C}_4 = (\neg\mathcal{C}_2 \wedge \neg\mathcal{C}_3)$, or $\mathcal{C}_4 = (\mathcal{C}_2)$ respectively.
  - If $\mathcal{C}_1$ is a *data*, *temporal*, or *probabilistic* condition, then $r'' = \sigma_{\mathcal{C}_4}(r)$ where $\mathcal{C}_4$ is the atomic, logical negation of $\mathcal{C}_1$.

To define selections with non-atomic conditions on annotated relations, all we need to do is replace in the definition above all instances of r and r'' with $\mathrm{ANN}(r)$ and $\mathrm{ANN}(r'')$ respectively.

The following theorem states that selections are well-defined.

Theorem 20 $\sigma_{\mathcal{C}_1}(\sigma_{\mathcal{C}_2}(r)) = \sigma_{\mathcal{C}_2}(\sigma_{\mathcal{C}_1}(r))$.

5.3.4 Difference

Difference is probably the hardest operation to define intuitively in a probabilistic setting. After all how *does* one define the difference between "Rain will occur on May 25 with probability 60% " and "Rain will occur on May 25 with probability 20%" ??? Dey and Sarkar ([14]) e.g., would have inserted the statement "Rain will occur on May 25 with probability 40%" (40% = 60% − 20%) into the result of difference. While 40% is the difference between the probabilities estimated for rain on May 25 by two relations, can we really conclude from this that 40% *is indeed the probability of "Rain will occur on May 25" event* ? Most certainly, not (In fact, according to our "preferred" way of looking at things these two statements are in contradiction).

Lakshmanan *et al.*in [36] took the "high road" by defining an $\ominus$ operator, which given two events e_1 and e_2 computes the probability of $e_1 \wedge \neg e_2$ (just as $\otimes$ and $\oplus$ operator, $\ominus$ is dependent on the relationship between the events we wish to pursue). This operator is then used to define the difference operation in ProbView.

For temporal probabilistic databases we chose a somewhat simpler, and more intuitive approach. As set theory teaches us $A - B = A - (A \cap B)$. I.e., every element of set A either belongs to $A \cap B$ or to $A - B$. Let us now re-examine the intersection. Remember, when intersection of two relations r_1 and r_2 was defined, we wanted the it to contain *information common to both r_1 and r_2*. In the example above, both relations would have contained *information about the possibility of a rain on May 25.* And according to our definition of intersection, tuples representing these statements would have made it into the intersection. But then, as $r_1 - r_2 = r_1 - (r_1 \cap r_2)$, *the information about the possibility of a rain on May 25* **should not** appear in the difference between the two relations at all. I.e., the difference of r_1 and r_2 *should contain information that r_1 has and r_2 does not.*

We formalize this intuition as follows.

Definition 5.41 (difference of two annotated relations (TATA)) The *difference of annotated relations ANN(r) and ANN(r′)*, denoted ANN(r) − ANN(r'), is defined as ANN(r'') = $\{at \in$ ANN$(r) \mid (\forall at' \in$ ANN$(r'))(d \neq d' \vee t \neq t')\}$. □

Definition 5.42 (difference of two TP-relations (TPA)) The *difference of TP-relations r and* r', denoted $r - r'$, can be constructed in the following way: Initially, let $r'' = r$. Then for each $tp = (d, \gamma) \in r''$ and each $tp' = (d', \gamma') \in r'$ where $(d = d')$,

1. Let $\gamma'' = \emptyset$ and let $C' = (C'_1 \vee \ldots \vee C'_{n'})$. Recall that tp' contains exactly n' TP-cases.
2. For each $\gamma_i \in \gamma$ where $C''_i = (C_i \wedge \neg C')$ is consistent, add TP-case $\langle C''_i, D_i, L_i, U_i, \delta_i \rangle$ to γ''.
3. Remove tp from r''. Then if $\gamma'' \neq \emptyset$, add TP-tuple (d, γ'') to r''. □

5.3.5 Projection

A list $\mathcal{F}$ of fields is said to be *projectable* w.r.t. TP-relation r if (i) every field in $\mathcal{F}$ is a manifest data field of r, and (ii) $\mathcal{F}$ is non-empty. $\mathcal{F}$ is *projectable* w.r.t. annotated relation ANN(r) iff $\mathcal{F}$ is *projectable* w.r.t. r. It is important to note that hidden fields cannot be projected out.

Definition 5.43 (projection on an annotated relation (TATA)) Let $\mathcal{F}$ be a list of fields which are *projectable* w.r.t. ANN(r) and let "$A_1, \ldots, A_n$" be the (possibly empty) list of all manifest data fields which appear in the primary key of ANN(r) but do not appear in $\mathcal{F}$. Then the *projection of field list $\mathcal{F}$ on annotated relation ANN(r)*, denoted $\pi_{\mathcal{F}}$(ANN(r)), is defined as ANN(r'') = $\{(d'', t, L_t, U_t) \mid (\exists at \in$ ANN$(r)) \wedge (d'' = (\pi_{\mathcal{F}}(\mathcal{P}(d)), h'')) \wedge (h'' = (d.$H || "$A_1$:$d$.A$_1$, ..., A_n:d.A$_\text{n}$"$))\}$. □

Here, $\pi_{\mathcal{F}}(\mathcal{P}(d))$ works in the same way as projection in the classical relational algebra except it does not remove duplicates.

Definition 5.44 (projection on a TP-relation) Let $\mathcal{F}$ be a list of fields which are *projectable* w.r.t. r and let "$A_1, \ldots, A_n$" be the (possibly empty) list of all manifest data fields which appear in the primary key of r but do not appear in $\mathcal{F}$. Then the *projection of field list $\mathcal{F}$ on TP-relation* r, denoted $\pi_{\mathcal{F}}(r)$, can be constructed in the following way: Initially, let $r'' = \emptyset$. Then for each $(d, \gamma) \in r$, add TP-tuple (d'', γ) to r'' where $d'' = (\pi_{\mathcal{F}}(\mathcal{P}(d)), h'')$ and $h'' = (d.$H || "A_1:d.A$_1$, ..., A_n:d.A$_\text{n}$"). □

5.3.6 Cartesian Product and Join

Cartesian product is another operation which requires certain explanations. In a classical relational algebra, cartesian product of two relations merges the tuples. I.e., if $\langle A_1, \ldots, A_k \rangle$ is a relational scheme of relation R and $\langle B_1, \ldots, B_m \rangle$ is a relational scheme of relation R', then the relational scheme of $R \times R'$ will be $\langle A_1, \ldots, A_k, B_1, \ldots, B_k \rangle$.

However, TP-relations differ from classical relations as they have a special TP-case field. By definition of a TP-tuple, a tuple in a TP-relation can only have **one** TP-case field. So, no matter how we define our operation of cartesian product, it must construct one TP-case statement in the resulting tuple.

In order to come up with a reasonable definition of cartesian product we again seek help from our intuition about what the tp-tuples represent. Let us consider two tp-tuples $tp = (d, \gamma)$ and $tp' = d', \gamma'$. Our intuition about these tuples is that γ describes when the event represented by d may possibly occur, and similarly γ' describes when the event represented by d' may occur. As we want our operation of cartesian product to extend the classical relational algebra operation, it is reasonable to wish that the data part of the TP-cartesian product of tp and tp' be d, d'. But then tp-tuple $tp'' = tp \times tp'$[8] has to represent the temporal probabilistic information about the event represented by d, d'. This is event is really a *conjunction of the events represented by d and by d'*. Now, the only question that remains is how, given γ and γ' do we construct γ'' which would describe the probability of this conjunction.

Two issues are to be addressed here. First, as we mentioned multiple times in this text, there in not a single correct way to compute the probability of a conjunction of two events. Such computations in our frameworks are performed by probabilistic strategies(Section 5.1.3, Section 3.1). This suggests that in our framework a family of cartesian product operations would exist, each one performing probabilistic computations according to a particular conjunctive probabilistic strategy.

Finally, last but not least, we need to take a look at *when* the conjunction of two events can occur. Clearly, if γ contains some timepoint t in one of the temporal constraints, but no temporal constraint in γ' contains it we can be sure that according to tp and tp' events described by these tuples could not occur at time t. This suggests that *only the timepoints that are shared by the two TP-cases should appear in the TP-case of the cartesian product.*

We formalize our intuition in the following definitions.

Definition 5.45 (cartesian product of two annotated relations (TATA)) The *cartesian product of annotated relations ANN(r) and ANN(r') under the α probabilistic conjunction strategy*, denoted $\text{ANN}(r) \times_\alpha \text{ANN}(r')$, is defined as $\text{ANN}(r'') = \{(d'', t, L''_t, U''_t) \mid (\exists at \in \text{ANN}(r)) \wedge (\exists at' \in \text{ANN}(r')) \wedge$
$(d'' = (\mathcal{P}(d), \mathcal{P}(d'), h'')) \wedge (h'' = (d.\text{H} \parallel d'.\text{H})) \wedge (t = t') \wedge ([L''_t, U''_t] = [L_t, U_t] \otimes_\alpha [L'_t, U'_t])\}$. □

Definition 5.46 (cartesian product of two TP-relations (TPA)) The *cartesian product of TP-relations r and r' under the α probabilistic conjunction strategy*, denoted $r \times_\alpha r'$, can be constructed in the following way: Initially, let $r'' = \emptyset$. Then for each $tp = (d, \gamma) \in r$ and each $tp' = (d', \gamma') \in r'$,

1. Let $\gamma'' = \emptyset$.
2. For each time point t where $t \in \text{sol}(C_i)$ for some $\gamma_i \in \gamma$ and $t \in \text{sol}(C'_j)$ for some $\gamma'_j \in \gamma'$,
   (a) Let $[L_t, U_t] = [L_i \cdot x_t, U_i \cdot x_t]$ where $x_t = \delta_i(D_i, t)$.
   (b) Let $[L'_t, U'_t] = [L'_j \cdot x'_t, U'_j \cdot x'_t]$ where $x'_t = \delta'_j(D'_j, t)$.
   (c) Let $[L''_t, U''_t] = ([L_t, U_t] \otimes_\alpha [L'_t, U'_t])$.
   (d) Add TP-case $\langle (\#), (t), L''_t, U''_t, u \rangle$ to γ''.
3. If $\gamma'' \neq \emptyset$, add TP-tuple (d'', γ'') to r'' where $d'' = (\mathcal{P}(d), \mathcal{P}(d'), h'')$ and $h'' = (d.\text{H} \parallel d'.\text{H})$. □

[8] Cartesian product operation is usually defined on relations (sets). In this example we slightly abuse notation by applying it to tp-tuples. This allows us to simplify the notation and make the example easier to read.

Now we are ready to combine our definitions of selection, projection and cartesian product to produce the definition of TP-join. For simplicity we consider only "natural join" here.

