ABSTRACT

Title of Dissertation: REASONING WITH PROBABILITIES AND TIME

Alexander Dekhtyar, Doctor of Philosophy, 2000

Dissertation directed by: Professor V.S. Subrahmanian
Department of Computer Science

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 Databases and Logic Programming.

The two main contributions of this thesis are Hybrid Probabilistic Logic Programs (HPLPs) – a framework for reasoning about uncertain events, which takes into account different dependencies and relationships that may exist between the events and Temporal Probabilistic Databases (TP-Databases) – a framework for succinctly representing and

efficiently querying "temporal probabilistic information", i.e., probabilistic information about the occurrence of events in time. We also present a simple Temporal Probabilistic Logic Programming framework, which can be viewed as a logical extension of the work on Temporal Probabilistic Databases. While seemingly independent, HPLPs and TP-Databases have a number of similar ideas behind their design.

The work presented here has roots in two generic probabilistic frameworks: Probabilistic Databases, developed previously by Lakshmanan, Leone, Ross and Subrahmanian ([64]), which first introduced different ways of computing probabilities of complex events and Probabilistic Logic Programs of Ng and Subrahmanian [82]. Our work improves upon these previous results, as well as upon the work of Dyreson and Snodgrass [34, 35] on Temporal Indeterminacy.

In our work on Hybrid Probabilistic Logic Programs, we describe the syntax and semantics of the Logic Programming language for reasoning with probabilistic information in the presence of multiple assumptions about dependencies (relationships) between different events. We develop model theoretic and fixpoint semantics of this language, study a number of different sound and complete proof procedures and provide a comprehensive analysis of algorithms and complexity problems arising in the framework.

Temporal Probabilistic Databases are designed to compactly store the uncertainty in the information about the occurrence of different events expressed probabilistically. We define a data model for such compact storage, TP-relations and TP-tuples, as well as a more conventional data model (annotated relations) to express the information stored in TP-relations. We develop relational algebras for both data models and show that the relational algebra on TP-relations (Temporal Probabilistic Algebra) correctly implements the operations of the relational algebra on annotated relations (Theoretical Annotated Temporal Algebra). We also study the query rewrite rules Temporal Probabilistic Algebra.

Finally, following the ideas developed in TP-Database framework we define a Logic Programming language for reasoning with temporal probabilistic information. For this language we provide model theoretic and fixpoint semantics and design a simple sound and complete proof procedure.

REASONING WITH PROBABILITIES AND TIME

by

Alexander Dekhtyar

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2000

Advisory Committee:

Professor V.S. Subrahmanian, Chairman and Advisor
Professor Samir Khuller
Professor James Hendler
Professor Leana Golubchik
Professor Luuiqa Raschid

ACKNOWLEDGEMENTS

The author would like to express his appreciation for many colleagues and friends whose input helped shape this thesis.

Michael I. Dekhtyar has greatly influenced my choice of the field of study. His comments on the content of the thesis have also been extremely valuable. Mikhail A. Taitslin, my undergraduate advisor has also contributed greatly to my decision to pursue a Ph.D.

Robert Ross has participated with me in designing the TP-Database framework from the very beginning. Without him, this framework would have been very different.

I would like to thank Samir Khuller for the discussions of the matching algorithms and for pointing out how the generalized weighted matching problem (Section 7.2.3) for functions used in our probabilistic strategies (Section 3.2 can be reduced to standard weighted matching problem. Kyung Dong Ryu provided some information on the problem of resource management in computer networks.

Over the years, I have had numerous discussions and has heard a lot of useful comments from Terrence Swift, Joseph Halpern, Anil Nerode, Jack Minker, Bill Gasarch, Jurgen Dix, Thomas Eiter, Selçuk Candan, Eenjun Hwang, Fatma Özcan, Timothy J. Rogers, John Benton, Marat Fayzullin, Brian Postow, Simon Hawkin, Jason Ernst. Fatma Özcan and Jason Ernst also provided some useful comments on this thesis. Sean Luke's advice on typesetting of this thesis was invaluable.

The author would like to express his gratitude to the anonymous reviewers of [23] and [25] for multiple valuable comments.

Finally, the words of gratitude go to V.S. Subrahmanian, my graduate advisor, who provided his support and guidance throughout the entire research process that lead to this work.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Part I

Introduction and Preliminaries

Chapter 1

Introduction:Probabilistic Reasoning in Logic Programming and Databases

"Do not expect to arrive at certainty in every subject which you pursue. There are a hundred things wherein we mortals. . . must be content with probability, where our best light and reasoning will reach no farther."
– Isaac Watts

We are surrounded by uncertain information. Will the plane leave on-time ? Will the Dow-Jones index fall in tomorrow's trading ? Will it rain today ? Will the web link returned to me by a search engine lead to a document relevant to my search ?

Traditional database and Logic Programming frameworks deal with exact data. As the computational power of computers grows and the problems found in real applications become more and more complex, it becomes harder and harder to ignore the uncertainty that arises in many situations.

Our approach to this problem is twofold. First, we study the reasoning and management of uncertain data in classical relational databases and logic programming. Our expectation is that due to the fairly simple structure of data in these frameworks, it would be possible to (a) develop a deep understanding of how to represent uncertain information and conduct reasoning with it and (b) these methods will provide intuition about how uncertain data management and reasoning can be conducted in situations when the data is more complex. At the second stage of our research, we want to use what we developed at the first stage to achieve exactly that latter goal: design methods for managing and reasoning with complex, heterogeneous uncertain data.

This thesis is a step towards accomplishing the first goal. We will describe our plans for working towards achieving the second goal in Chapter 16.

In our work, we chose probabilities as the means to measure the uncertainty of the data. Out of all frameworks for dealing with uncertainty, probability theory can be distinguished by the following two properties:

(i) it is the best developed mathematical apparatus, from the very first definitions of probability in 1600-1700, and through the works of Laplas, Poisson, Chebyshev to Kolmogorov's axiomatization [57] in early 1930s, to current research in probability theory and statistics.

(ii) it is much better understood by non-mathematicians than any other formalism for dealing with uncertainty. While it is true that probability theory is most mathematically involved, a statement "The probability that X is true is 55%" is generally more "user-friendly" than "The fuzzy measure of X is 0.55". The roots of probability theory in gambling and the fact that probability is presented as a measure of repeated trials help the general understanding of its meaning by people well outside the mathematical community.

Our choice of a probabilistic approach to managing and reasoning with uncertain data means that out methods will be restricted to the situations when there is a probabilistic interpretation of uncertainty. So we do not set forth a claim that *any* situation in which uncertain data is present can be modeled by our methods. Rather, we concentrate on modeling the situations when the uncertainty in data can be probabilistically quantified. Even in such cases we argue that it is a good idea to tailor the design of the framework to the particular kind of probabilistic uncertainty, as this will allow us to exploit additional properties (if known) of the underlying data. In particular, we will concentrate our efforts on studying the properties of the *temporal probabilistic information*.

In the motivating examples that follow (Section 1.2) we illustrate the notion of temporal probabilistic information, and the challenges related to managing it in more detail. Generally speaking, temporal probabilistic information appears when we are dealing with data of the following kind: (i) we have exact descriptions of the events as they are supposed to happen, (ii) we do not know when they have occurred (or are supposed to occur) and (iii) we are able to obtain information about the probability of these events occurring at particular times. We believe that there are a number of interesting problems that may require dealing with information of this kind.

Below (Chapter 2) we will give an overview of previous research on dealing with uncertain information and uncertain reasoning both in probabilistic and non-probabilistic settings. We will be especially interested in comparing and contrasting these approaches with the results of our research described here. In Chapter 2, only general overview of the approaches will be given together with the analysis of their strengths and weaknesses in Sections 2.4. Later, in Chapter 15 we will describe these approaches in more detail and compare them to our work.

Our research in both Databases and Logic Programming is rooted in the previous research on uncertainty in Logic Programming and Databases done at University of Maryland in the 1990s. Two frameworks: Probabilistic Logic Programs (PLPs) [82, 83, 84] and Probabilistic Databases (ProbView) [64] respectively form the base for Hybrid Probabilistic Logic Programs and Temporal Probabilistic Databases described in this thesis. PLPs and ProbView will be described with a fair amount of detail to allow us to demonstrate how our work fixes their shortcomings.

1.1 Contributions

The material of this thesis can be broken down into three major parts, describing three frameworks that we have been developing:

- **Hybrid Probabilistic Logic Programs** deal with reasoning with probabilistic data in the presence of explicit information about the dependencies in the data.
- **Temporal Probabilistic Databases**[1] deal with management of *temporal probabilistic information* in a relational database setting.
- **Temporal Probabilistic Logic Programs** deal with reasoning with complex temporal probabilistic data.

Here are the major contributions of this thesis broken down into the three frameworks described above.

Hybrid Probabilistic Logic Programs [Part II]

- We have probabilistic strategies - a the means of expressing information about the relationship (e.g. independence, positive correlation), between different events. We have studied the properties of probabilistic strategies. [Chapter 3]

[1]Throughout this thesis we will use "Temporal Probabilistic Databases" and "Probabilistic Temporal Databases" as synonyms.

- We have defined hybrid probabilistic logic programs (hp-programs). Hp-programs utilize probabilistic strategies to encode relationships between events. [Chapter 4, Section 4.1]

- We have provided model-theoretic and fixpoint semantics for hp-programs and have shown that they coincide. [Chapter 4, Sections 4.2.1 and 4.2.2]

- We have studied a number of different proof procedures for hp-programs. Since a straightforward proof procedure for hp-programs turned out to be very inefficient, we designed a more efficient, dynamic inference rule-based proof procedure. [Chapter 5]

- We have also developed a proof procedure based on tabulation (i.e. trying to decrease amount of computation by storing the intermediate results of evaluating a query and using them to avoid redundant derivations). [Chapter 6]

- For each proof procedure described, we proved its soundness and completeness.

- We have designed algorithms for

  - finding the minimal model for an hp-program,

  - determining entailment of a query from an hp-program,

  - determining consistency of an hp-program.

  We have also analyzed algorithmic complexity of these problems. While we have shown that the entailment problem is NP-complete in general, we were able to find classes of hp-programs for which this problem has polynomial time complexity. [Chapter 7]

- We have studied proof complexity for hp-programs. We have analyzed the structures of proofs in a sound and complete inference system derived for hp-programs[2] and have shown that all theorems of this inference system have proofs of polynomial[3] size. [Chapter 7, Section 7.4]

Temporal Probabilistic Databases [Part III]

- We have defined Temporal Probabilistic tuples (tp-tuples), which store temporal probabilistic information. We have defined Temporal Probabilistic Relations (tp-relations) as collections of tp-tuples. [Chapter 9, Section 9.2]

- We have defined a notion of Annotated Relations as flat relational equivalents of tp-relations. A tp-relation can be converted into an annotated relation via the operation of Annotation. Annotated relations provide a good data model for understanding how the relational operations on temporal probabilistic data should operate. However, as the size of an Annotations of TP-relation depends on the granularity of the calendar, and can grow large as the granularity becomes finer and finer, annotated relations are not useful for practical implementation. [Chapter 9, Section 9.3.1]

- We have defined the notion of consistency for tp-relations and annotated relations and established the correspondence between consistency of tp-relations and their respective annotations. [Chapter 9, Section 9.3.2]

- We have defined Temporal Annotated Theoretical Algebra (TATA) on annotated relations which would serve as declarative semantics for the relational alge-

[2]This system is a modification of the system used in the dynamic proof procedure.