Definition 5.47 (join of two annotated relations (TATA)) Let selection condition $\mathcal{C}$ be defined as $((\mathrm{ANN}(r).\mathcal{L}_1 = \mathrm{ANN}(r').\mathcal{L}_1) \wedge \ldots \wedge (\mathrm{ANN}(r).\mathcal{L}_n = \mathrm{ANN}(r').\mathcal{L}_n))$ where "$\mathcal{L}_1 \ldots \mathcal{L}_n$" is the list of all manifest data fields which occur in the schema for both $\mathrm{ANN}(r)$ and $\mathrm{ANN}(r')$. Then the *join of annotated relations ANN(r) and ANN(r′) under the α probabilistic conjunction strategy*, denoted $\mathrm{ANN}(r) \bowtie_\alpha \mathrm{ANN}(r')$, is defined as $\pi_{\mathcal{F}}(\sigma_{\mathcal{C}}(\mathrm{ANN}(r) \times_\alpha \mathrm{ANN}(r')))$ where $\mathcal{F}$ is the list of all manifest data fields which occur in the schema for either $\mathrm{ANN}(r)$ or $\mathrm{ANN}(r')$ after removing duplicate field names. $\Box$

Definition 5.48 (join of two TP-relations (TPA)) Let selection condition $\mathcal{C}$ be $((r.\mathcal{L}_1 = r'.\mathcal{L}_1) \wedge \ldots \wedge (r.\mathcal{L}_n = r'.\mathcal{L}_n))$ where "$\mathcal{L}_1 \ldots \mathcal{L}_n$" is the list of all manifest data fields which occur in the schema for both r and r'. Then the *join of TP-relations r and r' under the α probabilistic conjunction strategy*, denoted $r \bowtie_\alpha r'$, is defined as $\pi_{\mathcal{F}}(\sigma_{\mathcal{C}}(r \times_\alpha r'))$ where $\mathcal{F}$ is the list of all manifest data fields which occur in the schema for either r or r' after removing duplicate field names. $\Box$

5.3.7 Correctness of Relational Algebra

In this section we combine together the results that state that our definitions of relational operations in TPA *correctly implement* the operations of TATA. Recall from Def. 5.19 that a unary operation f^T of TPA *correctly implements* unary operation f^A of TATA iff for all TP-relations r $ANN(f^T(r)) = f^A(ANN(r))$, and a binary operation g^T of TPA *correctly implements* binary operation g^A of TATA iff for all TP-relations r, r' $ANN(g^T(r,r')) = g^A(ANN(r), ANN(r'))$.

We combine the results on correctness of implementation into one theorem.

Theorem 21 (Correctness of TPA w.r.t TATA) *Let r and r' be two TP-relations. All of the following is true:*

- *Compaction.*

  Let χ be any combination function. Then $\kappa_\chi(ANN(r)) = ANN(\kappa_\chi(r))$.

- *Intersection.*

  For any combination function χ, $\mathrm{ANN}(r \cap_\chi r') = \mathrm{ANN}(r) \cap_\chi \mathrm{ANN}(r')$.

- *Union.*

  For any combination function χ, $\mathrm{ANN}(r \cup_\chi r') = \mathrm{ANN}(r) \cup_\chi \mathrm{ANN}(r')$.

- *Selection.*

  For any select condition C, $\mathrm{ANN}(\sigma_{\mathcal{C}}(r)) = \sigma_{\mathcal{C}}(\mathrm{ANN}(r))$.

- *Difference.*

  $\mathrm{ANN}(r - r') = \mathrm{ANN}(r) - \mathrm{ANN}(r')$.

- *Cartesian Product.*

  For any conjunctive probabilistic strategy α, $\mathrm{ANN}(r \times_\alpha r') = \mathrm{ANN}(r) \times_\alpha \mathrm{ANN}(r')$.

- *Projection.*

  Let $\mathcal{F}$ be a projectable list of attributes. Then, $\mathrm{ANN}(\pi_{\mathcal{F}}(r)) = \pi_{\mathcal{F}}(\mathrm{ANN}(r))$.

- *Join.*

  For any conjunctive probabilistic strategy α, $r \bowtie_\alpha r' = \mathrm{ANN}(r) \bowtie_\alpha \mathrm{ANN}(r')$. □

5.4 Query Optimization

In this section we will describe the work which is currently under way. Query optimization, is no doubt a necessary part of any viable database system. While our approach to query optimization follows that of classical theory of relational databases, due to the structure of TP-relations, some of the results obtained for temporal probabilistic query optimization differ.

The theory of temporal probabilistic query optimization consists of two parts: rewrite rules and cost model. So far we have successfully established a number of rewrite rules for the TP-databases. Our plan for the research includes work on determining cost model or cost models appropriate for our framework.

Together with establishing some of the rewrite rules for TP Databases we were able to obtain a few negative results as well, which will be presented below. These results point out at the cases when *no general rewrite rule is possible.*

Theorem 22 (Selection-Projection) *Let r be a TP-relation, $\mathcal{F}$ be a list of data fields of r and C be a select condition which* ***does not*** *involve any fields from $\mathcal{F}$. Then*

$$\sigma_C(\pi_{\mathcal{F}}(r)) \equiv \pi_{\mathcal{F}}(\sigma_C(r))$$

Theorem 23 (Selection-Cartesian Product) *Let r and r' be two tp-relations and α be a conjunctive probabilistic strategy.*

1. *Let C_1 be a* **data** *selection condition which involves only fields from the relational schema of r. Then*
$$\sigma_{C_1}(r \times_\alpha r') \equiv \sigma_{C_1}(r) \times_\alpha r'$$
2. *Let C_2 be a* **data** *selection condition which involves only fields from the relational schema of r'. Then*
$$\sigma_{C_2}(r \times_\alpha r') \equiv r \times_\alpha \sigma_{C_2}(r')$$
3. *Let C be a* **temporal** *selection condition. Then*
$$\sigma_C(r \times_\alpha r') \equiv \sigma_C(r) \times_\alpha \sigma_C(r') = r \times_\alpha \sigma_C(r') = \sigma_C(r) \times_\alpha r'$$

Theorem 24 (No rewrite rules for probabilistic selection) *The following statements hold:*

- *Let C be a* ***probabilistic*** *condition. There exist such tp-relations r'', r, r', $r'' = r \times_\alpha r'$ that* ***no probabilistic conditions*** *C_1 on r and C_2 on r' will make the following equality* ***true:***
$$\sigma_C(r'') = \sigma_{C_1}(r) \times_\alpha \sigma_{C_2}(r).$$

- *Let $r'' = r \times_\alpha r'$. There exists such a* ***probabilistic*** *condition C on r'' that* ***no two probabilistic conditions*** *C_1 and C_2 (on r and r' respectively) make the equality*

$$\sigma_C(r'') = \sigma_{C_1}(r) \times_\alpha \sigma_{C_2}(r')$$

true.

Theorem 25 (Selection-Compaction) *Let C be a data or temporal select condition, and χ be a combination function. Then for any tp-relation r*

$$\sigma_C(\kappa_\chi(r)) \equiv \kappa_\chi(\sigma_C(r'))$$

Theorem 26 (Selection – Multiset Intersection) *Let r and r' be tp-relations, and C be a data or temporal select condition. Then*

$$\sigma_C(r \cap r') \equiv \sigma_C(r) \cap \sigma_C(r') \equiv \sigma_C(r) \cap r' \equiv r \cap \sigma_C(r').$$

Corollary 2 (Selection – Intersection) *Let χ be a combination function, C be a* ***data*** *or* ***temporal*** *select condition and r and r' be two tp-relations. Then*

$$\sigma_C(r \cap_\chi r') \equiv \sigma_C(r) \cap_\chi \sigma_C(r) \equiv r \cap_\chi \sigma_C(r') = \sigma_C(r) \cap_\chi r'.$$

Theorem 27 (Selection – Multiset Union) *Let C be an* ***arbitrary*** *select condition, r and r' be two tp-relations. Then*

$$\sigma_C(r \cup r') \equiv \sigma_C(r) \cup \sigma_C(r')$$

Proof(*sketch*).

Corollary 3 (Selection – Union) *Let χ be a combination function, C be a* ***data*** *or* ***temporal*** *select condition and r and r' be two tp-relations. Then*

$$\sigma_C(r \cup_\chi r') \equiv \sigma_C(r) \cup_\chi \sigma_C(r)$$

Theorem 28 (Selection – Difference) *Let C be a data or temporal select condition, and let r and r' be two tp-relations. Then*

$$\sigma_C(r - r') \equiv \sigma_C(r) - \sigma_C(r') \equiv \sigma_C(r) - r'.$$

Theorem 29 *Let C be a probabilistic select condition, and let r and r' be two tp-relations. Then*

$$\sigma_C(r - r') \equiv \sigma_C(r) - r'.$$

Notice that in many equivalence results above, probabilistic selects behave differently from data and temporal selects (the latter two actually behave in the same way everywhere). This difference even more drastic in the view of Theorem 24 which states than no rewrite rule is possible for the probabilistic selection – cartesian product combination.

For the latter combination the reason for this difference in behavior is clear: while the relational part of the tp-tuple in a cartesian product preserves all the data from both original relations, and while the two tuples from original relations must have coinciding timepoints in order for their composition to appear in a cartesian product, the probabilistic information gets changed by applying the appropriate conjunctive p-strategy. A data selection condition on the result of the cartesian product which satisfies the premises of the theorem above is therefore a data selection condition on one of the original relations. A temporal selection condition on the result of the cartesian product is a selection condition on *both* original relations. However a *probabilistic* selection condition on the result of the cartesian product is not automatically a selection condition on either of the original relation.

Similar reasons make probabilistic selections distinct in some other cases as well.

6 Temporal Probabilistic Logic Programs

The material of this section is from [10].

Creating a Logic Programming framework for reasoning about temporal probabilities can be done in a number of different possible ways. We decided to start with a simpler model, and upon developing it, to proceed towards more sophisticated temporal probabilistic logic programming theory or theories. As always when looking to create a logic programming framework our set of goals is

- Syntax;
- Model and Fixpoint Theories;
- Proof Procedures, including at least one "efficient" procedure;
- Algorithms and their Complexity;

In the rest of this section we describe how these four goals have been achieved in our first take at a Temporal Probabilistic Logic Programming framework. First we informally describe a few features of the already developed theory, and then suggest what other features might be implemented in the future extensions.

- **Use of possible worlds semantics for representing time.** The model is quite simple. A world (*TP-thread* in our terminology) is just one possible "spread" of events along the time allotted by our calendar. Collection of such worlds together with probabilistic distributions on them form our interpretations.
- **Compatibility with TP Databases.** While there are features of our TPLP framework which extend those of TP Databases, we made every attempt to have both models compatible with each other. This affected both the syntax of our temporal probabilistic logic programs (the definition of an annotation in TPLP framework mirrors the definition of a TP-case from TP Databases) and the semantics (a fixpoint of a temporal probabilistic logic program is defined to be a collection of annotated tuples from TP Databases [9]).

[9]Although there is a certain inconsistency in terminology here as these tuples are called *TP-tuples* in the logic programming theory. We are currently working on resolving such terminological inconsistencies.

- **Non-instantaneous events are allowed.** In this work we decided to take our TP Database model one step further and allow for events represented by our formulas to be non-instantaneous. This lead to changes in syntax (a notion of *probabilistic distribution function* had been replaced by a less restrictive *probabilistic weight function*).

- **Variable annotations are allowed.** This makes the theory much more versatile and flexible, at the cost of operating with somewhat inconvenient constructs in which the information about the probabilistic weights is stored.

- **Only atoms as heads of the rules.** This choice had been made to make the initial framework simpler. After HPLPs ([9]) we know how to extend a probabilistic logic programming framework to handle complex formulas in the heads of the rules and are planning to do so for the next extension of our current TPLP model.

- **"Lite" calendars.** As all timepoints in a calendar are linearly ordered, one can always consider without loss of generality a calendar to be a (*finite*) initial segment of natural numbers. This made the description of the theory somewhat more compact.

- **No probabilistic strategies.** This has been induced by our choice of model theory. Our interpretations, informally described above, encode the relationships between the events by themselves. When we are looking at *all interpretations that satisfy a conjunction or disjunction* we will be then looking at all possible relationships among the events involved. In fact it would be established below that our framework correctly describes the probabilities of conjunctions and disjunctions of temporal events *under the assumption of ignorance.* This makes it a direct descendant of the PLP work of Ng and Subrahmanian ([42, 43, 44]), rather than of our HPLP framework ([9]).