[3]In size of theorem, which includes the size of program.

bra on tp-relations. As mentioned above, annotated relations are a more convenient data model to define relational operations on, however, they are impractical. [Chapter 10]

- We have defined Temporal Probabilistic Algebra (TPA) on tp-relations. We have shown that the operations defined in TPA correctly (and efficiently) implement their theoretical, declarative analogs in TATA. We consider our main achievement to be the ability to translate the intuition about the relational operations on temporal probabilistic data in annotated form to the compact representation of this data, tp-relations provide.

  Among the operations defined in both algebras are Selection, Compaction, Projection, Cartesian Product, Join, Union, Intersection and Difference. In addition, TPA has special operations: Presentation, Compression and Tightening. [Chapter 11]

- Following the design of TP-relations and Temporal Probabilistic Algebra, Robert Ross has implemented a prototype Temporal Probabilistic Databases System. In a number of experiments it was shown that TPA operations on compact data (tp-relations) are faster than the respective TATA operations on annotated relations.

- We have commenced the study of query optimization in TPA/TATA. While the research on this is currently underway, we have included in this thesis the query equivalence results for TATA/TPA which we have obtained. [Chapter 12]

Temporal Probabilistic Logic Programs [Part III, Chapter 14]

- We have developed a Logic Programming language for reasoning about uncertain *instantaneous events*.

- We have provided model-theoretic and fixpoint semantics for Temporal Probabilistic Logic Programs (TPLPs).
- We have designed a simple proof procedure for queries in TPLP framework. This procedure is shown to be sound and complete w.r.t. the semantics of TPLPs.

From here, the thesis is organized in the following way. A few motivating examples complete this chapter. A general description of the previous and current work in Probabilistic Logic Programming and Probabilistic Databases together with an analysis of the strengths and weaknesses of the proposed frameworks form Chapter 2. Chapter 3 will describe one of the key concepts used in both Hybrid Probabilistic Logic Programs and Temporal Probabilistic Databases : probabilistic strategies. Part II (Chapters 4, 5, 6, 7) will describe our Hybrid Probabilistic Logic Programming framework, while Part III (Chapters 9, 10, 11, 12) is devoted to Temporal Probabilistic Databases. In Chapter 14 we introduce Temporal Probabilistic Logic Programs. A more detailed description of research in a number of fields related to our work (uncertainty in logic programming, uncertainty in databases, Bayesian Nets, Fuzzy Logic, and more) is contained in Chapter 15. We conclude the thesis with the description of a number of future directions of the research stemming out of the results in presented here (Chapter 16).

1.2 Motivating Examples

In this section we provide a number of examples that motivated our and past research efforts in the areas of reasoning with uncertain information and managing uncertain data. These examples come from different areas in and outside of Computer Science.

1.2.1 Logistics

"On time. Zero defects. Pick one." – Scott Adams

Logistics can be defined as the problem of getting the right stuff to the right place at the right time. To quote [97], "Most simply, logistics involves moving things from here to there and storing them along the way. The *logistics system* is the set of facilities, equipment, people, and operating policies that make possible the flow of goods (and transmission of information) from acquisition of the raw material through production and distribution into the hands of the customer."

Faced by every major and minor corporation, organization and government agency, logistics lies in the core of their day-by-day operations.

It is fair to say that logistics is management of flow. Every company that produces something or provides services usually has a set of suppliers which provide the goods, parts and/or materials necessary for the company to operate. For a grocery store chain its suppliers are companies that produce food, drugs and household items. For a bank the suppliers may be computer and office stationary vendors. For a car factory the suppliers would be the companies (or other plants belonging to the same auto-maker) producing spare parts and factory equipment. One of the tasks of the logistics system is to manage the flow of supplies to and through the company's physical and logical infrastructure.

For most of the companies the supply chain is in constant operation. Food has to be delivered to the grocery store on daily basis for perishables, and on weekly to monthly basis for other items. Parts for the cars need to arrive regularly as the production lines have to operate without interruption.

The process of transfer of supplies from the supplier to the consumer (our company in this case) is usually conducted by a transportation provider. Sometimes, a transportation provider will be affiliated with a either supplier or consumer, but more often it would be an independent vendor (a delivery company like UPS or DHL or a freight company like CSX).

Similarly, the company warehouse/headquarters may accept the supplies delivered to its door and then distribute it among it branches/retail outlets. In many cases a transportation provider will be involved in this as well.

Recently the main operating principle of many large companies became that of *lean inventories*. As the operating schedules become more and more dynamic and complicated, companies seek ways to cut their costs. And since traditionally, the storage space is expensive to maintain, many companies arrived to the conclusion that the best modus operandi for them is to use as little storage space as possible to maintain flawless and uninterrupted operations.

The desire of such companies to maintain lean inventories gave rise to so called *just in time logistics*. Its goal is to schedule supply deliveries and shipments at each company location in such a way as to insure (i) flawless operation of company and (ii) lean inventories. The key problem in this is that of scheduling. In order to maintain lean inventories and still have the production lines going one has to be able to schedule deliveries of supplies in such a way that they arrive just as the need in them appears.

One of the main problems encountered with the scheduling of deliveries and shipments in and out of a particular facility is in the fact that a number of parameters that need to be taken into account when scheduling is uncertain. In particular, the delivery schedule as executed by a transportation vendor is subject to a number of expected and unexpected delays. In some cases a delay of 2-3 hours is not a problem but for some other production lines a delay of 15-20 minutes is going to be significant since it may change among other things the workforce assignment at the facility.

Traditionally, all companies, posing both as suppliers and consumers as well as transportation providers track the supply chain very extensively. Thus, a supplier knows how much time it takes to fulfill any given order, and ship it. Transportation provider in its turn knows how much time every leg of a trip from supplier's facility to the consumer facility took. And consumer knows when a particular order was issued and when it arrived and what happened to it afterwards. So, between the three companies, every order is being heavily tracked and the information is being archived.

It is our assumption that the history of past deliveries can help facility coordinators in their day-by-day scheduling process. Indeed, if 90% of the time the truck with milk arrived 35-40 minutes late to a grocery store on Mondays, the manager would be well advised to assume that this is a likely situation for the upcoming Monday. In fact, it can be argued that the manager can assume that the probability that the truck will arrive 35-40 minutes late is around 90%.

To be able to use this information, the facility manager has to be able to issue queries asking for probabilities of certain events to his databases and receive meaningful answers. Since the original data is stored in the form a historic database with information about every single delivery made, there has to be a way to obtain the answers to such queries from this data. Since currently no database system allows for queries of this sort, and since the data is "not quite ready" to be used for the answers, it looks like some middleware is needed to achieve that goal. This middleware should be tailored to dealing with uncertain (probabilistic information), it should store the data in a format convenient for access and querying.

1.2.2 Scheduling/Prediction problems

Logistics problem is just one example from a large class of scheduling and prediction problems.

All scheduling and prediction problems have a similar structure. An agent (human or automated) entrusted with the task of solving a scheduling problem in real-time mode has to decide when, where and how certain actions have to be performed. Traditionally, the agent would have a set of rules placing certain conditions on the execution of the actions. The agent's task is to try to optimize performance of the system it monitors and operates. The task is aided by the fact that the agent possesses a historical database of the events as they have occurred in the system. We envision that this historical information can be used to find out information about the probability of certain events, occurring at particular times. This information in our opinion may be useful for agent's decision process.

A prediction problem is different from a scheduling problem in that the agent which monitors a given system is entrusted with the task of predicting its behavior. As in the case of scheduling, we assume that temporal probabilistic information extracted from the history of the system's past behavior can be used to in the prediction process.

Here are some more instances of the scheduling and prediction problems.

Resource Management in Computer Networks.

Consider a network of workstations all of which can be used to run scheduled jobs. A scheduling agent has the task of monitoring the individual computer workloads in the network and optimally scheduling new jobs by exploiting the unused resources on the workstations. In particular the agent tries to detect when a workstation becomes idle and upon making such a detection decide what jobs to schedule there. The latter decision may depend upon the agent's belief about the length of time the workstation would remain idle. Thus, in order to do proper scheduling the agent must be able to construct a prediction about period of idleness of the workstation. Clearly, it cannot be done without proper data. A historical database containing information about the idle times of all workstations in the network is a first step towards providing the agent with appropriate data. While each individual record in such historical database has little significance for the agent, the agent may combine information contained in many database records to construct the picture of the idle periods of any particular workstation. The statistics gathered in such a way may be used (especially if the size of the database is large) as an estimate of a probability distribution of the duration of idle times for the workstations. This data can then be queried and used by the agent for its scheduling decisions (e.g., if the probability that the owner of the workstation X leaves for at least 45 mins around 12pm is over 80%, the agent may decide to schedule jobs which take around 45 mins to complete on that workstation).

As in the logistics example, the caveat here is that the process of extracting data and creating necessary probability distributions on demand may take a lot of time and generate a lot of redundancy as the same distributions will be created over and over only to be discarded at the end of each iteration of the agent's operating cycle. A much better solution to this problem would be to provide a facility for storing the probabilistic information obtained from the historical database and to query it so as to suit the agent's needs.

Time Series Data Analysis.

There are various collections of data which had been accumulated over time. Examples of such data include information about the prices of shares at stock markets, meteorological information, sales figures and many more. In each case a number of measurements is taken and recorded at equal time intervals. People analyzing time series data are interested in finding different trends, seasonal fluctuations and other features in order to predict the possible future behavior of the system. The fact that the information being analyzed is of highly uncertain nature is recognized by everyone and Elaborate statistical methods are used to analyze time series data and try to construct predictions. However, traditionally applicability of statistical methods means the existence of an underlying model for the data being analyzed. In the cases when the primary goal is to actually find such a model, one may need to revert to data mining techniques. Here, it may make sense to express certain information in a form of probabilistic rules, e.g., "If the average temperature over the period of last 7 days was 85 F and current month is August, assume that the average temperature for the next week will be 82 - 86 F with probability 50 - 66 %" or "If the probability of 10 F temperature drop when it rains in June is above 50% then assume that the probability that the average temperature for June is over 87 F is 10 - 15%". The information currently contained in the dataset can serve to provide the initial probabilities. E.g., to check if the antecedent of the second rule holds we would have to look at the temperature fluctuations for every rain during every month of June recorded in the dataset[4] and compute the percentage of cases when the the temperature dropped at least 10 F. If the dataset is fairly large, we can assume that the computed percentage represents a good approximation of the actual probability and use it[5]. In order to be able to successfully analyze time series data in this case we will need (i) ability to express rules about probabilistic information and (ii) database support for elaborate queries on probabilistic data.

[4]We assume that we have all the necessary information in the dataset.

[5]Of course atmosphere scientists use much more sophisticated information about past and current weather patterns in their predictions.

1.2.3 Other Examples

Scouting robots.

Reconnaissance without the use of human scouts is now an integral 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 no humans were 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). It seems that if scouting robots are to be developed and deployed, the ability to recognize, store, query and reason with uncertain information (with uncertainty to be most likely expressed probabilistically) should be a part of their software design.