At this stage we are at the end of defining the first TPLP framework. Our research in the near future will concentrate on extending (and maybe modifying) it to create an analog of hybrid probabilistic programs for probabilistic temporal reasoning in the spirit of [9]. The latter framework would allow us to reason about conjunctions and disjunctions of the events under different assumptions about their relationship and complex heads of rules, while trying to preserve all the other features from the list above. Our experience with HPLPs suggests that a major overhaul of model theory would be needed to accommodate for the proposed changes.

In the rest of this section we will formally describe the first TPLP framework we have been constructing. As this is still considered work in progress (although at later stages of it), the description will not be quite complete. For example, no efficient proof procedure in the spirit of HR_P refutations (see Section 3.3, Def. 3.17) is given as of yet, although the second proof procedure described allows for creation of such a procedure on its basis. We do not know if there are any reasonably simple proof procedures which would be better than our second proof procedure.

6.1 Temporal Probabilistic Logic Programs: Syntax

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.The interpretation of these function symbols is given below. If p is an n-ary predicate symbol and $a_1, \ldots, a_n$ are either constants of variables, then $p(a_1, \ldots, a_n)$ is called a *simple event atom.* When all the a_i's are constants, $p(a_1, \ldots, a_n)$ is said to be *ground.* Each ground event atom represents one unique instance of an instantaneous event of the "real world". We use B_L to denote the set of ground simple event atoms.

Definition 6.1 *Let $A_1, \ldots A_k$ be all simple event atoms. Then $A_1 \wedge \ldots \wedge A_k$ and $A_1 \vee \ldots \vee A_k$ are called* compound event atoms. Simple event atoms *and compound event atoms are both* event atoms.

Definition 6.2 A *probabilistic annotation function* f_p of arity n is a total function $f_p : [0,1]^n \longrightarrow [0,1]$.

A *temporal annotation function* f_t of arity n is a total function $f_t : \tau^n \longrightarrow \tau$

We assume that associated with each *annotation function* is a body of software code computing that function, that is guaranteed to terminate on all inputs.

We also assume that all variable symbols from L are partitioned into three classes. We will call one class *object variables*, and this class will contain the regular first order logic variable symbols. The second and third classes of variable symbols, *probabilistic annotation variables* and *temporal annotation variables* will contain variable symbols that can over the interval $[0,1]$ and over calendar τ respectively. A ***calendar*** τ is defined as a set $\{1, \ldots t_{\max}\}$, i.e. calendars are initial segments of the set of natural numbers. Not that ***all our calendars are finite***.

Annotation variables can appear only inside *annotation items*, which are defined below:

Definition 6.3 A *probabilistic annotation item* δ is one of the following:

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

A *temporal annotation item* δ is one of the following:

- *a timepoint in the calendar τ.*
- *a temporal annotation variable symbol from L*
- *let f be a temporal annotation function symbol from L of arity n and let $\lambda_1, \ldots, \lambda_n$ be temporal annotation items. Then $f(\lambda_1, \ldots, \lambda_n)$ is also a temporal annotation item.*

Definition 6.4 *A temporal constraint $c = c(t, t_1, \ldots, t_n)$ is one of the following:*

- *Let λ be a temporal annotation items with $t_1, \ldots t_n$ being its only variable symbols. Then t op λ where op $\in \{=, <, >, \leq, \geq, \neq\}$ is a temporal constraint.*
- *Let λ_1 and λ_2 be temporal annotation items that contain only variables $t_1, \ldots t_n$. Then $t : \lambda_1 \sim \lambda_2$ is a temporal constraint.*

  In the constraints above variable t is called dependent variable *and all other variables are called* independent.

- *Let c_1 and c_2 be temporal constraints with the same dependent variable. Then $c_1 \wedge c_2$, $c_1 \vee c_2$ and $\neg c_1$ are temporal constraints.*

A temporal constraint c is called ground *if it contains no independent variables.*

A solution set for a temporal ground temporal constraint $c = c(t)$, denoted $sol(c)$ is determined by Def. 5.3.

A solution set for a temporal constraint $c = c(t, t_1, \ldots, t_k)$ is the set $\{sol(c(t, \bar{a})|\bar{a} = a_1, \ldots a_k$ is a ground substitution of independent variables $t_1, \ldots t_n\}$ (i.e., all possible solutions of c).

Definition 6.5 *A* **probabilistic weight function** *is any function $w : T \longrightarrow [0, 1]$, where $T \subseteq \tau$.*

If $c = c(t, t_1, \ldots, t_k)$ is a temporal constraint, a **generalized probabilistic weight function** *ω_c is a function that takes as arguments (i) a substitution $\bar{a} = (a_1, \ldots, a_k)$ of values for $t_1, \ldots t_k$ and (ii) a timepoint $\alpha \in \tau$ and returns a number between 0 and 1. We only require that if $\omega_c(\bar{a}, \alpha) \neq 0$, then $\alpha \in sol(c(t, a_1, \ldots, a_k))$.*

We notice that if temporal constraint c is *ground* then any generalized probabilistic weight function ω_c is reduced to a simple probabilistic weight function defined on $sol(c)$. In this case, as no independent variables are present in c, we will write $\omega_c(t)$.

Definition 6.6 *A* temporal probabilistic annotation *is a quadruple $\langle c, l, u, \omega_c \rangle$ where c is a temporal constraint, l and u are probabilistic annotation items and ω is a generalized probabilistic weight function.*

Definition 6.7 *Let $F = A_1 * \ldots * A_k$ be an event atom ($* \in \{\vee, \wedge\}$) and $\mu = \langle c, l, u, \omega_c \rangle$ be a tp-annotation. Then $F : \mu$ is a* tp-annotated basic formula *or just an* annotated basic formula*.*

Intuitively when F is ground, $F : \langle c, l, u, \omega_c \rangle$ represents the fact that the events described by F happened at some timepoint in $sol(c)$ with a probability in the interval $[l, u]$, and that the probability that these events occurred at a particular timepoint $t \in sol(c)$ is given by the weight function ω_c.

Definition 6.8 *Let $A : \mu, F_1 : \mu_1, \ldots F_m : \mu_m$ be tp-annotated basic formulas, $A \in B_L$. Then $A : \mu \longleftarrow F_1 : \mu_1 \wedge \ldots \wedge F_m : \mu_m$ is called a* temporal probabilistic clause *or a* tp-clause*.*

Definition 6.9 *A* temporal probabilistic program *(tp-program) is a finite set of tp-clauses. If P is a tp-program, we let $ground(P)$ denote the set of all ground instances of rules of P.*

Example 6.1 Let $\tau = [0, \ldots 30]$. Consider the following tp-program:

arrived($Item$,$Place$): $[x : 3 \sim 5, 0.5, 0.8, \{0.5, 0.3, 0.2\}] \longleftarrow$
    sent($Item$,$Place$): $[x = 1, 0.9, 1, \sharp]$.
arrived($Item$,$Place$): $[x : 6 \sim 8, 0.2, 0.4, \{0.75, 0, 0.25\}] \longleftarrow$
    sent($Item$,$Place$): $[x = 1, 0.9, 1, \sharp]$.
arrived($Item$,paris): $[x : 3 \sim 4, 0.5, 0.9, \{0.6, 0.4\}] \longleftarrow$
    sent($Item$,paris): $[x = 1, 0.95, 1, \sharp] \wedge$
    express_mail($Item$): $[x = 1, 1, 1, \sharp]$.
sent(shoes,rome): $[x = 1, 1, 1, \sharp] \longleftarrow$.
sent(letter,paris): $[x = 1 \vee x = 2, 1, 1, \{0.95, 0.1\}] \longleftarrow$.
express_mail(letter,paris): $[x = 1, 1, 1, \sharp] \longleftarrow$.

This program represents a small part of a deductive database of some company that deals with projected arrivals of the packages shipped by the company. The first two rules of the program provide the information on the probability distribution of the arrival time of an arbitrary package sent to any place. The third rule gives some extra information about the arrival time of the packages sent to Paris via express-mail. Three facts about shipments complete this simple program.

6.2 Model Theory and Fixpoint Semantics

6.2.1 Model Theory

In this section we introduce the model theory for tp-programs.

Definition 6.10 (Thread) *A* thread *is a function* $th : B_L \rightarrow 2^{\tau}$.

Intuitively, a thread th contains information about the times when *each* event occurs. It specifies one possible way events could occur in time. If for some ground event atom A, $th(A) = \emptyset$, we interpret it as meaning that event associated with A does not occur in thread th at all. The notion of TP-world below says that we may not know which of several possible threads actually describes the occurrence of events over time.

Definition 6.11 *A TP-world M is a pair $M = \langle TH, p\rangle$ where $TH = \{th\}$ is a set of threads and $p : TH \rightarrow [0,1]$ is a probability distribution function such that $\sum_{th \in TH} p(th) \leq 1$.*

Here, TH represents a set of possible ways events could occur, while p specifies the probability of each thread. Given a TP-world M, we may specify the probability of a formula via the following definition.

Definition 6.12 *Suppose $M = \langle TH, p\rangle$ is a TP-world and $A \in B_L$. The probability $p_M(A,t)$ that the event denoted by A occurred at time t according to the TP-model M is defined as follows:*

$$p_M(A,t) = \sum_{th \in TH, th(A) \ni t} p(th).$$

If $F = A_1 \wedge \ldots \wedge A_k$ and $G = A_1 \vee \ldots \vee A_k$ are ground compound event atoms*, then $p_M(F,t)$ and $p_M(G,t)$ are definable as:*

$$p_M(F,t) = \sum_{th \in TH, th(A_1) \ni t, \ldots, th(A_k) \ni t} p(th).$$

$$p_M(G,t) = \sum_{th \in TH, (\exists i \in \{1, \ldots k\})(th(A_i) \ni t)} p(th).$$

The above definition specifies the probability of a formula at a given point in time. We can extend this to define the probability that a formula is true at some (possibly more than one) time point in a solution of a ground temporal constraint c.

Definition 6.13 *If $M = \langle TH, p\rangle$ is a TP-world, $F = A_1 \wedge \ldots \wedge A_k$ and $G = A_1 \vee \ldots \vee A_k$ are event atoms, and $c = c(t)$ is a temporal constraint. Then $p_M(F, c)$ is defined as:*

$$p_M(F, c) = \sum_{th \in TH, th(A_1) \cap \ldots \cap th(A_k) \cap sol(c) \neq \emptyset} p(th).$$

$$p_M(G, c) = \sum_{th \in TH, (\exists i \in \{1, \ldots k\})(th(A_i) \cap sol(c) \neq \emptyset)} p(th).$$

We are now in position to define satisfaction of *ground formulas*.

Definition 6.14 (Satisfaction) *Let $M = \langle TH, p\rangle$ be a TP-world.*

- $M \models F : [c, l, u, \omega]$ **iff**
  - $p_M(F, c) \in [l, u]$
  - $(\forall t \in sol(c)) p_M(F, t) \in [\omega_c(\bar{a}, t) \cdot l, \omega_c(\bar{a}, t) \cdot u]$.
- $M \models F_1 : \mu_1 \wedge \ldots \wedge F_k$ **iff** $\forall i \in \{1, \ldots k\} M \models F_i : \mu_i$.
- $M \models F : \mu \longleftarrow F_1 : \mu_1 \wedge \ldots \wedge F_k$ **iff** *either $M \models F : \mu$ or $M \not\models F_1 : \mu_1 \wedge \ldots \wedge F_k$.*

A TP-world M is called a model *of a tp-program P ($M \models P$)* **iff** $(\forall r \in P)(M \models r)$*. A tp-program P is* consistent *if it has a model.*

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

6.2.2 Fixpoint Semantics

We now provide a fixpoint procedure to compute the *semantics* of tp-programs. The fixpoint procedure maps certain structures called tp-interpretations to tp-interpretations — in order to define tp-interpretations, we need to define some intermediate structures called tp-piles and tp-sets below.

Definition 6.15 *Let F be an ground event atom, t be a timepoint and $[l, u] \subseteq [0, 1]$. Then the quadruple (F, t, l, u) is called a* tp-tuple*.*

A collection of tp-tuples is called a TP-pile*. If R is a TP-pile we use $R[F, t]$ to denote the set $\{(F, t, \_, \_) \in R\}$. If for each pair F, t the size of $R[F, t]$ is at most 1, then we call R a* TP-set*.*

Intuitively, each tp-tuple contains information about the probability of an event associated with F (which can be compound) at timepoint t. A TP-pile is an arbitrary collection of such information. In a TP-pile there may be two or more tp-tuples that have information about the probability of some event F at some time t. Below we will show how such TP-piles can be simplified to TP-sets, a special kind of TP-piles where each pair F, t has at most one tp-tuple associated with it.

A TP-set R is complete iff $(\forall F)(\forall t)(R[F, t] \neq \emptyset)$. On non-complete TP-sets we define a *completion* operation *compl* as follows: $compl(R) = R \cup \{(F, t, 0, 1) | R[F, t] = \emptyset\}$. Clearly for each TP-set R there is a uniquely defined completion of it. Therefore without loss of generality from now on we will consider only *complete TP-sets*. On such sets we define the satisfaction as follows:

Definition 6.16 *Let R be a* TP-set *and $F : [c, l, u, \omega_c]$ be an event atom where $c = c(t, t_1, \ldots, t_k)$ is a temporal constraint.*

- *R satisfies $F : [c, l, u, \omega_c]$ ($R \models F : [c, l, u, \omega_c]$)* **iff** *for all substitutions $\bar{a} = (a_1, \ldots, a_n)$ such that $sol(c(t, \bar{a}) \neq \emptyset$ and for all $t \in sol(c(t, \bar{a})$ there exists such interval $[l_t, u_t]$ that*
  - $(F, t, l_t, u_t) \in R$
  - $[l_t, u_t] \subseteq [\omega_c(\bar{a}, t) \cdot l, \omega_c(\bar{a}, t) \cdot u]$
- $R \models F_1 : [c_1, l_1, u_1, \omega_{c_1}] \wedge \ldots \wedge F_n : [c_n, l_n, u_n, \omega_{c_n}]$ **iff** $(\forall i \in \{1, \ldots n\}) R \models F_i : [c_i, l_i, u_i, \omega_{c_i}]$
- $R \models F : [c, l, u, \omega] \longleftarrow F_1 : [c_1, l_1, u_1, \omega_1] \wedge \ldots \wedge F_n : [c_n, l_n, u_n, \omega_n]$ **iff** $R \models F : [c, l, u, \omega_c]$ *or* $F \not\models F_1 : [c_1, l_1, u_1, \omega_{c_1}] \wedge \ldots \wedge F_n : [c_n, l_n, u_n, \omega_{c_n}]$
- *R satisfies a tp-program P ($R \models P$)* **iff** *R satisfies every clause in P.*

A TP-interpretation is a TP-set that satisfies certain simple axioms. It will turn out that these axioms are exactly "right" from the point of view of making our fixpoint procedure compute the notion of logical consequence associated with tp-programs.

Definition 6.17 *A TP-set R is called a TP-interpretation* **iff** *the following conditions hold:*

- Conjunctive ignorance.
  $(\forall F = A_1 \wedge \ldots \wedge A_k, G = B_1 \wedge \ldots \wedge B_m)(\forall t)((F, t, l_1, u_1) \in R \wedge (G, t, l_2, u_2) \in R) \Rightarrow (R \models (F \wedge G) : [t, \max(0, l_1 + l_2 - 1), min(u_1, u_2), \sharp])$
- Disjunctive ignorance.
  $(\forall F = A_1 \vee \ldots \vee A_k, G = B_1 \vee \ldots \vee B_m)(\forall t)((F, t, l_1, u_1) \in R \wedge (G, t, l_2, u_2) \in R) \Rightarrow (R \models (F \vee G) : [t, max(l_1, l_2), min(1, u_1 + u_2), \sharp])$

The intervals $[\max(0, l_1 + l_2 - 1), min(u_1, u_2)]$ and $[max(l_1, l_2), min(1, u_1 + u_2)]$ represent the intervals in which the probability of respectively conjunction and disjunction of two events with probability intervals $[l_1, u_1]$ and $[l_2, u_2]$ will lie, if *nothing is known about the relationship between the events.*

In order to define a fixpoint operator, we first define an intermediate operator T_P^* which applied to a TP-interpretation produces a TP-pile. The operator T_P corrects the result of T_P^* to make it a $TP - interpretation$. In the definition of T_P^* we will employ the $\oplus$ notation defined below to "split" a compound event atom.

Definition 6.18 *Let $F = F_1 * \ldots * F_n$, $G = G_1 * \ldots * G_k$ ($k > 0$), $H = H_1 * \ldots * H_m$ ($m > 0$) where $* \in \{\wedge, \vee\}$. We write $G \oplus H = F$* **iff:**
(a) $\{G_1, \ldots, G_k\} \cup \{H_1, \ldots, H_m\} = \{F_1, \ldots, F_n\}$ and
(b) $\{G_1, \ldots, G_k\} \cap \{H_1, \ldots, H_m\} = \emptyset$.

This definition may now be used to define the operator T_P^* which takes TP-interpretations as input and produces TP-piles as output.

Definition 6.19 *Let P be a tp-program.*

- *Let $F = A$ be an atom of B_L.*

  $T_P^*(R)(A,t) = \{(A,t,l,u) | (\exists r \in ground(P))$
  $(r = A : [c, l', u', \omega_c] \longleftarrow Body; R \models Body;$
  $t \in sol(c)$ and $[l,u] = [\omega_c(t) \cdot l', \omega_c(t) \cdot u'])\}$

- *Let $F = A_1 \wedge \ldots \wedge A_k$.*

  $T_P^*(R)(F,t) = \{(F,t,l,u) | (\exists H, G)$
  $(F \equiv H \oplus G; (H,t,l_1,u_1), (G,t,l_2,u_2) \in R$
  and $[l,u] = [\max(0, l_1 + l_2 - 1), \min(u_1, u_2)])\}$

- *Let $F = A_1 \vee \ldots \vee A_k$.*

  $T_P^*(R)(F,t) = \{(F,t,l,u) | (\exists H, G)$
  $(F \equiv H \oplus G; (H,t,l_1,u_1), (G,t,l_2,u_2) \in R$
  and $[l,u] = [\max(l_1, l_2), \min(1, u_1 + u_2)])\}$

We can now extend the definition of T_P^* to map tp-interpretations to tp-interpretations.

Definition 6.20 *Let P be a tp-program. We define operator T_P as follows: $T_P(R)(F,t) = (F,t,l,u)$, where $[l,u] = \cap\{[l',u'] | (F,t,l',u') \in T_P^*(R)(F,t)$.*

In order to describe the fixpoint procedure based on the operators described above, we need to introduce an ordering on TP-interpretations, and prove that T_P operator is monotonic w.r.t. this order.

Definition 6.21 *Let R and S be two TP-sets. We say that $R \leq S$* **iff** *for every F and $t \in \tau$ if $(F,t,l,u) \in R$ and $(F,t,l',u') \in S$ then $[l,u] \supseteq [l',u']$.*

The following statement shows that the set $\mathcal{R}_{B_L}$ of all TP-interpretations for the formulas constructed out of atoms of B_L forms a complete lattice.

Lemma 3 *$\langle \mathcal{R}_{B_L}, \leq \rangle$ is a complete lattice.*

We can now show that our T_P operator is monotonic and proceed with the fixpoint procedure.

Theorem 30 *Let $R \leq S$ and let P be a TP-program. Then $T_P(R) \leq T_P(S)$.*

The following definition specifies how we might iteratively apply the T_P operator.

Definition 6.22 *Let P be a tp-program.*

- $T_P^0 = \bot = compl(\emptyset)$
- $T_P^{i+1} = T_P(T_P^i)$
- $T_P^\lambda = \cap_{i \leq \lambda}(T_P^i)$

The following theorem is important. It shows that T_P has a least fixpoint, $lfp(T_P)$, and that this least fixpoint precisely captures the model-theoretic notion of logical consequence associated with tp-programs. Thus, iterative application of the T_P operator yields all ground event atoms that are logical consequences of P.

Theorem 31 *1.* $R \models P$ **iff** $T_P(R) \leq R$

2. $lfp(T_P) \models P$

3. $(\forall F)(P \models F : [c, l, u, \omega_c]$ **iff** $(\forall \bar{a}(sol(c(t, \bar{a}) \neq \emptyset) \Rightarrow (\forall t \in sol(c))(\exists l_t, u_t)((F, t, l_t, u_t) \in lfp(T_P) \wedge [l_t, u_t] \subseteq [\omega_c(\bar{a}, t) \cdot l, \omega_c(\bar{a}, t) \cdot u]))).$

6.3 Fixpoint Computation and Entailment Problem (Ground Case)

In this section, we develop algorithms and associated complexity results for computation of the fixpoint of a ground tp-program, checking its consistency, and query answering. To simplify time bounds we assume that such elementary problems as checking "$t \in sol(c)$" and computing $\omega_c(\bar{a}, t)$ can be done in constant time [10] , and enumeration of $sol(c)$ in time linear of $|sol(c)|$.

For any TP-set (TP-interpretation) R let $atoms(R) = \{(A, t, l, u) \in R, A \textit{ is a simple event atom}\}$. Let P be a ground tp-program over the set of ground simple event atoms $\{A_1, ..., A_N\}$ and the calendar $\tau = [0, t_{\max}]$. Let us denote by R_P its least fixpoint $lfp(T_P)$. The following lemma shows that R_P can be easy defined by $atoms(R_P)$.

Lemma 4 *For all $t \in \tau$ and ground compound event atoms F*

- *if $F = A_1 \wedge \ldots \wedge A_k$ and for each $1 \leq i \leq k \quad (A_i, t, l_i, u_i) \in atoms(R_P)$ then $(F, t, l, u) \in R_P$, for $l = l_1 + \ldots + l_k + 1 - k$ and $u = \min\{u_1, \ldots, u_k\}$;*
- *if $F = A_1 \vee \ldots \vee A_k$. and for each $1 \leq i \leq k \quad (A_i, t, l_i, u_i) \in atoms(R_P)$ then $(F, t, l, u) \in R_P$, for $l = \max\{l_1, \ldots, l_k\}$ and $u = \min\{1, u_1 + \ldots + u_k\}$.*

This lemma allows us to propose an efficient algorithm to construct $atoms(R_P)$. This algorithm will also check the consistency of tp-programs. Let P be a tp-program over the set $B_L = \{A_1, ..., A_N\}$ and let $\{F_1, ..., F_s\}$ be the set of all compound event atoms included in clauses of P.

Algorithm LFP-atoms.
Input: a ground tp-program P.
Output: $atoms(R_P)$.

BEGIN (algorithm)
1. $OLD := \emptyset$; $NEW := \emptyset$;
2. FOR $i = 1$ TO N DO
FOR EACH $t \in \tau$ DO
$NEW := NEW \cup \{(A_i, t, 0, 1)\}$;

[10] We do this to make our complexity results independent of the issues such as the implementation of ω functions. We can then factor the complexity of computing $\omega(\bar{a}, t)$ back into our complexity estimate. It should be noted that for ground programs on which we are concentrating in this section the assumption that $\omega_c(t)$ can be computed in constant time is not all that unreasonable.