Information Retrieval

Information Retrieval has long been one of the areas where probabilistic methods had been widely used [19]. In fact, some researchers in this area use Datalog equipped with probabilities [38] to express retrieval rules. However, most of the probabilistic retrieval methods so far concentrate on the assumption that anytime there is a need to compute the probability of a conjunction of two statements, it is sufficient to assume that the statements are independent (and the probability of conjunction thus can be computed as the product of the probabilities of two statements). Such necessity appears fairly often during information retrieval, as in many cases we need to know the probability of relevance of a document that contains two keywords given the probabilities of relevance of a document containing just one of the keywords. Independence assumption allowed researchers in IR to develop a number of elegant and simple methods for probabilistic retrieval.

However, in many real life IR domains, the independence assumption is no longer applicable (e.g., in collection of texts on Computer Science one would expect that the distribution of terms “Turing machine” and “computability” is not independent, just as the distribution of terms “context-free grammar”, “parser” and “compiler” would be). It is desirable in this case to have the ability to express the relationship between different statements in IR domain (such as statements about term distribution) directly in the query language of an IR system.

Chapter 2

Background Information

2.1 Overview of Previous Work

There are a number of different ways to approach reasoning with uncertainty in logic programming and management of uncertain information in databases (see [109] for a survey). Frameworks based on fuzzy logic and multi-valued logics have been known for years. Most such frameworks incorporate uncertain reasoning elegantly into the logic programming language or database design. However, there are some restrictions on the applicability of these models to uncertain reasoning.

- Logics of reasoning about knowledge and belief [37] 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.

- Constraint logic programs [51, 76] and constraint databases [52, 53, 93], provide for an opportunity to represent indefinite information in databases [59, 61], (e.g., information about an occurrence of an event expressed w.r.t. the occurrence of another event, such as `''Jon came home after Jon left work''`. This qualitative approach to representing indefinite information is rather powerful, but it does not meaningfully quantify uncertainty.

- When multivalued logics and lattices based on them [56, 109] are used to simulate uncertain reasoning and to manage uncertain data. Each truth value in the logic will correspond to some 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 [108] 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. For example, if an event A has a fuzzy value of 0.5, fuzzy logic will assign 0.5 as the fuzzy value of the event $\neg A$. Then the fuzzy measure corresponding to the conjunction $A \wedge \neg A$ will be found as the minimum of the two values, i.e. 0.5. However, it is clear that both A and $\neg A$ cannot occur.

These and other similar concerns about non-probabilistic uncertain reasoning lead researchers to develop more complex frameworks based on probability theory. In chapter 15 we will spend more time discussing some non-probabilistic approaches to dealing with uncertain information.

In 1854 G.Boole published his work *The Laws Of Thought* [12] in which he described, amongst 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 ([46]) was one of the first to develop 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. ([36, 39, 47, 1]) The logics they proposed extended their modal logics of knowledge and belief ([37]), and it had been shown by Abadi and Halpern ([1]) that they cannot be finitely axiomatized. This was due to the great expressive power of the logics, in which one could make statements such as "The probability of probability of probability of event e is 0.15". While interesting from a theoretical point of view, this logic unfortunately could not serve as a basis for realistic database and logic programming models.

Another version of probabilistic logic is due to Nilsson [85] who described a model theory for probabilistic inference and a number of methods for 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 [82, 83, 84] who introduced a Probabilistic Logic Programming language (PLP), based on General Annotated Programs (GAPs)[56], 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 2.2.

Concurrently with our current research, V.S. Lakshmanan and F. Sadri in [66] and later V.S. Lakshmanan and N. Shiri in [67] 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.

A number of other approaches to Logic Programming with probabilistic information have also been proposed. Fuhr [38] developed Probabilistic Datalog for information retrieval. Lukasiewicz [71, 72] proposed a number of Probabilistic Logic Programming approaches and studied disjunctive probabilistic programs.

Several approaches to Probabilistic Logic Programming were created to address the problem of computing conditional probabilities. Among these is the work of Guntzer, Kiessling et. al [95, 45, 54], and the efforts to unite Bayesian Networks [86] and Logic Programming. The pioneering work in combining the two belongs to Poole [88]; Ngo and Haddawy extended it in [80, 81].

In Section 2.4 we will discuss a number of advantages and disadvantages of these frameworks, and in Chapter 15 (Section 15.1) we will compare them to our Hybrid Probabilistic Logic Programs in more detail.

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, with the notable exception of the work of Dubois and Prade on fuzzy databases [30] 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[15] 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 [7]. The key assumption in this model was that the probabilities of events can be always determined exactly. In addition, their models assumed independence between all events.

D. Dey and S. Sarkar in [28] proposed a 1NF approach to 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 (see Section 2.4).

Management of uncertain information in temporal databases has also been studied extensively. Koubarakis [59, 60, 61] presented a constraint-based framework for representing temporal indefinite information in databases. Gadia, Nair and Poon [41] presented a framework for representing incomplete information. One of the features of their framework was compact representation of information (see also [18]): one tuple would represent the entire history of a particular object/subject. The first work which attempted to quantify temporal uncertainty in databases was by Dubois and Prade [31] who used fuzzy set theory as the model for uncertainty.

Dyreson and Snodgrass [34, 35] were among the first who studied probabilities in temporal databases. Our work in that area correlates with [35] but is also different in many aspects. We have included a special comparison between our framework and theirs in Section 15.3.1.

The study of temporal uncertainty/indeterminacy in deductive databases historically followed from the study of uncertainty in temporal databases. Chittaro and Combi [17] proposed a framework for managing temporal indeterminacy in deductive databases based on event calculus of Kowalski and Sergot [62].

Last, but not least, among the frameworks for managing probabilistic information in Databases is the ProbView model of V.S. Lakshmanan, N. Leone, R. Ross and V.S. Subrahmanian described in [64]. 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, 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 2.3.

In Chapter 15 we devote Section 15.2 to the comparison of our approach to existing approaches. We then compare our work to the work in temporal uncertainty in 15.3, where we concentrate on the similarities and differences of our TP-Databases with the Dyreson, Snodgrass [34, 35] framework in Section 15.3.1 and talk about other approaches to dealing with temporal uncertainty in databases in Section 15.3.2.

In the next two sections of this chapter we give a more detailed overview of Probabilistic Logic Programs of Ng and Subrahmanian [82, 83, 84] and ProbView [64]. In Section 2.4 we will look at the strengths and weaknesses of these and other frameworks.

2.2 Probabilistic Logic Programs

"The first step towards knowledge is to know that we are ignorant."
– Richard Cecil

In [82], Ng and Subrahmanian proposed the following Probabilistic Logic Programming Language. They considered a function-free logical 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 the 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 [83, 84] Ng and Subrahmanian extended their language to allow

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

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

- 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 [12] has shown that under such an assumption (see also [46])

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

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. Furthermore 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 2.2.1 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 knowledge about the probability of event a based on knowledge of probability of 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 compute the best interval that can be successfully inferred about the event $\mathsf{b} \wedge \mathsf{c}$. There are no rules in P with this event as the head (such rules are not permitted by the language described in [82] but this restriction has been relaxed in [83, 84]). 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 $[0.6, 0.8]$. 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.

- Use of probability intervals instead of point probabilities. This has been discussed above.

- Use of ignorance assumption about the dependencies between events. This too has already been discussed.

- Different multiple derivations of the same formula. 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.

- Use of *intersection* to combine together the results of different derivations. 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 II), developed as a part of this work we free ourselves from the ignorance assumption about the dependencies between 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.

2.3 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 [64] 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 [64].

The key notion defined in [64] 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 preceding section, intervals, rather than points, are used to represent probabilities in this framework.

A simple example demonstrates this definition.

Example 2.3.1 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 $(\text{IMAGE1.GIF}, \mathsf{John}, \mathsf{London}, 0.1, 0.4)$ 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 relations.

A few important features of the ProbView framework had direct influence on the development of our Temporal 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

[1] In [64] 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.

(although as it will be described in Section 9 our temporal probabilistic tuples are a somewhat scaled down version of the probabilistic tuples).

- 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. However, 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 [64] this problem is addressed by introducing a notion of *generic conjunction* function. Generic conjunction functions take two intervals as inputs and return an interval as output. Generic conjunctions must satisfy certain axioms, described in Sectionsec:DBP-str of this thesis . Examples of generic conjunctions for different assumptions about the relationship between the events (ignorance, independence, positive correlation) are also given in that section.

  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 by just one. Clearly the problem is in computing the "new" probability interval. ProbView follows the works of Lakshmanan and Sadri [66] 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, however, that the compaction approach in [64] ("compaction by disjunction") is different from the "expert system" compaction approach (or in other words "compaction by intersection") adopted by Ng and Subrahmanian in [82] and demonstrated in Example 2.2.1. 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 2.3.1 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 in IMAGE3.GIF, i.e., this could be an image of some other person, currently unknown to the database.

2.4 Strengths and Weaknesses of Previous Research

We have described in detail Probabilistic Logic Programs and the Probabilistic Database framework ProbView, which will serve as a springboard for the frameworks described in this thesis. While detailed descriptions of other related work is given in Chapter 15, we will summarize some of the features of past and ongoing research here, and will try to point at potential avenues for improvement.

2.4.1 Reasoning with Probabilistic Information

Probabilistic logics developed since the mid-80s were the first frameworks for reasoning with probabilistic information. Nilsson [85] was the first to propose the use of possible worlds as the semantics for probabilistic logics. Basically, given a statement S it is either true or false. We can consider two sets of *possible worlds*, one where S is true and another one – where S is false. Our knowledge about the actual state of the world may be incomplete or faulty, hence, we may not be able to determine whether the actual world belongs to the first set or to the second. However, we may associate a probability p_1 (p_2) with the statement "Actual world belongs to the set where S is true (false)". In fact, since these are the only two possible outcomes, we will know that $p_2 = 1 - p_1$. In this case we can state that the probability that S is true is p_1.

Possible worlds models have been widely used in logic to represent the semantics of different modal logics in past years. Their use for probabilistic reasoning thus gives us some clear parallels between probabilistic and modal logics. These parallels have been explored in great detail by Bacchus [4, 5], who noticed a number of problems with this approach and proposed the use of probabilities over domains, and Halpern, et al. [36, 47, 39, 1]in a series of works on logics of probability. They have developed the theory of (first-order) probabilistic logics and explored the parallels between them and modal logics for representing knowledge and belief. At the same time, Halpern [47], following Bacchus, noticed that there are two notions of probability that can be successfully expressed by different methods in probabilistic logics. The probability distribution on the domain expresses probability as the measure of randomness in repeated trials and serves to represent statements like "probability that a randomly chosen x from the domain satisfies property ϕ is greater than or equal to $1/2$". On the other hand, probabilities on possible worlds in Halpern's model cover the probability as the means of expressing degrees of belief. Possible worlds are used to model the probability in a statement like "Tweety flies with probability 0.9".

While these studies shed light on different aspects of reasoning with probabilistic information, they also highlight problems with probabilistic logic. In [1], Abadi and Halpern study the decidability and expressiveness of the logics from [47]. It was shown there that both for probabilistic reasoning on domain and probabilistic reasoning on possible worlds the validity problem becomes undecidable in the presence of a single binary predicate (for probabilities on possible worlds even a single unary predicate makes the logic undecidable). While some bounded cases of probabilistic logics are shown to be decidable, the decision algorithm is of high computational complexity.