3. FOR $j = 1$ TO s DO
FOR EACH $t \in \tau$ DO
$NEW := NEW \cup \{(F_j, t, 0, 1)\}$;
4. WHILE $OLD \neq NEW$ DO
BEGIN
$OLD := NEW$;
5. FOR EACH $r = A : [c, l, u, \omega_c] \leftarrow Body \in P$ such that $OLD \models Body$ DO
6. FOR EACH $t \in sol(c)$ DO
BEGIN
LET $(A, t, l', u') \in NEW$;
$l_1 := \max(l', \omega_c(t) * l)$;
$u_1 := \min(u', \omega_c(t) * u)$;
IF $u_1 < l_1$ THEN RETURN $\emptyset$;
$NEW := (NEW \setminus \{(A, t, l, u)\}) \cup \{(A, t, l_1, u_1)\}$;
$P := P \setminus \{r\}$
END;
7. FOR $j = 1$ TO s DO
8. FOR EACH $t \in \tau$ DO
BEGIN
LET $F_j = B_1^j * \ldots * B_k^j$ AND FOR EACH $1 \leq p \leq k$ $(B_p^j, t, l_p^j, u_p^j) \in NEW$
IF $* = \wedge$ THEN BEGIN
$l := l_1^j + \ldots + l_k^j + 1 - k$; $u := \min\{u_1^j, \ldots, u_k^j\}$ END
ELSE BEGIN $\{* = \vee\}$
$l := \max\{l_1^j, \ldots, l_k^j\}$; $u := \min\{1, u_1^j + \ldots + u_k^j\}$ END;
$NEW := (NEW \setminus \{(F_j, t, \_, \_)\}) \cup \{(F_j, t, l, u)\}$
END;
END;
9. RETURN $atoms(NEW)$
END.

The following theorem shows that algorithm **LFP-atoms** is a correct way of checking P for inconsistency and (if P is consistent) of computing the least fixpoint of T_P for simple event atoms. It also establishes the polynomial time complexity of the algorithm. The proof of this result uses lemma 4.

Theorem 32 *Let P be any ground tp-program and let P contain m clauses. Let $|P|$ be the size of P (under some standard encoding). Then* **Algorithm LFP-atoms** *returns $\emptyset$ iff P is inconsistent. If P is consistent then it computes $atoms(R_P)$ in time $O(m|P|t_{max})$.*

Now we consider the entailment problem: given a tp-program P and a *ground query* G, check whether $P \models G$. Our queries will be conjunctions of annotated basic formulas of the form $F : [c, l, u, \omega_c]$ where F is ground. One way to answer such queries is to add F to the list of compound events $\{F_1, ..., F_s\}$ of P run start algorithm **LFP-atoms**, and after the algorithm finishes, check the values (F, t, l, u) for all $t \in sol(c)$. But it seems more reasonable to use algorithm **LFP-atoms** one time as preprocessor to obtain $atoms(R_P)$, and then to answer to queries using the following simple algorithm.

Algorithm Simple query.
Input: set $atoms(R_P)$ and a simple query $F : [c, l, u, \omega_c]$.
Output: "YES" if $P \models F : [c, l, u, \omega_c]$, otherwise "NO".

```
BEGIN (algorithm)
  LET F = B_1 * ... * B_k
  FOR EACH t ∈ sol(c) DO
  BEGIN    LET FOR EACH 1 ≤ p ≤ k   (B_p, t, l_p, u_p) ∈ atoms(R_P);
    IF * = ∧ THEN BEGIN
      L := l_1 + ... + l_k + 1 − k;  U := min{u_1, ..., u_k} END
    ELSE BEGIN    {* = ∨}
      L := max{l_1, ..., l_k}; U := min{1, u_1 + ... + u_k}  END;
    IF (L < l * ω_c(t)) OR (M > u * ω_c(c, t))
    THEN RETURN "NO"
  END
  RETURN "YES"
END.
```

The following theorem shows that after polynomial time preprocessing any entailment problem can be solved in linear time.

Theorem 33 *(1) Algorithm* **Simple query** *gives a correct answer to any simple query of the form* $F : [c, l, u, \omega_c]$ *in time* $O(|F| + |sol(c)|)$.

(2) There exists an algorithm which given the set $atoms(R_P)$ *answers to any query* G *of the form* $F_1 : [c_1, l_1, u_1, \omega_1] \wedge ... \wedge F_n[c_n, l_n, u_n, \omega_n]$ *in time linear of* $(|G| + \sum_{i=1}^{n} |sol(c_i)|\ )$.

6.4 Proof Procedure(s)

In this section, we present two basic proof procedures – the first is inefficient, but easy to understand, while the second is more efficient but also more complex.

6.5 Unification

If $*$ is either $\wedge$ or $\vee$, then we say that two event atoms $A_1 * \ldots * A_k$ and $B_1 * \ldots * B_m$ are unifiable iff there is a substitution θ such that $\{A_1\theta, \ldots, A_k\theta\} = \{B_1\theta, \ldots, B_m\theta\}$. This notion of unification was introduced in [42] and it was proved that though most general unifiers are not necessarily unique, a corresponding notion of a *maximally* general unifier (max-gu) exists. For space reasons, we do not go into details of this here.

6.5.1 Basic Proof Procedure

In this section, we describe how we can expand a tp-program P into a program called its *closure*. This closure may then be used to define a resolution based proof procedure.

Definition 6.23 (Explosions) *Let* $F : [c, l, u, \omega_c]$ *be a basic formula. The* explosion *of* $F : [c, l, u, \omega_c]$, *denoted* $\mathcal{E}(F : [c, l, u, \omega_c])$ *is defined as* $\{F : [t, \omega_c(c, t) \cdot l, \omega_c(c, t) \cdot u, \sharp] | t \in sol(c)\}$. *The* exploded basic formula *for* $F : [c, l, u, \omega_c]$, *denoted* $\mathcal{E}_{\mathcal{F}}(F : [c, l, u, \omega_c])$, *is* $\bigwedge_{t \in sol(c)} F : [t, \omega_c(c, t) \cdot l, \omega_c(c, t) \cdot u, \sharp]$.

Let $r = F : [c, l, u, \omega_c] \longleftarrow Body$ *be a tp-clause. Then the* explosion *of* r, *denoted* $\mathcal{E}(r)$ *is defined as* $\{F : [t, \omega_c(c, t) \cdot l, \omega_c(c, t) \cdot u, \sharp] \longleftarrow Body | t \in sol(c)\}$.

If P is a tp-program, then the **explosion** *of P, denoted $\mathcal{E}(P)$ is defined as $\{\mathcal{E}(r) | r \in P\}$*

The explosion operation explicitly enumerates the probabilities of an event at all time points associated with a constraint. (As this set of time points can potentially be very large, we use the term explosion to describe this operation). Using this, we can now define the **timepoint-based closure** of program P.

Definition 6.24 (Timepoint-based Closure) *Let P be a tp-program.*

$\bullet REDUN(P) = \mathcal{E}(P) \cup \{F : [t, 0, 1, \sharp] \longleftarrow | F \in B_L; t \in \tau\}$

- *We define the closure of P (denoted $CL(P)$) iteratively:*

1. $CL^0(P) = REDUN(P)$
2. *To construct $CL^{i+1}(P)$ given $CL^i(P)$ we have to apply the following rule to* all possible pairs of clauses *from $CL^i(P)$:*
   - **Clarification rule:**
     If clauses $F : [t, l, u, \sharp] \longleftarrow Body$ and $F' : [t, l', u', \sharp] \longleftarrow Body'$ are in $CL^i(P)$ and F and F' are unifiable via max-gu Θ, add the clause
     $)F : [t, \max(l, l'), \min(u, u'), \sharp] \longleftarrow Body \wedge Body')\Theta$ *to* $CL^{i+1}(P)$.
   - **$\wedge$-composition rule:**
     If clauses $F : [t, l, u, \sharp] \longleftarrow Body$ and $F' : [t, l', u', \sharp] \longleftarrow Body'$ are in $CL^i(P)$, $F = A_1 \wedge \ldots \wedge A_k$ and $G = B_1 \wedge \ldots \wedge B_m$ $(k, m \geq 1)$, add the clause
     $(F \wedge G) : [t, \max(0, l + l' - 1), \min(u, u'), \sharp] \longleftarrow Body \wedge Body'$ *to* $CL^{i+1}(P)$.
   - **$\vee$-composition rule:**
     If clauses $F : [t, l, u, \sharp] \longleftarrow Body$ and $F' : [t, l', u', \sharp] \longleftarrow Body'$ are in $CL^i(P)$, $F = A_1 \vee \ldots \vee A_k$ and $G = B_1 \vee \ldots \vee B_m$ $(k, m \geq 1)$, add the clause
     $(F \vee G) : [t, \max(l, l'), \min(u + u', 1), \sharp] \longleftarrow Body \wedge Body'$ *to* $CL^{i+1}(P)$.
3. $CL(P) = \cup_{i \geq 0} CL^i(P)$.

The following result says that CL(P) is sound w.r.t. P.

Lemma 5 *For every clause $r \in CL(P)$, $P \models r$.*

Notice that the syntax of CL(P) is somewhat different from the syntax of P, as we allow the rules in CL(P) to have non-atomic heads. However this extension is supported by our definitions of satisfaction on TP-models (def. 6.14) and TP-sets (def. 6.16). We are now ready to proceed with the resolution procedure.

Definition 6.25 *A* **tp-query** *is an expression of the form $\exists(H_1 : [c_1, l_1, u_1] \wedge \ldots \wedge H_n : [c_n, l_n, u_n])$. The* **explosion** *of a tp-query Q, denoted $\mathcal{E}_{\mathcal{F}}(Q)$ is an expression of the form:*

$$\bigwedge_{i=1}^{n} \mathcal{E}_{\mathcal{F}}(H_i : [c, l_i, u_i, \omega_c])$$

In other words, the explosion of a tp-query is a conjunction of the explosions of all basic formulas in the query.

Definition 6.26 *Let Q be a query and let $\mathcal{E}_{\mathcal{F}}(Q) = F_1 : [t_1, l_1, u_1, \sharp] \wedge \ldots F_n : [t_n, l_n, u_n, \sharp]$. Let $r \equiv G : [t, l, u, \sharp] \longleftarrow G_1 : \mu_1 \wedge \ldots G_k : \mu_k \in CL(P)$ and G be unifiable with F_i via max-gu Θ. Then*

$$\boxed{\begin{array}{l} \exists((F_1 : [t_1, l_1, u_1, \sharp] \wedge \ldots \wedge F_{i-1} : [t_{i-1}, l_{i-1}, u_{i-1}, \sharp] \wedge \\ \mathcal{E}_{\mathcal{F}}(G_1 : \mu_k) \wedge \ldots \wedge \mathcal{E}_{\mathcal{F}}(G_k : \mu_k) \wedge \\ F_{i+1} : [t_{i+1}, l_{i+1}, u_{i+1}, \sharp] \wedge \ldots \wedge F_n : [t_n, l_n, u_n, \sharp])\Theta) \end{array}}$$

is a tp-resolvent *of r and Q* **iff** $[l, u] \subseteq [l_i, u_i]$.

Definition 6.27 Let $Q \equiv \exists(F_1 : \mu_1 \wedge \ldots \wedge F_n : \mu_n)$ be an initial query, and P a tp-program. A **tp-deduction** of Q from P is a sequence $< Q_1, r_1, \Theta_1 > \ldots < Q_s, r_s, \Theta_s > \ldots$ where, $Q_1 = \mathcal{E}_{\mathcal{F}}(Q)$, for all $i \geq 1$, r_i is a renamed version of a clause in CL(P), and Q_{i+1} is a *tp-resolvent* of Q_i and r_i via max-gu Θ_i.

A **tp-refutation** of Q from P is a *finite tp-deduction* $< Q_1, r_1, \Theta_1 > \ldots < Q_s, r_s, \Theta_s >$ where, the tp-resolvent of Q_s and r_s via Θ_s is the empty query. $\Theta_1 \ldots \Theta_r$ is called the *computed answer substitution.*

The following two theorems state that our first proof procedure is sound and complete.

Theorem 34 *[Soundness of tp-refutation]*
Let P be a tp-program, and Q be an initial query. If there exists a tp-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 35 *[Completeness of tp-refutation]*
Let P be a consistent tp-program and Q be a query. Then, if $P \models Q$ then there exists a tp-refutation of Q' from P.

6.6 Temporal Constraint-based Proof Procedure

It is clear that the above proof procedure is highly inefficient because the *explosion* operation is applied to the entire program and entire query, regardless of any specific aspects of either that may allow for optimization. So, in our attempt to define a better proof procedure, mandatory "hard-wired" explosion has to replaced by a more sophisticated procedure that would allow us to keep as much information as possible in a "compressed" state, and would allow resolution of such "compressed" formulas. We start by introducing two extensions of the tp-program syntax. We will use expressions in one format of the extended syntax to store the new closure of a tp-program. The other extended syntax format will be used during the resolution procedure to keep track of the parts of the query which had already been "established" and the parts which need to be checked.

Definition 6.28 *Let c and d be temporal constraints such that $sol(d) \subseteq sol(c)$. Then we say that an expression of the form $F : [c, d, l, u, \omega_c]$ is in* extended query syntax.
For an annotated basic formula $F : [c, l, u, \omega_c]$ let $Ext_{\mathcal{Q}}(F : [c, l, u, \omega_c]) = F : [c, c, l, u, \omega_c]$.
We also define $Ext_{\mathcal{Q}}(F_1 : [c_1, l_1, u_1, \omega_{c_1}] \wedge \ldots \wedge F_n : [c_n, l_n, u_n, \omega_{c_n}]) = \bigwedge_{i=1}^{n} Ext_{\mathcal{Q}}(F_i : [c_i, l_i, u_i, \omega_{c_i}])$.

Let ω_c^l and ω_c^u be generalized weight functions. Then we say that an expression of the form $F : [c, l, u, \omega_c^l, \omega_l^u]$ is in extended program syntax*. For annotated basic formula $F : [c, l, u, \omega_c]$ let $Ext_{\mathcal{P}}(F : [c, l, u, \omega_c]) = F : [c, l, u, \omega_c, \omega_c]$. If $r = F : [c, l, u, \omega_c] \longleftarrow Body$ is a tp-clause, then we define $Ext_{\mathcal{P}}(r) = Ext_{\mathcal{P}}(F : [c, l, u, \omega_c]) \longleftarrow Body$.*

Expressions in *extended query syntax* have an extra temporal constraint in them. Throughout the query resolution process, the temporal constraint c of a query in extended syntax will not change and will be equal to the initial constraint. Constraint d in the meantime will contain information about the timepoints in $sol(c)$ for which we still need to establish probability ranges. Initially $d = c$, as when the query is asked we have no information about any timepoints. In our query resolution procedure we will make sure that whenever a formula is added to the query it is added in the extended query syntax format.

Expressions in the *extended program syntax* have two distribution functions. In an annotated formula $F : [c, l, u, \omega_c]$, ω_c describes how **both** lower and upper bounds of the probability interval $[l, u]$ are distributed among the elements of $sol(c)$. However, as the resolution procedure described below proceeds, we may wind up in a situation when for for a newly created temporal constraint the lower and the upper bounds of the probability intervals computed are distributed differently. The *extended program syntax* format is used for the purpose of keeping track of both distributions. As our resolution process will be described in terms of resolution against a closure of a program P, the clauses of the closure of P will be in the extended program syntax format. *Note that these extended syntaxes are used by a query processing procedure and they will not be visible to users, nor will they change either the semantics of tp-programs.* We can now define the **temporal constraint-based closure** of program P.

Definition 6.29 (Temp. Constraint-based Closure) *Let P be a tp-program.*

- $REDUN_{Ext}(P) = \{Ext_{\mathcal{P}}(r) | r \in P\} \cup$
  $\{F : [c^\star, 0, 1, \top, \top] \longleftarrow | F \in B_L;$
  $c^\star = \neg(\bigvee_{F:[c,l,u,\omega_c]\longleftarrow \in P} c)\}$
- *We define the* **temporal constraint-based closure** *of P (denoted $CCL(P)$) iteratively:*

  1. $CCL^0(P) = REDUN_{Ext}(P)$
  2. *To construct $CCL^{i+1}(P)$ given $CCL^i(P)$ we have to apply the following rule to* all possible pairs of clauses *from $CCL^i(P)$:*

     - **Clarification rule:**
       If clauses $F : [c, l, u, \omega^l_c, \omega^u_c] \longleftarrow Body$ and $F' : [c', l', u', \omega'^l_{c'}, \omega'^u_{c'}] \longleftarrow Body'$ are in $CCL^i(P)$, F and F' are unifiable via max-gu Θ, and $sol(c \wedge c') \neq \emptyset$ add to $CCL^{i+1}(P)$ the clause $(F : [c^\star, l^\star, u^\star, \omega^{\star l}_{c^\star}, \omega^{\star u}_{c^\star}] \longleftarrow Body \wedge Body')\Theta$, where $c^\star = c \wedge c'$,

$$\boxed{\begin{array}{l} l^\star = \sum_{t\in sol(c\wedge c')} \max(\omega^l_c(t) \cdot l, \omega'^l_{c'}(t) \cdot l') \\ u^\star = \sum_{t\in sol(c\wedge c')} \min(\omega^u_c(t) \cdot u, \omega'^u_{c'}(t) \cdot u') \\ \omega^{\star l}_{c^\star}(t) = \frac{\max(\omega^l_c(t)\cdot l, \omega'^l_{c'}(t)\cdot l')}{l^\star};\ \text{if } l^\star \neq 0 \text{ and} \\ \omega^{\star u}_{c^\star}(t) = 0 \text{ otherwise};\ t \in sol(c \wedge c') \\ \omega^{\star u}_{c^\star}(t) = \frac{\min(\omega^u_c(t)\cdot u, \omega'^u_{c'}(t)\cdot u')}{u^\star};\ \text{if } u^\star \neq 0 \text{ and} \\ \omega^{\star u}_{c^\star}(t) = 0 \text{ otherwise};\ t \in sol(c \wedge c') \end{array}}$$

     - **$\wedge$-composition rule:**
       If clauses $F : [c, l, u, \omega^l_c, \omega^u_c] \longleftarrow Body$ and $G : [c', l', u', \omega'^l_c, \omega'^u_{c'}] \longleftarrow Body'$ are in $CCL^i(P)$, $F = A_1 \wedge \ldots \wedge A_k$ and $G = B_1 \wedge \ldots \wedge B_m$ and $sol(c \wedge c') \neq \emptyset$, then add to $CCL^{i+1}(P)$ the clause
       $(F \wedge G) : [c^\star, l^\star, u^\star, \omega^\star_{c^\star}, \omega^{\star u}_{c^\star}] \longleftarrow Body \wedge Body'$, where $c^\star = c \wedge c'$

$$\boxed{\begin{array}{l} l^\star = \sum_{t \in sol(c \wedge c')} \max(\omega_c^l(t) \cdot l + \omega_{c'}^{\prime l}(t) \cdot l' - 1, 0) \\ u^\star = \sum_{t \in sol(c \wedge c')} \min(\omega_c^u(t) \cdot u, \omega_{c'}^{\prime u}(t) \cdot u')] \\ \omega_{c^\star}^{\star l}(t) = \frac{\max(\omega_c^l(t) \cdot l + \omega_{c'}^{\prime l}(t) \cdot l' - 1, 0)}{l^\star}; \text{ if } \ l^\star \neq 0 \\ \text{and } \ \omega_{c^\star}^{\star l}(t) = 0 \text{ otherwise; } \ t \in sol(c \wedge c') \\ \omega_{c^\star}^{\star u}(t) = \frac{\min(\omega_c^u(t) \cdot u, \omega_{c'}^{\prime u}(t) \cdot u')}{u^\star}; \text{ if } \ u^\star \neq 0 \\ \text{and } \ \omega_{c^\star}^{\star_u}(t)(t) = 0 \text{ otherwise; } \ t \in sol(c \wedge c') \end{array}}$$

- $\vee$-composition rule:

If clauses $F : [c, l, u, \omega_c^l, \omega_c^u] \longleftarrow Body$ *and* $G : [c', l', u', \omega_{c'}^{\prime l}, \omega_{c'}^{\prime u}] \longleftarrow Body'$ *are in* $CCL^i(P)$, $F = A_1 \vee \ldots \vee A_k$ *and* $G = B_1 \vee \ldots \vee B_m$ *and* $sol(c \wedge c') \neq \emptyset$, *then add to* $CCL^{i+1}(P)$ *the clause*
$(F \vee G) : [c^\star, l^\star, u^\star, \omega_l^\star, \omega_u^\star] \longleftarrow Body \wedge Body'$, *where* $c^\star = c \wedge c'$

$$\boxed{\begin{array}{l} l^\star = \sum_{t \in sol(c \wedge c')} \max(\omega_c^l(t) \cdot l, \omega_{c'}^{\prime l}(t) \cdot l') \\ u^\star = \sum_{t \in sol(c \wedge c')} \min(\omega_c^u(t) \cdot u + \omega_{c'}^{\prime u}(t) \cdot u', 1)] \\ \omega_{c^\star}^{\star l}(t) = \frac{\max(\omega_c^l(t) \cdot l, \omega_{c'}^{\prime l}(t) \cdot l')}{l^\star}; \text{if } \ l^\star \neq 0 \text{ and} \\ \omega_l^\star(t) = 0 \text{ otherwise; } \ t \in sol(c \wedge c') \\ \omega_{c^\star}^{\star u}(t) = \frac{\min(\omega_c^u(t) \cdot u + \omega_{c'}^{\prime u}(t) \cdot u', 1)}{u^\star}; \text{ if } \ u^\star \neq 0 \text{ and} \\ \omega_{c^\star}^{\star u}(t) = 0 \text{ otherwise; } \ t \in sol(c \wedge c') \end{array}}$$

3. $CCL(P) = \cup_{i \geq 0} CCL^i(P)$.

The following result says that CCL(P) is sound w.r.t. P.

Lemma 6 *For every clause* $r \in CCL(P)$, $P \models r$.

Our resolution procedure will convert the initial query into the extended query syntax. "Chunks" of each conjunct of the query can be resolved against the appropriate rules of the CCL(P). If a query conjunct is of a form $F : [c, d, l, u, \omega_c]$, CCL($P$) contains a clause r with the head $F' : [c', l', u', \omega_l', \omega_u']$, F and F' are unifiable and $d \wedge c'$ has a non-empty set of solutions, then if we can prove the body of r then we can also extract some information about probability of F over the $sol(d \wedge c')$. To prove the initial query we will still need to prove $F : [c, d \wedge \neg c', l, u, \omega_c]$. This intuition is formalized in the definitions that follow.