This suggests that such probabilistic logics are too complex to serve as the basis for logic programming frameworks, where we would want to have tractable decision algorithms. Indeed, there are some features of probabilistic logics which one can easily part with in designing a logic programming framework; e.g., the possibility to construct sentences of a form "the probability of the statement 'the probability that a randomly selected bird will fly is 0.4' is less than 0.5" need not be present in logic programming languages.

One of the main achievements of different Probabilistic Logic Programming approaches proposed over the years by Lakshmanan and Sadri [65, 66], Ng and Subrahmanian [82, 83, 84], Lukasiewicz [71], Fuhr [38], Ngo and Haddawy [80, 81], Kiessling, Guntzer and Thone [45, 54] is the fact that by restricting the expressive power of the proposed systems, some measure of tractability was achieved.

With a notable exception of [65, 66] the drawback of all previous work in Probabilistic Logic Programming is the fact that all the frameworks make specific assumptions about the relationships between the events considered in programs. For instance, Fuhr [38] builds his work upon the assumption of *independence* between all events in his model. This leads to a number of inconvenient syntactic and semantical decisions as even if the initial events in the system are independent, it is possible to come up with the example of the event constructed out of these initial events which will not be independent. Fuhr's Probabilistic Datalog spends a lot of effort tracking down these situations and carrying the inference path information together with the data in the derivations to prevent incorrect probability computations. Ngo and Haddawy propose a Probabilistic Logic Programming paradigm [80] equivalent to Bayesian Networks, and therefore in their system, the conditional independence assumption plays a key role.

Lukasiewicz [72], for some of his work chooses *positive correlation* assumption for his probabilistic logic programming framework. This assumption allows him to derive an elegant lattice-based theory for his probabilistic logic programs. However, while real-world situations where there are many independent events are not all that rare, the possibility of having a real world model where for any pair of events one of them implies the other one is fairly remote.

Ng and Subrahmanian in [82, 83, 84] took a different road. As described earlier (see Section 2.2) the inference in their Probabilistic Logic Programming framework proceeds under the assumption of *ignorance* of the relationship between the events. This forces the use of intervals, rather than point probabilities as knowing point probabilities of simple events may still lead to an interval as the probability of their conjunction or disjunction.

Lakshmanan and Sadri [66] define Probabilistic Deductive Databases which are based on bilattices of probabilistically interpreted confidence and doubt intervals. In their first framework they ascribe uncertainty to the inference and not to the data. In later work, their Probabilistic Deductive Database framework was extended to allow for different *modes* of computation for disjunction and conjunction of events (this latter work, however was done concurrently and independently of our work).

Research in Probabilistic Logic Programming has been largely concentrated on creating languages for expressing probabilistic reasoning and providing sound and complete semantics for them. Very little has been done in terms of assessing computational complexity of the proposed frameworks (the notable exceptions are the works of Lakshmanan and Sadri [66] who studied the complexity of the fixpoint computation for their semantics and Lukasiewicz [71] who provides results on the computational complexity of his framework). Outside of that, fixpoint computation, entailment and consistency (where applicable) problems associated with different frameworks have not, to our knowledge, been generally studied.

Our research in Probabilistic Logic Programming attempts to close these semantical and scope gaps. Our initial goal was to design and study properties of the PLP framework in those cases where the programmer has knowledge about relationships between the events. For this framework, we want to define a sound and complete semantics, find an efficient proof procedure, and design and evaluate algorithms for solving basic problems arising in the framework: fixpoint computation, entailment, and consistency of the program.

2.4.2 Managing Probabilistic Information.

When creating a framework for managing probabilistic information in databases, the following issues should be addressed:

- **Data Representation.**

Clearly, in order to manage probabilistic information, classical relations have to be augmented with structures storing it. In this respect probabilistic databases are similar to temporal databases: the key information stored in the relations has to be treated specifically, differently from how classical relational attributes get treated.

Barbara, Garcia-Molina and Porter in [7] already show that in order to accommodate their data model, in which non-key attributes may be *stochastic*, i.e. subject to probability distributions over sets of possible values and in fact may be *jointly distributed* with other non-key attributes, the structure of relations has to be significantly changed compared to the classical relational model. Similar changes were proposed in ProbView [64] and other probabilistic databases.

When choosing an appropriate data representation mechanism a number of issues have to be kept in mind:

- Compactness/Scalability.

  It is important for some applications that information about probability distributions over attribute values in relational tuples is stored compactly. While it may not be an issue when domains of uncertain attributes are fairly small, or when the relations are of manageable size, it quickly may become a dominating issue when the attribute values have huge domains. Temporal attributes present this problem in applications where time should have very small granularity, such as milliseconds. In such cases to explicitly express information about a probability of some event occurring in an interval of 30 minutes, one would have to store information about $30 \cdot 60 \cdot 1000 = 180000$ possibilities, which is prohibitive.

- Ease of Use.

While we will concentrate on the probabilistic query languages below, it has to be mentioned that the complexity of query languages depends among other things on the data model used in the database. Traditionally, the simpler the data representation is, the simpler the query language definitions will be.

Similarly, the more the data model resembles classical relational algebra, the more the relational operations on it will be similar to their classical relational counterparts.

This issue, however, may be in conflict with the issue of compactness of representation discussed above. In fact, the data model for storing probabilities of all instances explicitly will be the similar to the classical relational model, and thus relational operations on this model would be easily definable, but the overhead in the sizes of the relations may be such that more intricate data models and relational operations definitions will be required for successful manipulation of data.

– Expressiveness.

Another important issue deals with the kinds of probabilistic information expressible in the data model. How are probabilities represented? Point representations assume that exact probability is always known. Intervals provide a way to keeping track of probabilities when exact values are not known, or when the framework makes it impossible to compute exact probabilities for complex data even when the exact probabilities for the base data are known. Some probabilistic databases require that the probability distribution on each uncertain attribute in a tuple be complete, i.e., the sum of probabilities should be 1. Other frameworks allow one to model incompleteness in the data by either explicit "catch-all" cases or by simple omission.

- **Query Language.**

  New data models for probabilistic databases spell the necessity of redefining the traditional relational operations. This task, depending on a particular data model may be anywhere from straightforward (for models very similar to classical relations) to highly nontrivial (for models which differ significantly from classical relations). As mentioned above, simplicity of operation definitions may actually mean that the database framework is inadequate for managing probabilistic information.

  Syntactically, one may choose to modify relational algebra, tuple calculus or SQL (or any other declarative query language).

  Semantically, one need to be very careful when defining operations on probabilistic relations as the results of operations must be (i) *meaningful* and (ii) *obey probability theory*. This means that in the relation produced by a result of a probabilistic relational operation, the numbers stored in places of probabilities must really be probabilities of certain events/statements and must be consistent with the basic purpose of the given operation.

  Just as the expressiveness of the data model is an issue in designing a probabilistic database model, expressiveness of probabilistic query language[2] is also important. In particular, we are interested in what the underlying probabilistic concepts for some of the operations (cartesian product/join, projection) are and what data dependencies can be dealt with by different operations.

[2]Under a fairly general understanding of the term "expressiveness", and not the traditional notion of expressive power. We are more interested in what kind of relationships on the data can be managed (expressed) using the query language, and we allow ourselves to slightly abuse the term "expressiveness" here.

If we subject all probabilistic database frameworks (including the specialty frameworks, such as Temporal Indeterminacy of Dyreson, Snodgrass [34, 35]) to the analysis based on the outline above, we will notice that they all have advantages and disadvantages. Since our work in this thesis is devoted to managing Probabilistic Temporal information we will also look at these frameworks from the perspective of their potential applicability to managing this information.

The Probabilistic Data Management model of Barbara, Garcia-Molina and Porter [7] has an impressive set of features. Given a relation, any number of its non-key attributes may be declared to be stochastic, and each tuple may store any number of possible values (with probabilities) for each stochastic attribute. Also, it is possible to define joint probability distributions on collections of stochastic attributes (as opposed to defining a probability distribution for each stochastic attribute). The framework also allows one to represent incomplete information by adding a "$*$" value to the list of possible values for a stochastic attribute to represent the lack of information. Projections, selects and joins had been defined for this model.

Despite the richness of the framework and its considerable influence on the probabilistic database models that followed it, the model of [7] had a number of shortcomings. First, the model used only point probabilities, which meant that all probabilities had to be computed exactly. In the presence of missing information, this sometimes (due to the model's understanding of the concept of "missing information") makes it impossible to compute a correct point probability. Another important weakness of the model was the obligatory assumption of independence between different attributes with different probability distributions in the bodies. Looking at the expressiveness of the relational operations proposed by Barbara et. al., one can notice a number of desired features that were missing. First, the definition of projection assumed that only the non-key attributes were projected out. This left out an important case of projection as projecting out key attributes could have created data-identical tuples for which new methods would have been needed to make the relation probabilistic again (some other frameworks deal with similar issues by introducing operations called compaction or coalescence). A similar issue is the lack of traditional set-theoretical operations of intersection, union and difference of two probabilistic relations.

From the perspective of applicability of this framework to storing temporal probabilistic information, one may notice that the data representation is such that the probability distributions on attributes must be stored explicitly, and therefore, this data model may be prone to problems when storing information about thousands of timepoints in the same tuple.

Cavallo and Pitarelli in [15] proposed a distinctly different approach to probabilistic databases. In their data model, tuples are no different from the tuples in the classical relational model. But instead of associating with each relation its characteristic function (i.e., a function from the cartesian product of domains of all attributes in the relation to the set $\{0, 1\}$) Cavallo and Pitarelli associate a probability distribution. They define operations of projection and join(-projection) on their data, together with a number of probability-theory related measures of information content and information-loss during operations and concentrate on the procedures by which a classical relation can be converted into a probabilistic one.

While this work serves to highlight a number of important properties of probabilistic information, it also has a number of weaknesses. The most important one is the fact that each relation represents the probability distribution for the possible values of one statement/event (for comparison, this would be stored as a single tuple in Barbara et.al. model [7]). Only two relational operations are defined (although definition of select is more or less straightforward and the definition of projection does coalesce tuples which become identical after projecting attributes out; the probabilities are added in this case). It is not clear what the meaning of set-theoretical operations on this data should be. In addition to that this model is not convenient for expressing dependencies between different events. And as far as storing temporal probabilistic information, these relations are highly inefficient.

In [28] Dey and Sarkar extend the framework of Barbara et. al. [7]. They propose a 1NF relational model for probabilistic data and a full relational algebra. Their data model breaks the complex tuples of [7] into their flat relational equivalents, with a single probability value associated with each tuple. The simplicity of the data model allows for fairly straightforward definitions of relational operators (as for each operation, only one field, the probability needs special treatment).

Dey and Sarkar were the first ones to discuss the necessity of coalescence operations on probabilistic data models, and propose ways to deal with it. They define two coalescence operations $\oplus$ and $\odot$ which combine probabilities by adding them and taking the maximum respectively. These operations are used in the definitions of Union and Projection operations where the need for coalescence of tuples is natural[3].