Definition 6.30 *Let* Q *be a query in extended query format and let* $Q = F_1 : [c_1, d_1, l_1, u_1, \omega_c] \wedge \ldots F_n : [c_n, d_n, l_n, u_n, \omega_n]$. *Let* $r \equiv G : [c, l, u, \omega_l, \omega_u] \longleftarrow Body \in CCL(P)$ *and* G *be unifiable with* F_i *via max-gu* Θ. *Then*

$$\boxed{\begin{array}{l} \exists((F_1 : [c_1, d_1, l_1, u_1, \omega_1] \wedge \ldots \wedge F_i : [c_i, d_i \wedge \neg c, l_i, u_i, \omega_i] \\ \wedge Ext_{\mathcal{Q}}(Body) \wedge F_{i+1} : [c_{i+1}, d_{i+1}, l_{i+1}, u_{i+1}, \omega_{i+1}] \\ \wedge \ldots \wedge F_n : [c_n, d_n, l_n, u_n, \omega_n])\Theta) \end{array}}$$

is a ctp-resolvent *of* r *and* Q **iff**
$(1) sol(d_i \wedge c) \neq \emptyset$
$(2)(\forall t \in sol(d_i \wedge c))([\omega_i(c_i, t) \cdot l, \omega_i(c_i, t) \cdot u] \supseteq [\omega_c^l(c, t) \cdot l, \omega_c^u(c, t) \cdot u])$

Note that if $sol(d_i) = sol(c)$ *we will have* $sol(d_i \wedge \neg c) = \emptyset$. *In this case,* $F_i : [c_i, d_i \wedge \neg c, l_i, u_i, \omega_i]$ *should not be added to the resolvent, as its proof is finished.*

Definition 6.31 Let $Q \equiv \exists(F_1 : \mu_1 \wedge \ldots \wedge F_n : \mu_n)$ be an initial query, and P a tp-program. A **tp-deduction** of Q from P is a sequence $< Q_1, r_1, \Theta_1 > \ldots < Q_s, r_s, \Theta_s > \ldots$ where, $Q_1 = \exists(Ext_{\mathcal{Q}}(Q)$, for all $i \geq 1$, r_i is a renamed version of a clause in CCL(P) and Q_{i+1} is a ctp-resolvent of Q_i and r_i via max-gu Θ_i.

n ctp-refutation of Q from P is a *finite ctp-deduction* $< Q_1, r_1, \Theta_1 > \ldots < Q_s, r_s, \Theta_s >$ where, the ctp-resolvent of Q_s and r_s via Θ_s is the empty query. $\Theta_1 \ldots \Theta_r$ is called the *computed answer substitution.*

Our second proof procedure is also sound and complete.

Theorem 36 *[Soundness of ctp-refutation]*
Let P be a tp-program, and Q be an initial query. If there exists a ctp-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 37 *[Completeness of ctp-refutation]*
Let P be a consistent tp-program and Q be a query. Then, if $P \models Q$ then there exists a tp-refutation of Q' from P.

The second procedure is a major improvement on the first. First, while *tp-resolution* mandates the explosion of the entire program and all queries down to the timepoint level, our *ctp-resolution* avoids this This makes the closure set significantly smaller. Second, during resolution, we may employ knowledge about specific distribution functions and probability intervals to avoid computing the probability intervals for every timepoint. This provides a wide variety of opportunities for the optimization of the implementation of this proof procedure. E.g. if we are combining two rules where the distributions in the heads are uniform it is sufficient to compute the probability intervals only for a single timepoint from each temporal constraint, which saves *a lot* of computations. Similar optimizations can be a applied to some other distribution functions such as, geometric or binomial.

7 nDimensional Probabilistic Databases and Logic Programming

Last but not least we are ready to describe our approach to dealing with probabilistic data when probabilities are distributed in n-space. This work is currently in the initial stages, and no formal descriptions of models will be included here. Rather than that, we outline our goals and approaches and present the plan for the research in this area.

The n-space in which the probabilities are distributed may be pure two- or three-dimensional space (e.g., information about positioning of ground troops and airplanes) or this may be two- / three- dimensional space + time (e.g., same information considered dynamically). Our **first goal** is to create a framework which would *successfully (and correctly) manipulate data regardless of the type of the underlying n-space.* Clearly, there are some intuitive differences between reasoning about probabilities distributed in 3D and reasoning about probabilities distributed in 2D + Time. We want our framework to incorporate these differences parametrically.

The **second goal** we want to pursue is ensure that *our new model for databases is a proper extension of Temporal Probabilistic Databases.* Indeed, one can think of temporal probabilistic data as 1D probabilistic data, and we want the restriction of our model onto 1D to coincide with TP Databases.

This latter goal, combined with a **third goal** of *achieving compact representation of the nD data in the model and finding efficient methods of manipulating it* suggests that the structure of the model will be very similar to the structure of TP Database model. Like there, we plan to introduce two types of relations: nD Probabilistic Relations which will serve as compact (implicit) representation of the data and Annotated Relations which will serve as the flat relational equivalents of nD Relations representing the data explicitly.

We will then define two relational algebras: one for explicit relations and one for implicit relations. The first algebra will serve as our *Declarative Semantics* defining the meanings of each operation, while the second algebra will be used as our *Operational Semantics* defining our *modus operandi* on the n-Dimensional probabilistic data.

In defining the relational algebras we will take into account our **fourth goal**: *to ensure that there is a clear intuitive meaning associated with each operation* as any time we perform an operation on our data we want to be able to interpret the results in a meaningful way.

For the nD Probabilistic Relational Algebra defined, we plan to develop efficient algorithms of performing operations, optimization, and other "standard" issues that arise in databases, such as updates and view maintenance. A prototype implementation is also planned, although it is unlikely that the implementation will be a part of this thesis.

The development of n-Dimensional Probabilistic Logic Programming language should follow the same steps as the development of Temporal Probabilistic Logic Programs. First, we are planning to extend Ng and Subramanian's PLPs ([42]) and Temporal Probabilistic Logic programs described in 6 onto the n-Dimensional case. This logic programming language will compute correctly the probabilities under the assumption of ignorance, just as its direct predecessor do. We then would like to build a hybrid version of nD Probabilistic Logic Programming Language which would allow us to parameterize the probabilistic conjunctions and disjunctions.

8 Related Work

In this section we summarize the related work.

8.1 Uncertainty in Logic Programming

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 [18]. Shapiro was one of the first to develop results in fuzzy logic programming [47]. Baldwin [2] was one of the first to introduce *evidential* logic programming and a language called FRIL. Van Emden [54] was the first to provide formal semantical foundations for logic programs that was later extended by Subrahmanian [50] and then completely generalized in a succession of papers by Blair and Subrahmanian[4], and Fitting[24], Ginsberg [26], and applied to databases by Kifer and Li [31] and Kifer and Subrahmanian [32]. *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 [42, 43, 44], developed techniques for probabilistic logic programming under the assumption of *ignorance* (see Section 2.1 for more detailed description of this work). Their work built upon earlier work on probabilistic logics due to Fagin, Halpern and and others [21, 23, 29, 1], Hailperin [28] and Nilsson [45].

In contrast, Kiessling and his group [27, 53, 46] 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 [38, 37, 39], which have been developing a parametric framework to represent varied probabilistic strategies in logic programs [39]. This work, which was developed slightly ahead and independently of our HPLP framework[11], can express some of what we try to express here, though there is no support for basic formulas in the heads of rules, and hence there is also no need for decomposition functions. However, by a rather complex translation that significantly increases the size of a program, and by introducing specially programmed functions, they can express some hp-programs with atoms (not basic formulas !) in the heads in their syntax. Thus, the two approaches share a common intersection, but neither appears to subsume the other.

In addition, 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. In contrast, our work offers a variety of cache-based query processing algorithms that complement their query optimization work.

8.2 Probabilistic Databases

A number of probabilistic database frameworks have been developed in the last decade.

Barbara et al. [3] develop a probabilistic data model and propose probabilistic operators. Their work is based on the assumption that probabilities of compound events can always be precisely determined, an assumption valid for few combination strategies. In contrast, we allow interval probabilities which permit margins of error in the probability data. In addition, when performing joins, they assume that Bayes' rule applies (and hence, as they admit upfront, they make the assumption that all events are independent). Also, as they point out, unfortunately their definition leads to a "lossy" join.

Cavallo and Pittarelli [7] propose a model for probabilistic relational databases. In their model, tuples in a probabilistic relation are interpreted using an exclusive or, meaning at most one of the data-tuples is assumed to be present in the underlying classical relation. This is a rather restrictive assumption, and we make no such assumptions. Furthermore, due to the above assumptions, Cavallo and Pittarelli [7] only propose probabilistic projection and join operations, but the other relational algebra operations are not specified.

An important paper on the topic is by Dey and Sarkar [14] who propose an elegant `1NF` approach to handling probabilistic databases. Their paper is a significant improvement to the work of Barbara et al. [3]. In particular, their framework (i) supports having uncertainty about some objects but certain information about others, (ii) uses first normal form which is easy to understand and use, (iii) introduces elegant new operations like conditionalization, and (iv) removes assumptions about deterministic keys prevalent in previous approaches. The `1NF` representation used by them is a special case of the annotated representation in our TP Database framework — as discussed in Section 5.2. Many of our operators generalize theirs — for instance, their notion of *union* clusters together all data-identical tuples and takes their `max`, difference clusters together all data-identical tuples and subtracts probability values, and their notion of projection clusters together all data-identical tuples and takes the sum of the tuples' probabilities (or 1, whichever is smaller) to be the

[11] Actually, the first paper that allows different conjunction and disjunction strategies to be incorporated into logic programs was [51].

probability. These computations are probabilistically legitimate only under some assumptions on the dependencies between the events involved. Our notion of *combination functions* generalize these substantially. In addition, their notion of join only applies under an independence assumption, which we have been able to remove through the notion of p-strategies. Similarly, our notion of compaction operations may be viewed as extensions of the two *coalesce* operations proposed by them — we propose whole families of coalesce operations in contrast, and our algebra uses such operations as parameters. Dey and Sarkar [14] propose some operations such as *conditionalization* and N^{th}-Moment that have no analogs in our paper, and deserve further study.

Our work in Temporal Probabilistic Databases builds on top of the ProbView system for probabilistic databases[36]. ProbView extends the classical relational algebra by allowing users to specify in their query, what probabilistic strategy (or strategies) should be used to parameterize the query. ProbView removed the independence assumption from previous works. A detailed description of analysis of ProbView can be found in Section 2.2.

8.3 Temporal Databases

8.3.1 Comparison with Dyreson and Snodgrass

Dyreson and Snodgrass [20] were one of the first to model temporal uncertainty using probabilities by proposing the concept of an *indeterminate instant.* Intuitively, an indeterminate instant is an interval of time-points with an associated probability distribution. They propose an extension of SQL that supports (i) specifying which temporal attributes are indeterminate, (ii) correlation credibility which allows a query to use uncertainty to modify temporal data — for example, by using an EXPECTED value correlation credibility, the query will return a *determinate* relation that retains the most probable time point for the event, (iii) ordering plausibility which is an integer between 1 and 100 where 1 denotes that any possible answer to the query is desired while 100 denotes that only a definite answer is desired, and (iv) specifying that certain temporal intervals are indeterminate. Dyreson and Snodgrass [20] develop a semantics for their version of SQL. In addition, they show how to compute probabilities of temporal relationships such as "event e_1 occurs before event e_2," 'event e_1 occurs at the same time as event e_2," etc., and provide efficient data structures to represent probability mass functions.

Our TP Database framework may be viewed as in improvement over the the Dyreson-Snodgrass framework in the manner described below. In addition, there is an important philosophical difference between our work and theirs — we are adding time to "kosher" probabilistic databases, while they are adding probability to "kosher" temporal databases.

SQL vs. Algebra. First and foremost, Dyreson and Snodgrass present a version of SQL for temporally indeterminate databases. In contrast, we present an *algebra* and prove that all our algebraic operations are correct.

Base Relations. In the Dyreson and Snodgrass [20] framework, the base relations are temporal relations. In effect, base relations in [20] may be viewed as special cases of TP-relations. The structure of our TP-tuple is more sophisticated as it allows us to express information about the probability of an event at a variety of timepoints. Also [20] uses point probabilities, while *interval probabilities* are at the heart of all our frameworks.