However, further analysis of the relational algebra of Dey and Sarkar demonstrates that a number of operations produce counterintuitive, in our opinion, results. For example, according to [28], the result of $r - r'$ will be determined as follows: first, all tuples in r for which no value-identical tuples in r' exist will be put in the resulting relation. Next, for all value-identical tuples $d \in r$, $d' \in r'$, if $p(d) > p(d')$ the tuple d is added to

[3]After projection, data tuples which were different in the original relation may now be *value-identical* or as we call them in this thesis *data-identical*. In the union of two relations, data-identical tuples from different relations may easily appear.

the result with it probability attribute set to $p(d) - p(d')$.

Consider the following two relations: $r = \{d_1, d_2\}$ and $r' = \{d'\}$. Let $d_1 = (x, p_1)$, $d_2 = (y, p_2)$ and $d' = (y, p')$ and $p' < p_2$, where x and y are data parts of the tuples and p_1, p_2 and p' are their probability attributes.

In this case, the information in relation r is interpreted as follows: the probability that x is true is p_1, and the probability that y is true is p_2. Similarly, r' has information that the probability that y is true is p'. Now, $r - r'$ will consist of two tuples: $d_1 = (x, p_1)$ and $d'' = (y, p_2 - p')$. The information in d_1 in $r - r'$ can still be interpreted the same way: the probability that x is true is p_1. But now, can we really claim that the meaning of tuple d'' is "the probability that y is true is $p_2 - p'$" ? The problem is that either answer to this question gives rise to a problem. If we do want to store the probability that y is true in $r - r'$ then $p_2 - p'$ is clearly not the correct answer, as it is merely a difference in the probability estimates of r and r' for y. If we do not want to have probability of y in $r - r'$ then the problem is that now different tuples in $r - r'$ have different meaning as tuple d_1 still gets interpreted as containing the probability of x.

Similar problems exist with the definitions of other set-theoretical operations. As other weaknesses of the data model we should point out the use of points to represent probabilities, which forces the use of independence assumption for join-like operation.

ProbView framework [64] has already been described in detail, as our TP-Databases were influenced by that work. ProbView framework successfully fixes a number of problems with previous frameworks for managing probabilistic data, among which are: relational algebra with probabilistically sound operations, use of intervals to represent probabilities, ability to represent incomplete information, use of different assumptions about the relationship between events for join/cartesian product and projection. However, there are still a number of places where ProbView falls short. Out of the two formats for storing probabilistic data: the compact one, similar to [7] and the annotated one, similar to [28] the latter was chosen are the underlying data model for the relational algebra. To quote the authors: " Probabilistic relations are a convenient and expressive device for modeling uncertainty and incompleteness. However, probabilistic relations are …not directly amenable for algebraic manipulation. Also, it is unclear how a generic algebra …can be defined on probabilistic relations …" [64]. This presents a potential problem for the applications (such as temporal probabilistic data management) with large domains of uncertain attributes.

In the work of Dyreson and Snodgrass [34, 35] on Valid-Time Indeterminacy the underlying data model comes from Temporal, not Probabilistic databases. The advantage of this approach is that the data model in [34, 35] can use the wealth of research in temporal databases to represent temporal information in a compact form and then tailor the probabilistic information to fit the temporal data. Dyreson and Snodgrass were the first to use *probability distribution functions* to define probabilities over a set of timepoints (instead of explicit probability intervals for each attribute value as in probabilistic database models described above). The fact that probabilistic information was secondary to temporal data however also served to somewhat limit the framework. For example, only complete information could be stored in the relations described in [34, 35]. A more detailed description of this work and comparison to our results can be found in Section 15.3.1.

All in all, while there has been considerable body of work in Probabilistic Databases, we note that all the existing frameworks have one or more weaknesses when analyzed according to the scheme presented above. While in this thesis we will present a probabilistic database framework for dealing with a specific kind of uncertainty, we still want our model to be free as possible of the weaknesses highlighted in this section .

Chapter 3

Probabilistic Strategies.

One of the key concepts used extensively in our work is that of a ***probabilistic strategy***.

Intuitively, a ***probabilistic strategy*** (or ***p-strategy***) can be viewed as a way of computing probabilities of complex events given probabilities of simple events that constitute them under certain assumptions about the relationships between the events in question. A simple example would be computing the probability of an event $a \wedge b$ given $p(a)$ and $p(b)$ assuming that a and b are conditionally independent.

Lakshmanan, Leone, Ross and Subrahmanian we the first to propose the means of dealing with the problem of computing probabilities of complex events from probabilities of simple event in [64] in a database framework. There, probabilistic conjunction $\otimes_\rho$ and probabilistic disjunction $\oplus_\rho$ had been introduced. For each operation type, a collection of axioms was provided.

In our work on extending Probabilistic Database framework with time we will employ these very notions of probabilistic operations. However, for extending Logic Programming frameworks, some additional work has to be done because, in logic programming, we sometimes need to know the answer to the question: "given the probability of a complex event, estimate the probabilities of the simple events constituting it".

In what follows, we will first describe probabilistic conjunction and disjunction strategies as defined in [64]. Then, we will formally define a notion of a probabilistic strategy (p-strategy) which allows for computation of probabilities both ways: from simple to complex events and the other way around. We will then study some basic properties of p-strategies.

We will slightly abuse notation and will use the phrase "probabilistic strategy" ("p-strategy") to refer to both the definitions in Sections 3.1 and 3.2 as the use of these definitions in our frameworks is disjoint (temporal probabilistic databases will rely on the definitions from Section 3.1 and logic programming frameworks will employ definitions from Section 3.2).

3.1 Probabilistic Conjunctions and Disjunctions in ProbView.

The material in this subsection is not new and follows the treatment in [64] and its reinstatement in [25].

Given the probabilities p_1 and p_2 of events e_1 and e_2, how do we compute the probability p of compound event $(e_1 \wedge e_2)$? As argued in [64], the answer depends on the relationship between e_1 and e_2. For instance if e_1 and e_2 are mutually exclusive, p should be zero; if e_1 and e_2 are independent of each other, p should be $(p_1 \cdot p_2)$. A similar situation arises when computing the probability of $(e_1 \vee e_2)$. We address these problems by introducing probabilistic conjunction strategies and probabilistic disjunction strategies.

Before proceeding, recall that intervals obey the following definitions/properties:

1. $[L_1, U_1] \leq [L_2, U_2]$ iff $(L_1 \leq L_2 \wedge U_1 \leq U_2)$.

2. $[L_1, U_1] \geq [L_2, U_2]$ iff $(L_1 \geq L_2 \wedge U_1 \geq U_2)$.

3. $[L_1, U_1] \subseteq [L_2, U_2]$ iff $(L_1 \geq L_2 \wedge U_1 \leq U_2)$.

4. $[L, U] = ([L_1, U_1] \cap [L_2, U_2])$ iff $(L = \max(L_1, L_2) \wedge U = \min(U_1, U_2) \wedge L \leq U)$.

Definition 3.1.1 (probabilistic conjunction strategy) *Let events e_1, e_2 have probabilistic intervals $[L_1, U_1]$ and $[L_2, U_2]$ respectively. Then a* probabilistic conjunction strategy *is a binary operation $\otimes$ which uses this information to compute the probabilistic interval $[L, U]$ for event $(e_1 \wedge e_2)$. When the events involved are clear from context, we use "$[L, U] = [L_1, U_1] \otimes [L_2, U_2]$" to denote "$(e_1 \wedge e_2, [L, U]) = (e_1, [L_1, U_1]) \otimes (e_2, [L_2, U_2])$". Every conjunctive strategy must conform to the following probabilistic postulates:*

1. Bottomline: $([L_1, U_1] \otimes [L_2, U_2]) \leq [\min(L_1, L_2), \min(U_1, U_2)]$.

2. Ignorance: $([L_1, U_1] \otimes [L_2, U_2]) \subseteq [\max(0, L_1 + L_2 - 1), \min(U_1, U_2)]$.

3. Identity: *When $(e_1 \wedge e_2)$ is consistent and $[L_2, U_2] = [1, 1]$, $([L_1, U_1] \otimes [L_2, U_2]) = [L_1, U_1]$.*

4. Annihilator: $([L_1, U_1] \otimes [0, 0]) = [0, 0]$.

5. Commutativity: $([L_1, U_1] \otimes [L_2, U_2]) = ([L_2, U_2] \otimes [L_1, U_1])$.

6. Associativity: $(([L_1, U_1] \otimes [L_2, U_2]) \otimes [L_3, U_3]) = ([L_1, U_1] \otimes ([L_2, U_2] \otimes [L_3, U_3]))$.

7. Monotonicity: $([L_1, U_1] \otimes [L_2, U_2]) \leq ([L_1, U_1] \otimes [L_3, U_3])$ *if* $[L_2, U_2] \leq [L_3, U_3]$.

$\square$

A brief explanation of Ignorance axiom is in order. Boole proved in 1854 [12] that if events e_1, e_2 are known to have probabilities in the intervals $[L_1, U_1], [L_2, U_2]$, and we do not know anything about the relationship between these two events, then the best that can be said about the probability of $(e_1 \wedge e_2)$ is that it lies in the interval shown above. This forms the basis for numerous pieces of work in the AI and deductive database literature ([39, 82, 84] to name a few). This axiom merely says that if we know something about the dependency between e_1, e_2, then we must be able to infer a tighter probability interval than complete ignorance about dependencies would allow us to infer.

The following are some sample conjunctive strategies:

- Use the $\otimes_{ig}$ (ignorance) operator when we do not know the dependencies between e_1 and e_2.
  $([L_1, U_1] \otimes_{ig} [L_2, U_2]) = [\max(0, L_1 + L_2 - 1), \min(U_1, U_2)]$.

- Use the $\otimes_{pc}$ (positive correlation) operator when the overlap between e_1 and e_2 is maximal.
  $([L_1, U_1] \otimes_{pc} [L_2, U_2]) = [\min(L_1, L_2), \min(U_1, U_2)]$.

- Use the $\otimes_{nc}$ (negative correlation) operator when the overlap between e_1 and e_2 is minimal.
  $([L_1, U_1] \otimes_{nc} [L_2, U_2]) = [\max(0, L_1 + L_2 - 1), \max(0, U_1 + U_2 - 1)]$.

- Use the $\otimes_{in}$ (independence) operator when e_1 and e_2 are independent.
  $([L_1, U_1] \otimes_{in} [L_2, U_2]) = [L_1 \cdot L_2, U_1 \cdot U_2]$.

Note that we use the more general notion of a probability interval $[L, U] \subseteq [0, 1]$ instead of a point probability $p \in [0, 1]$; intervals allow us to reason about the probabilities of compound events (through operators such as $\otimes_{ig}$) without making traditional assumptions like independence [64].

Probabilistic conjunctions will be useful when describing TATA and TPA semantics for *cartesian products* (§10.6, §11.7).

Definition 3.1.2 (probabilistic disjunction strategy) *Let events e_1, e_2 have probabilistic intervals $[L_1, U_1]$ and $[L_2, U_2]$ respectively. Then a* probabilistic disjunction strategy *is a binary operation $\oplus$ which uses this information to compute the probabilistic interval $[L, U]$ for event $(e_1 \vee e_2)$. When the events involved are clear from context, we use "$[L, U] = [L_1, U_1] \oplus [L_2, U_2]$" to denote "$(e_1 \vee e_2, [L, U]) = (e_1, [L_1, U_1]) \oplus (e_2, [L_2, U_2])$". Every disjunctive strategy must conform to the following probabilistic postulates:*

1. Bottomline: $([L_1, U_1] \oplus [L_2, U_2]) \geq [\max(L_1, L_2), \max(U_1, U_2)]$.
2. Ignorance: $([L_1, U_1] \oplus [L_2, U_2]) \subseteq [\max(L_1, L_2), \min(1, U_1 + U_2)]$.[1]
3. Identity: $([L_1, U_1] \oplus [0, 0]) = [L_1, U_1]$.
4. Annihilator: $([L_1, U_1] \oplus [1, 1]) = [1, 1]$.
5. Commutativity: $([L_1, U_1] \oplus [L_2, U_2]) = ([L_2, U_2] \oplus [L_1, U_1])$.
6. Associativity: $(([L_1, U_1] \oplus [L_2, U_2]) \oplus [L_3, U_3]) = ([L_1, U_1] \oplus ([L_2, U_2] \oplus [L_3, U_3]))$.
7. Monotonicity: $([L_1, U_1] \oplus [L_2, U_2]) \leq ([L_1, U_1] \oplus [L_3, U_3])$ *if* $[L_2, U_2] \leq [L_3, U_3]$.

$\square$

The following are some sample disjunctive strategies:

- Use the $\oplus_{ig}$ (ignorance) operator when we do not know the dependencies between e_1 and e_2.
  $([L_1, U_1] \oplus_{ig} [L_2, U_2]) = [\max(L_1, L_2), \min(1, U_1 + U_2)]$.

[1] The rationale for this axiom is similar to that described for Ignorance in conjunction strategies earlier. This expression was also derived by Boole in 1854 [12].

- Use the $\oplus_{pc}$ (positive correlation) operator when the overlap between e_1 and e_2 is maximal.

  $([L_1, U_1] \oplus_{pc} [L_2, U_2]) = [\max(L_1, L_2), \max(U_1, U_2)]$.

- Use the $\oplus_{nc}$ (negative correlation) operator when the overlap between e_1 and e_2 is minimal.

  $([L_1, U_1] \oplus_{nc} [L_2, U_2]) = [\min(1, L_1 + L_2), \min(1, U_1 + U_2)]$.

- Use the $\oplus_{in}$ (independence) operator when e_1 and e_2 are independent.

  $([L_1, U_1] \oplus_{in} [L_2, U_2]) = [L_1 + L_2 - (L_1 \cdot L_2), U_1 + U_2 - (U_1 \cdot U_2)]$.

The intuitive meaning of the axioms for p-strategies is described in the next section.

As conjunctive and disjunctive probabilistic strategies are commutative and associative, we can extend the definition of either strategy to apply to more than two arguments. We adopt the notations $([L_1, U_1] \otimes \ldots \otimes [L_k, U_k])$ and $([L_1, U_1] \oplus \ldots \oplus [L_k, U_k])$ to represent this generalization.

3.2 Probabilistic Strategies.

As mentioned above, for any pair of 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 2.2 the precise probability of both conjunction and disjunction cannot be computed, but intervals in which these probabilities lie can be precisely established. We repeat here the bounds found by Boole in [12] :

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

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 ([64]).

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 closed 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 3.2.1 *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 3.2.2 • *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 the probability interval of a compound event, it returns the set of all pairs of probability intervals for the simple events which could have generated this probability interval.

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 by 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 this interval contains only one point. Thus the intuition behind the inclusion monotonicity principle can be termed as the idea of "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 is the case that $c([a, b], [c, d]) = [\alpha, \beta]$. This guarantees 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 this work we will be interested in another property of composition functions: *continuity*.

Definition 3.2.3 *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.*

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

3.2.1 Examples of P-strategies

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

- **Independence.**

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

  – The conjunctive p-strategy $inc = \langle c_{inc}, d_{inc} \rangle$ is defined as:

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

  – The disjunctive p-strategy $ind =< c_{ind}, d_{ind} >$ is defined as:

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

  $$d_{ind}([a, b]) \ni \langle [a_1, b_1], [a_2, b_2] \rangle \in \mathcal{C}[0, 1] \times \mathcal{C}[0, 1]$$

  iff

  $$a_1 + a_2 - a_1 a_2 = a \quad \text{and} \quad b_1 + b_2 - b_1 b_2 = b$$

- **Ignorance.**

  When nothing is known about the relationship between the events we are forced to use p-strategies that reflect ignorance [12, 36, 85, 82, 84, 64, 66]. $igc = \langle c_{igc}, d_{igc} \rangle$ below is a conjunctive ignorance strategy, while $igd = \langle c_{igd}, d_{igd} \rangle$ is a disjunctive ignorance strategy.

– Conjunctive ignorance p-strategy

$igc = \langle c_{igc}, d_{igc} \rangle$, where

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

$$d_{igc}([a, b]) \ni \langle [a_1, b_1], [a_2, b_2] \rangle$$

iff

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

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

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

– Disjunctive ignorance p-strategy

$igd = \langle c_{igd}, d_{igd} \rangle$, where

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

$$d_{igd}([a, b]) \ni \langle [a_1, b_1], [a_2, b_2] \rangle$$

iff

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

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

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

Sometimes we will use ig instead of igc or igd whenever it is clear from the context whether a conjunctive or disjunctive strategy is under consideration.

- **Positive Correlation.**

Sometimes we know that the occurrence of event e_1 implies the occurrence of event e_2 (e.g., one would assume that `''Jon rides a bus''` would imply `''Jon` `bought a ticket''`). Below are conjunctive and disjunctive strategies for this case.

- Conjunctive p-strategy

  $pcc = \langle c_{pcc}, d_{pcc} \rangle$, where

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

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

  iff

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

  and

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

- Disjunctive p-strategy

  $pcd = \langle c_{pcd}, d_{pcd} \rangle$, where

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

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

  iff

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

  and

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

- **Negative correlation.**

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

$$ncd = \langle c_{ncd}, d_{ncd} \rangle,$$

where

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

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

such that:

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

$$\text{if } a < 1 \; then \; a_1 + a_2 = a$$

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

$$\text{if } b < 1 \; then \; b_1 + b_2 = a$$

3.2.2 Validity of Examples

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

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

Proof.

First we notice that all the composition functions under consideration satisfy the axiom of Separation. Indeed in the definition of every composition function c_{inc}, c_{igc}, c_{pcc},

c_{ind}, c_{igd}, c_{pcd}, c_{ncd} the lower bound of the result is dependent only on the lower bounds of the arguments and similarly, the upper bound of the result depends only on the upper bounds of the arguments. Also, we notice that all composition functions mentioned above are continuous as all the functions that compute their lower and upper bounds are continuous in both arguments. Now, we prove the rest of the axioms for each individual strategy.

- inc is a conjunctive coherent p-strategy.

  – inc is a **conjunctive** p-strategy.

    Let us establish that inc satisfies all the axioms of a conjunctive p-strategy.

    1. Commutativity.

       $c_{inc}([a_1, b_1], [a_2, b_2]) = [a_1a_2, b_1b_2] = [a_2a_1, b_2, b_1] = c_{inc}([a_2, b_2], [a_1, b_1])$

    2. Associativity.

       $c_{inc}(c_{inc}([a_1, b_1], [a_2, b_2]), [a_3, b_3]) = c_{inc}([a_1a_2, b_1b_2], [a_3, b_3]) = [a_1a_2a_3, b_1b_2b_3] = [a_1(a_2a_3), b_1(b_2b_3)] = c_{inc}([a_1, b_1], [a_2a_3, b_2b_3]) = c_{inc}(([a_1, b_1], c_{inc}([a_2, b_2], [a_3, b_3])$

    3. Inclusion Monotonicity.

       Let $[a_1, b_1] \subseteq [a_3, b_3]$, i.e. $a_1 \geq a_3 \geq 0$ and $0 \leq b_1 \leq b_3$.

       $c_{inc}([a_1, b_1], [a_2, b_2]) = [a_1a_2, b_1b_2]$.

       $c_{inc}([a_3, b_3], [a_2, b_2]) = [a_3a_2, b_3b_2]$.

       $a_1 \geq a_3 \geq 0$ implies $a_1a_2 \geq a_3a_2$; $0 \leq b_1 \leq b_3$ implies $b_1b_2 \leq b_3b_2$, which in turn, means that $[a_1a_2, b_1b_2] \subseteq [a_3a_2, b_3b_2]$.

    4. Bottomline.

$c_{inc}([a_1, b_1], [a_2, b_2] = [a_1 a_2, b_1 b_2]$. Since $0 \leq a_1, a_2 \leq 1$, $a_1 a_2 \leq a_1$ and $a_1 a_2 \leq a_2$, i.e. $a_1 a_2 \leq \min(a_1, a_2)$.

Similarly, since $0 \leq b_1, b_2 \leq 1$, $b_1 b_2 \leq \min(b_1, b_2)$.

This implies $[a_1 a_2, b_1 b_2] \leq_t [\min(a_1, a_2), \min(b_1, b_2)]$.

5. Identity.

   $c_{inc}([a, b], [1, 1]) = [a \cdot 1, b \cdot 1] = [a, b]$

6. Annihilator.

   $c_{inc}([a, b], [0, 0]) = [a \cdot 0, b \cdot 0] = [0, 0]$.

- inc is a **coherent** p-strategy.

  We know that $d_{inc}([a, b]) = \{\langle [a_1, b_1], [a_2, b_2] \rangle | [a_1 a_2, b_1 b_2] = [a, b]\}$, i.e., $\langle [a_1, b_1], [a_2, b_2] \rangle \in d_{inc}([a, b])$ **iff** $c_{inc}([a_1, b_1], [a_2, b_2]) = [a, b]$, which means that *inc* is a coherent p-strategy.

- igc is a conjunctive coherent p-strategy.

  - igc is a **conjunctive** p-strategy.

    Let us establish that igc satisfies the axioms of conjunctive p-strategy.

    1. Commutativity.

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

    2. Associativity.

       $c_{igc}(c_{igc}([a_1, b_1], [a_2, b_2]), [a_3, b_3]) = c_{igc}([\max(0, a_1 + a_2 - 1),$
       $\min(b_1, b_2)], [a_3, b_3]) = [\max(0, \max(0, a_1 + a_2 - 1) + a_3 - 1),$
       $\min(\min(b_1, b_2), b_3)] = [\max(0, 0 + a_3 - 1, a_1 + a_2 + a_3 - 2),$
       $\min(b_1, b_2, b_3)] = [\max(0, a_1 + a_2 + a_3 - 2), \min(b_1, b_2, b_3)] =$
       $[\max(0, a_1 + \max(0, a_2 + a_3 - 1) - 1), \min(b_1, \min(b_2, b_3))] =$

$c_{igc}([a_1, b_1], c_{igc}([a_2, b_2], [a_3, b_3]))$.

3. Inclusion Monotonicity.

   Let $[a_1, b_1] \subseteq [a_3, b_3]$, i.e. $a_1 \geq a_3 \geq 0$ and $0 \leq b_1 \leq b_3$.

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

   $c_{igc}([a_3, b_3], [a_2, b_2]) = [\max(0, a_3 + a_2 - 1), \min(b_3, b_2)]$.

   Since $a_1 \geq a_3 \geq 0$, $a_1 + a_2 - 1 \geq a_3 + a_2 - 1$, we get: $\max(0, a_1 + a_2 - 1) \geq \max(0, a_3 + a_2 - 1)$.