Conversely, there are some things that can be expressed in the Dyreson-Snodgrass framework [20] which we do not handle — for example, in TP Databases, we have assumed tuples have only one indeterminate temporal attribute (i.e. our events are instantaneous) while [20] allows more than one. We are currently handling non-instantaneous events in our Temporal Probabilistic Logic Programming framework. Also, we have no analog of correlation credibility or ordering plausibility.

Distribution Functions. In [20], all PDFs are assumed to be complete. In contrast, in TP-Databases, we allow both complete and incomplete PDFs. In fact, we noticed for the first time that determinate PDFs (all complete PDFs are determinate) guarantee linear time consistency checks for TP-databases.

Independence Assumptions. In [20], all indeterminate events are assumed to be independent. This assumption is valid for many applications, and invalid for others. In our work on TP Databases we continued the ProbView [36] tradition of working with events whose relationship is parameterized by *probabilistic strategies*.

Our algebra supports a host of operations that do not appear to be supported in the Dyreson-Snodgrass framework [20]. For instance, we provide whole families of compaction methods.

Semantics. We provide formal, model-theoretic descriptions of consistency in our paper for TP-relations, and provide very efficient means to check consistency (linear time) when determinate PDFs are used. When indeterminate PDFs are used, consistency checking is more complex.

8.3.2 Other Works

Snodgrass was one of the first to model indeterminate instances in his doctoral dissertation [49] — he proposed the use of a model based on three valued logic. Dutta [19] later proposed a fuzzy logic based approach to handle *generalized temporal events* — events that may occur multiple times.

This approach is also used by Dubois and Prade [17].

Gadia [25] proposes an elegant model to handle incomplete temporal information as well. He models values that are completely known, values that are unknown but are not to have occurred, values that are known if they have occurred, and values that are unknown even if they occurred. Gadia [25] shows that his model is sound. However, he makes no use of probabilistic information.

An important body of work is that of Koubarakis [33, 34] who proposes the use of constraints for representing temporal data. In this sense, our work is directly related and builds upon Koubarakis' work. Like our work on TP-Databases, Koubarakis uses constraints to represent when an even occurst. Koubarakis' framework allows stating the facts that event e_1 occurred between 8 and 11 AM, and that event e_2 occurs after 12pm. From this, we may conclude that event e_2 occurs after e_1 — our framework can support this conclusion as well. However, inside our TP-tuples, we cannot state that event e_2 occurs after e_1 — something we can do in a query, but which Koubarakis [33, 34] can explicitly encode in his tuples.

Another important body of work is that of Brusoni *et al.*[6] who developed a system called LaTeR. LaTeR restricts constraints to conjunctions of linear inequalities, as does Koubarakis' work. LaTeR makes a compromise — when tuples are inserted, it builds a constraint network (which increases insertion time), but this pays off because at query time, queries can be efficiently processed. We can benefit from this strategy in our work — as constraint networks are main memory data structures, an adaptation to disk-based structures would greatly enhance scalability. While this work is scheduled to be included in the next part of our study of TP Databases [12], it is not a part of this proposal.

9 Conclusions

We plan to study various problems in Database and Logic Programming models for reasoning with Probabilities distributed over time or n-Dimensional space. Although each framework sports its own features, they share the same background and there is a general approach behind them all. In designing our approaches to reasoning with probabilities in temporal and n-dimensional settings we paid or are planning to pay close attention to two key issues:

- *Clear common sense intuition behind our choice of operations.* This principle is really in the core of our Temporal Probabilistic frameworks, and we are planning on following it in the development of n-dimensional probabilistic models.
- *Compatibility between the models.* While all the models we consider are based on very similar principles, newer models tend to be more sophisticated and parts of the newer models may very well be quite different from similar parts of the older models (e.g., compare difference operation in TP Databases (Section 5.3.3) with the difference operation in ProbView ([36])). In cases like this, the first principle of providing more intuitive operations outweighs the compatibility principle, but we hope that while going from TP frameworks to nDimensional ones we will have better compatibility between the systems.

In the table below we provided the outline (and the timeline) of the work already done, currently in progress and scheduled for the nearest future (the original contributions of this proposal are highlighted by **bold san serif** font).

	Databases	Logic Programming
Probabilistic	ProbView, V.S. Lakshmanan, N. Leone, R. Ross and V.S. Subrahmanian [36] (1997)	PLPs, R. Ng, V.S. Subrahmanian [42, 43, 44] (1993-94)
		HPLPs. Theory: [9] (1997); Algorithms : [13] (1998)
Temporal Probabilistic	**TP Databases**. Theory: [11] (1997–99) Optimization : [12] (1999) - *work in progress*	**TPLPs**, [10] (1999) - *work in progress* **Hybrid TPLPs**, *future work*: Summer 1999
n-Dimensional Probabilistic	**n-Dimensional Probabilistic Databases** *work in progress*: Summer - Fall 1999	**n-Dimensional Probabilistic LPs** *future work:* Fall 1999 - Winter 200

The main contributions of our work on probabilistic reasoning include:

- Hybrid Probabilistic Logic Programs.
  - Definition of *probabilistic strategy*; study of properties of probabilistic strategies.
  - Definition of *syntax*, *model theory*, *fixpoint theory* and *generic proof procedure* for Hybrid Probabilistic Logic Programs.
  - Construction of *inference rule based proof procedure* for HPLPs.
  - Application of *tabulation techniques* to proof procedure for HPLPs (b-caches, b-cache update strategies).
  - Construction of *algorithms* and determining their complexity for *fixpoint computation*, *entailment* and *consistency* problems for HPLPs. Separation of HPLPs into classes for which solutions of the problems above have different complexities. Determining the classes of HPLPs for which these problems are solvable in *polynomial time*.

- Temporal Probabilistic Databases.
  - Definition of a *TP-relation*; its *annotation*; *interpretations* of TP-relations and annotated relations.
  - Definition of TATA (Theoretical Annotated Temporal Algebra) on annotated relations and of TPA (Temporal Probabilistic Algebra) on TP-relations; establishment of the connection between TATA and TPA.
  - Description of *query rewrite rules* for TPA.
- Temporal Probabilistic Logic Program.
  - Definition of *syntax*, *model theory* and *fixpoint semantics* for the extension of Ng and Subrahmanian's PLPs ([42]) onto temporal probabilistic framework.
  - Construction of the *algorithms* for *fixpoint computation* and *entailment* for the ground case.
  - Description of proof procedures.

In Section 7 we described in detail the plan for the study of n-Dimensional Probabilistic Databases and Logic Programs.

Our other plans for the thesis research include creating a *hybrid version* of Temporal Probabilistic Logic Programs (that would follow our HPLPs in spirit) and completing the study of optimization in TP-databases by creation of one or more *cost models* for evaluating the queries and combining them with our rewrite rules to obtain optimization algorithms. Study of *view maintainance* and *updates* for TP-databases is also planned, but it may be done outside the framework of the thesis.

References

[1] M. Abadi, J. Halpern (1994) *Decidability and Expressiveness For First-Order Logics of Probability*, Information and Computation, 112(1), pps. 1-36.

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

[3] 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.

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

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

[6] V. Brusoni, L. Console, P. Terenziani and B. Pernici. (1995) *Extending temporal relational databases to deal with imprecise and qualitative temporal information*, Proc. Intl. Workshop on Recent Advances in Temporal Databases (eds. S. Clifford and A. Tuzhilin), pps 3–22, Springer Verlag.

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

[8] 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.

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

[10] A. Dekhtyar, M. Dekhtyar and V.S. Subrahmanian. (1999) *Temporal Probabilistic Logic Programs*, in preparation.

[11] A. Dekhtyar, R.Ross and V.S. Subrahmanian. (1998) *Probabilistic Temporal Databases, I: Algebra*, available as University of Maryland tech report CS-TR-3987

[12] A. Dekhtyar, R. Ross and V.S. Subrahmanian. (1999) *Probabilistic Temporal Databases, II: Indexes, Query Optimization and Updates*, in preparation.

[13] M. Dekhtyar, A. Dekhtyar and V.S. Subrahmanian. (1999) *Hybrid Probabilistic Logic Programs:Algorithms and Complexity*, accepted to Uncertainty in AI-99, extended version available as University of Maryland tech report CS-TR-3969.

[14] D. Dey and S. Sarkar. (1996) *A Probabilistic Relational Model and Algebra*, ACM Transactions on Database System s, Vol. 21, 3, pps 339–369.

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

[16] 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.

[17] D. Dubois and H. Prade. (1989) *Processing Fuzzy Temporal Knowledge*, IEEE Transactions on Systems, Man and Cybernetics, 19, 4, pps 729–744.

[18] 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.

[19] S. Dutta. (1989) *Generalized Events in Temporal Databases* , in: Proc. 5th Intl. Conf. on Data Engineering, pps 118–126.

[20] C. Dyreson and R. Snodgrass. (1998) *Supporting Valid-Time Indeterminacy*, ACM Transactions on Database Systems, Vol. 23, Nr. 1, pps 1—57.

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

[22] R.Fagin, J. Halpern, Y.Moses, M. Vardi, *Reasoning About Knowledge*, MIT Press, 1995.

[23] R. Fagin, J. Y. Halpern and N. Megiddo. (1988) *A logic for reasoning about probabilities*, Information and Computation, 87(1/2):78-128, July/August 1990.

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

[25] S. Gadia, S. Nair and Y.C. Poon. (1992) *Incomplete Information in Relational Temporal Databases*, Proc. Intl. Conf. on Very Large Databases.

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

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

[28] T. Hailperin. (1984) *Probability Logic*, Notre Dame J. of Formal Logic 25, 3, pps 198-212.

[29] J. Halpern, (1989) *An Analysius of First Order Logics of Probability*, Artificial Intellegence, 46, pp. 311-350

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

[31] 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.

[32] 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.

[33] M. Koubarakis. (1994) *Database Models for Infinite and Indefinite Temporal Information*, Information Systems, Vol. 19, 2, pps 141–173.

[34] M. Koubarakis. (1994) *Complexity Results for First Order Theories of Temporal Constraints*, Proc. 4th Intl. Conf. on Principles of Knowledge Representation and Reasoning (KR-94), Bonn, Germany, pps 379–390.

[35] S. Kraus, Y. Sagiv and V.S. Subrahmanian. (1996) Representing and Integrating Multiple Calendars. University of Maryland Technical Report CS-TR-3751. Submitted for journal publication.

[36] 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.

[37] 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.

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

[39] V.S. Lakshmanan and N. Shiri. (1997) *A Parametric Approach with Deductive Databases with Uncertainty*, accepted for publication in IEEE Transactions on Knowledge and Data Engineering.

[40] .W. Lloyd (1987) *Foundations of Logic Programming*, Springer Verlag, 1987.

[41] E. Lowler. *Combinatorial Optimization: Networks and Matroids*, Holt, Reinhart & Winston, New York, 1976.

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

[43] 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.

[44] R. Ng and V.S. Subrahmanian. (1995) Stable Semantics for Probabilistic Deductive Databases, Information and Computation, 110, 1, pps 42-83.

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

[46] 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.

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

[48] J. Shoenfield. (1967) *Mathematical Logic*, Addison Wesley.

[49] R.T. Snodgrass. (1982) *Monitoring Distributed Systems: A Relational Approach*, PhD dissertation, Carnegie Mellon University.

[50] 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.

[51] V.S. Subrahmanian. (1988) *Generalized Triangular Norm and Co-Norm Based Semantics for Quantitative Rule Set Logic Programming*, Logic Programming Research Group Technical Report LPRG-TR-88-22, Syracuse University.

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

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

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

[55] C. Zaniolo, S. Ceri, C. Faloutsos, R. Snodgrass, V.S. Subrahmanian, and C. Zicari. (1997) *Advanced Database Systems*, Morgan Kaufman.