   Since $0 \leq b_1 \leq b_3$, $\min(b_1, b_2) \leq \min(b_3, b_2)$. From this we obtain $c_{igc}([a_1, b_1], [a_2, b_2]) = [\max(0, a_1+a_2-1), \min(b_1, b_2)] \subseteq [\max(0, a_3 + a_2 - 1), \min(b_3, b_2)] = c_{igc}([a_3, b_3], [a_2, b_2])$.

4. Bottomline.

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

   Clearly, $0 \leq \min(a_1, a_2)$. Also since $a_2 \leq 1$, $a_1+a_2-1 \leq a_1$. Similarly, since $a_1 \leq 1$, $a_1 + a_2 - 1 \leq a_2$. The two inequalities allow us to deduce that $a_1 + a_2 - 1 \leq \min(a_1, a_2)$ and, therefore, $\max(0, a_1 + a_2 - 1) \leq \min(a_1, a_2)$.

   From this we obtain

   $[\max(0, a_1 + a_2 - 1), \min(b_1, b_2)] \leq_t [\min(a_1, a_2), \min(b_1, b_2)]$.

5. Identity.

   $c_{igc}([a, b], [1, 1]) = [\max(0, a + 1 - 1), \min(b, 1)] =$

   $[\max(0, a), \min(b, 1)] = [a, b]$.

6. Annihilator. $c_{igc}([a, b], [0, 0]) = [\max(0, a + 0 - 1), \min(b, 0)] = [0, 0]$.

– igc is a **coherent** p-strategy. Let $[a, b], [a_1, b_1], [a_2, b_2] \in \mathcal{C}[0, 1]$.

  * Let $\langle [a_1, b_1], [a_2, b_2] \rangle \in d_{igc}([a, b])$. We consider the following two possibilities for the relationships between a, a_1 and a_2:

- $\cdot$ $a = 0$ and $a_1 + a_2 \leq 1$. In this case $\max(0, a_1 + a_2 - 1) = 0 = a$.
- $\cdot$ $a > 0$ and $a_1 + a_2 - 1 = a$.

As far as the relationships between b, b_1 and b_2 are concerned, by definition of d_{igc}, $b = b_1$ if $b_1 \leq b_2$ an $b = b_2$ if $b_2 \leq b_1$, which means that $b = \min(b_1, b_2)$.

Combining our results together we obtain: $c_{igc}([a_1, b_1], [a_2, b_2]) = [\max(0, a_1 + a_2 - 1), \min(b_1, b_2)] = [a, b]$.

$*$ Let $c_{igc}([a_1, b_1], [a_2, b_2]) = [a, b]$. Then, we know that $[a, b] = [\max(0, a_1+a_2-1), \min(b_1, b_2)]$. This means that if $a = 0$, $\max(0, a_1 + a_2 - 1) = 0$, i.e., $a_1 + a_2 \leq 1$ and if $a > 0$ then $a = a_1 + a_2 - 1$. Similarly, $b = \min(b_1, b_2)$ implies, $b = b_1$ when $b_1 \leq b_2$ and $b = b_2$ when $b_2 \leq b_1$.

This means that by definition of d_{igc} $\langle [a_1, b_1], [a_2, b_2] \rangle \in d_{igc}([a, b])$.

- pcc is a conjunctive coherent p-strategy.

  - pcc is a **conjunctive** p-strategy.

    Let us establish that pcc satisfies the axioms of conjunctive p-strategy.

    1. Commutativity.

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

    2. Associativity.

       $c_{pcc}(c_{pcc}([a_1, b_1], [a_2, b_2]), [a_3, b_3]) = c_{pcc}([\min(a_1, a_2), \min(b_1, b_2)]$
       $, [a_3, b_3]) = [\min(\min(a_1, a_2), a_3), \min(\min(b_1, b_2), b_3)] =$
       $[\min(a_1, a_2, a_3), (\min(b_1, b_2, b_3)] = [\min(a_1, \min(a_2, a_3)),$
       $\min(b_1, (\min(b_2, b_3))] = c_{pcc}([a_1, b_1], c_{pcc}([a_2, b_2], [a_3, b_3]))$.

3. Inclusion Monotonicity.

   Let $[a_1, b_1] \subseteq [a_3, b_3]$, i.e. $a_1 \geq a_3 \geq 0$ and $0 \leq b_1 \leq b_3$.

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

   $c_{pcc}([a_3, b_3], [a_2, b_2]) = [\min(a_3, a_3), \min(b_1, b_2)]$.

   Since $a_1 \geq a_3 \geq 0$, $\min(a_1, a_2) \geq \min(a_3, a_3)$; since $0 \leq b_1 \leq b_3$, $\min(b_1, b_2)] \leq \min(b_1, b_2)]$. This implies $[\min(a_1, a_2), \min(b_1, b_2)] \subseteq [\min(a_3, a_3), \min(b_1, b_2)]$.

4. Bottomline.

   $c_{pcc}([a_1, b_1], [a_2, b_2]) = [\min(a_1, a_2), \min(b_1, b_2)] \leq_t$
   $[\min(a_1, a_2), \min(b_1, b_2)]$ (since $\leq_t$ is reflexive).

5. Identity.

   $c_{pcc}([a, b], [1, 1]) = [\min(a, 1), \min(b, 1)] = [a, b]$.

6. Annihilator.

   $c_{pcc}([a, b], [0, 0]) = [\min(a, 0), \min(b, 0)] = [0, 0]$.

– pcc is a **coherent** p-strategy.

  Let $[a, b], [a_1, b_1], [a_2, b_2] \in \mathcal{C}[0, 1]$.

  * Let $\langle[a_1, b_1], [a_2, b_2]\rangle \in d_{pcc}([a, b])$. Then either $a = a_1$ and $a_2 \geq a_1$ or $a = a_2$ and $a_1 \geq a_2$. In either case $a = \min(a_1, a_2)$.

    Similarly, since either $b = b_1$ and $b_2 \geq b_1$ or $b = b_2$ and $b_1 \geq b_2$, we get $b = \min(b_1, b_2)$.

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

  * Let $c_{pcc}([a_1, b_1], [a_2, b_2]) = [a, b]$. Then $a = \min(a_1, a_2)$ and $b = \min(b_1, b_2)$. This means that either $a = a_1$ and $a_2 \geq a_1$ or $a = a_2$ and $a_1 \geq a_2$ and similarly, either $b = b_1$ and $b_2 \geq b_1$ or $b = b_2$ and $b_1 \geq b_2$. But this means that $\langle[a_1, b_1], [a_2, b_2]\rangle \in d_{pcc}([a, b])$.

- ind is a disjunctive coherent p-strategy.
  - ind is a **disjunctive** p-strategy.

    Let us establish that ind satisfies the axioms of disjunctive p-strategy.

    1. Commutativity.

       $c_{ind}([a_1, b_1], [a_2, b_2]) = [a_1 + a_2 - a_1a_2, b_1 + b_2 - b_1b_2] = [a_2 + a_1 - a_2a_1, b_2 + b_1 - b_2b_1] = c_{ind}([a_2, b_2], [a_1, b_1])$.

    2. Associativity.

       $c_{ind}(c_{ind}([a_1, b_1], [a_2, b_2]), [a_3, b_3]) = c_{ind}([a_1 + a_2 - a_1a_2, b_1 + b_2 - b_1b_2], [a_3, b_3]) = [a_1 + a_2 - a_1a_2 + a_3 - (a_1 + a_2 - a_1a_2)a_3, b_1 + b_2 - b_1b_2 + b_3 - (b_1 + b_2 - b_1b_2)b_3] = [a_1 + a_2 + a_3 - a_1a_2 - a_2a_3 - a_1a_3 + a_1a_2a_3, b_1 + b_2 + b_3 - b_1b_2 - b_2b_3 - b_1b_3 + b_1b_2b_3] = [a_1 + (a_2 + a_3 - a_2a_3) - a_1(a_2 + a_3 - a_2a_3), b_1 + (b_2 + b_3 - b_2b_3) - b_1(b_2 + b_3 - b_2b_3)] = c_{inc}([a_1, b_2], c_{inc}([a_2, b_2], [a_3, b_3]))$

    3. Inclusion Monotonicity.

       Let $[a_1, b_1] \subseteq [a_3, b_3]$, i.e. $a_1 \geq a_3 \geq 0$ and $0 \leq b_1 \leq b_3$.

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

       $c_{ind}([a_3, b_3], [a_2, b_2]) = [a_3 + a_2 - a_3a_2, b_3 + b_2 - b_3b_2]$.

       Since $a_1 \geq a_3 \geq 0$, we have $a_1 - a_1a_2 = a_1(1 - a_2) \geq a_3(1 - a_2) = a_3 - a_3a_2$

       and therefore $a_1 + a_2 - a_1a_2 \geq a_3 + a_2 - a_3a_2$.

       Similarly, since $0 \leq b_1 \leq b_3$, we have $b_1 - b_1b_2 = b_1(1 - b_2) \leq b_3(1 - b_3) = b_3 - b_3b_2$

       which in turn implies that $b_1 + b_2 - b_1b_2 \leq b_3 + b_2 - b_3b_2$.

       From this it follows that $[a_1 + a_2 - a_1a_2, b_1 + b_2 - b_1b_2] \subseteq [a_3 + a_2 - a_3a_2, b_3 + b_2 - b_3b_2]$.

4. Bottomline.

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

   We have to show that $a_1 + a_2 - a_1 a_2 \geq \max(a_1, a_2)$ and $b_1 + b_2 - b_1 b_2 \geq \max(b_1, b_2)$.

   Indeed, since $0 \leq a_1, a_2 \leq 1$, $a_2 - a_1 a_2 \geq 0$ and $a_1 - a_1 a_2 \geq 0$. Therefore, $a_1 + (a_2 - a_1 a_2) \geq a_1 + 0$ and $a_2 + (a_1 - a_1 a_2) \geq a_2 + 0$, i.e. $a_1 + a_2 - a_1 a_2 \geq \max(a_1, a_2)$.

   Similarly, since $0 \leq b_1, b_2 \leq 1$, $b_2 - b_1 b_2 \geq 0$ and $b_1 - b_1 b_2 \geq 0$. Therefore, $b_1 + (b_2 - b_1 b_2) \geq b_1 + 0$ and $b_2 + (b_1 - b_1 b_2) \geq b_2 + 0$, i.e. $b_1 + b_2 - b_1 b_2 \geq \max(b_1, b_2)$.

5. Identity.

   $c_{ind}([a, b], [0, 0]) = [a + 0 - a * 0, b + 0 - b * 0] = [a, b]$.

6. Annihilator.

   $c_{ind}([a, b], [1, 1]) = [a + 1 - a * 1, b + 1 - b * 1] = [1, 1]$.

– ind is a **coherent** p-strategy.

Let $[a, b], [a_1, b_1], [a_2, b_2] \in \mathcal{C}[0, 1]$.

* Let $\langle [a_1, b_1], [a_2, b_2] \rangle \in d_{ind}([a, b])$. Then $a_1 + a_2 - a_1 a_2 = a$ and $b_1 + b_2 - b_1 b_2 = b$, which means that $c_{ind}([a_1, a_2], [b_1, b_2]) = [a_1 + a_2 - a_1 a_2, b_1 + b_2 - b_1 b_2] = [a, b]$.

* Let $c_{ind}([a_1, b_1], [a_2, b_2]) = [a, b]$. $c_{ind}([a_1, b_1], [a_2, b_2]) = [a_1 + a_2 - a_1 a_2, b_1 + b_2 - b_1 b_2]$, which means that $a = a_1 + a_2 - a_1 a_2$ and $b = b_1 + b_2 - b_1 b_2$. Therefore,$\langle [a_1, b_1], [a_2, b_2] \rangle \in d_{ind}([a, b])$.

• igd is a disjunctive coherent p-strategy.

– igd is a **disjunctive** p-strategy.

Let us establish that igd satisfies the axioms of disjunctive p-strategy.

1. Commutativity.

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

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

2. Associativity.

   $c_{igd}(c_{igd}([a_1, b_1], [a_2, b_2]), [a_3, b_3]) = c_{igd}([\min(a_1, a_2),$

   $\max(1, b_1 + b_2)], [a_3, b_3]) = [\min(\min(a_1, a_2), a_3), \max(1, \max(1, b_1 +$

   $b_2) + b_3)] = [\min(a_1, a_2, a_3), \max(1, b_1 + b_2 + b_3)] =$

   $[\min(a_1, \min(a_2, a_3), \max(1, b_1 + \max(1, b_2 + b_3))] =$

   $c_{igd}([a_1, b_1], c_{igd}([a_2, b_2], [a_3, b_3])$.

3. Inclusion Monotonicity.

   Let $[a_1, b_1] \subseteq [a_3, b_3]$, i.e. $a_1 \geq a_3 \geq 0$ and $0 \leq b_1 \leq b_3$.

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

   $c_{igd}([a_3, b_3], [a_2, b_2]) = [\max(a_3, a_2), \min(1, b_3 + b_2)]$.

   Since $a_1 \geq a_3 \geq 0$ we have $\max(a_1, a_2) \geq \max(a_3, a_2)$. Since $0 \leq b_1 \leq b_3$

   we have $\min(1, b_1 + b_2) \leq \min(1, b_3, b_2)$.

   This implies $[\max(a_1, a_2), \min(1, b_1 + b_2)] \subseteq [\max(a_3, a_2), \min(1, b_3 + b_2)]$.

4. Bottomline.

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

   We have to show that $\min(1, b_1 + b_2) \geq \max(b_1, b_2)$. This is clearly so, since, $b_1, b_2 \leq 1$, i.e. $\max(b_1, b_2) \leq 1$, and $b_1, b_2 \geq 0$, i.e., $b_1 \leq b_1 + b_2$ and $b_2 \leq b_1 + b_2$, which makes $\max(b_1, b_2) \leq b_1 + b_2$, therefore yielding the desired result.

5. Identity.

   $c_{igd}([a,b],[0,0]) = [\max(a,0), \min(1,b)] = [a,b]$.

6. Annihilator.

   $c_{igd}([a,b],[1,1]) = [\max(a,1), \min(1,b+1)] = [1,1]$.

– igd is a **coherent** p-strategy.

  Let $[a,b],[a_1,b_1],[a_2,b_2] \in \mathcal{C}[0,1]$.

  * Let $\langle[a_1,b_1],[a_2,b_2]\rangle \in d_{igc}([a,b])$.

    Then either $a = a_1$ and $a_2 \leq a_1$ or $a = a_2$ and $a_1 \leq a_2$. In either case $a = \max(a_1,a_2)$.

    Also, either we have $b = 1$ and $b_1 + b_2 \geq 1$ or $b < 1$ and $b_1 + b_2 = b$. In either case $b = \min(1, b_1 + b_2)$.

    But then, we get $c_{igc}([a_1,b_1],[a_2,b_2]) = [\max(a_1,a_2), \min(1, b_1+b_2)] = [a,b]$.

  * Let $c_{igc}([a_1,b_1],[a_2,b_2]) = [a,b]$.

    In this case $a = \max(a_1,a_2)]$ and $b = \min(1, b_1 + b_2)$. This means that either $a = a_1$ and $a_2 \leq a_1$ or $a = a_2$ and $a_1 \leq a_2$ and also either $b = 1$ and $b_1 + b_2 \geq 1$ or $b < 1$ and $b_1 + b_2 = b$.

    But then, $\langle[a_1,b_1],[a_2,b_2]\rangle \in d_{igc}([a,b])$.

• pcd is a disjunctive coherent p-strategy.

– pcd is a **disjunctive** p-strategy.

  Let us establish that pcd satisfies the axioms of disjunctive p-strategy.

1. Commutativity.

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

2. Associativity.

$c_{pcd}(c_{pcd}([a_1, b_1], [a_2, b_2]), [a_3, b_3]) = c_{pcd}([\max(a_1, a_2), \max(b_1, b_2)],$

$[a_3, b_3]) = [\max(\max(a_1, a_2), a_3), \max(\max(b_1, b_2), b_3)] =$

$[\max(a_1, a_2, a_3), \max(b_1, b_2, b_3)] = [\max(a_1, \max(a_2, a_3)),$

$\max(b_1, \max(b_2, b_3))] = c_{pcd}([a_1, b_1], c_{pcd}([a_2, b_2], [a_3, b_3])).$

3. Inclusion Monotonicity.

Let $[a_1, b_1] \subseteq [a_3, b_3]$, i.e. $a_1 \geq a_3 \geq 0$ and $0 \leq b_1 \leq b_3$.

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

$c_{pcd}([a_3, b_3], [a_2, b_2]) = [\max(a_3, a_2), \max(b_3, b_2)].$

Since $a_1 \geq a_3 \geq 0$ we have $\max(a_1, a_2) \geq \max(a_3, a_2)$. Similarly, since $0 \leq b_1 \leq b_3$, we have $\max(b_1, b_2) \leq \max(b_3, b_2)$.

This implies $[\max(a_1, a_2), \max(b_1, b_2)] \subseteq [\max(a_3, a_2), \max(b_3, b_2)]$.

4. Bottomline.

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

$[\max(a_1, a_2), \max(b_1, b_2)].$

5. Identity.

$c_{pcd}([a, b], [0, 0]) = [\max(a, 0), \max(b, 0)] = [a, b].$

6. Annihilator.

$c_{pcd}([a, b], [1, 1]) = [\max(a, 1), \max(b, 1)] = [a, a].$

– pcd is a **coherent** p-strategy.

Let $[a, b], [a_1, b_1], [a_2, b_2] \in \mathcal{C}[0, 1]$.

* Let $\langle [a_1, b_1], [a_2, b_2] \rangle \in d_{pcd}([a, b])$. Then either $a = a_1$ and $a_1 \geq a_2$ or $a = a_2$ and $a_2 \geq a_1$. In either case $a = \max(a_1, a_2)$. Similarly, we have either $b = b_1$ and $b_1 \geq b_2$ or $b = b_2$ and $b_2 \geq b_1$. In either case $b = \max(b_1, b_2)$.

Therefore $c_{pcd}([a_1,b_1],[a_2,b_2]) = [\max(a_1,a_2), \max(b_1,b_2)] = [a,b]$.

* Let $c_{pcd}([a_1,b_1],[a_2,b_2]) = [a,b]$. In this case $a = \max(a_1,a_2)$ and $b = \max(b_1,b_2)$. This means that either $a = a_1$ and $a_1 \geq a_2$ or $a = a_2$ and $a_2 \geq a_1$ and also either $b = b_1$ and $b_1 \geq b_2$ or $b = b_2$ and $b_2 \geq b_1$. Therefore, $\langle [a_1,b_1],[a_2,b_2] \rangle \in d_{pcd}([a,b])$.

- ncd is a disjunctive coherent p-strategy.

  - ncd is a **disjunctive** p-strategy.

    Let us establish that ncd satisfies the axioms of disjunctive p-strategy.

    1. Commutativity.

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

    2. Associativity.

       $c_{ncd}(c_{ncd}([a_1,b_1],[a_2,b_2]),[a_3,b_3]) = c_{ncd}([\min(1,a_1+a_2), \min(1,b_1+b_2)],[a_3,b_3]) = [\min(1,\min(1,a_1+a_2)+a_3), \min(1,\min(1,b_1+b_2)+b_3)] = [\min(1,a_1+a_2+a_3), \min(1,b_1+b_2+b_3)] = [\min(1,a_1+\min(1,a_2+a_3)), \min(1,b_1+\min(1,b_2+b_3))] =$
       $c_{ncd}([a_1,b_1],c_{ncd}([a_2,b_2],[a_3,b_3]))$.

    3. Inclusion Monotonicity.

       Let $[a_1,b_1] \subseteq [a_3,b_3]$, i.e. $a_1 \geq a_3 \geq 0$ and $0 \leq b_1 \leq b_3$.
       $c_{ncd}([a_1,b_1],[a_2,b_2]) = [\min(1,a_1+a_2), \min(1,b_1+b_2)]$.
       $c_{ncd}([a_3,b_1],[a_3,b_2]) = [\min(1,a_3+a_2), \min(1,b_3+b_2)]$.
       Since $a_1 \geq a_3 \geq 0$ we have $a_1+a_2 \geq a_3+a_2$ and therefore,
       $\min(1,a_1+a_2) \geq \min(1,a_3+a_2)$. Similarly, since $0 \leq b_1 \leq b_3$,
       we have $\min(1,b_1+b_2) \leq \min(1,b_3+b_2)$.

This implies that $[\min(1, a_1 + a_2), \min(1, b_1 + b_2)]$

$\subseteq [\min(1, a_3 + a_2), \min(1, b_3 + b_2)]$.

4. Bottomline.

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

   We need to show $\min(1, a_1 + a_2) \geq \max(a_1, a_2)$ and $\min(1, b_1 + b_2) \geq \max(b_1, b_2)$.

   Since, $a_1, a_2 \leq 1$, we can get $\max(a_1, a_2) \leq 1$, and since $a_1, a_2 \geq 0$, we have $a_1 \leq a_1+a_2$ and $a_2 \leq a_1+a_2$, which makes $\max(a_1, a_2) \leq a_1+a_2$. Similarly, since, $b_1, b_2 \leq 1$, we can get $\max(b_1, b_2) \leq 1$, and since $b_1, b_2 \geq 0$, we have $b_1 \leq b_1 + b_2$ and $b_2 \leq b_1 + b_2$, which makes $\max(b_1, b_2) \leq b_1 + b_2$, therefore yielding the desired result.

5. Identity.

   $c_{ncd}([a, b], [0, 0]) = [\max(a, 0), \max(b, 0)] = [a, b]$.

6. Annihilator.

   $c_{ncd}([a, b], [1, 1]) = [\max(a, 1), \max(b, 1)] = [1, 1]$.

– ncd is a **coherent** p-strategy.

Let $[a, b], [a_1, b_1], [a_2, b_2] \in \mathcal{C}[0, 1]$.

* Let $\langle [a_1, b_1], [a_2, b_2] \rangle \in d_{ncd}([a, b])$. Then we know that either $a = 1$ and $a_1 + a_2 \geq 1$ or $a < 1$ and $a_1 + a_2 = a$. This implies $a = \min(1, a_1 + a_2)$. Similarly, either $b = 1$ and $b_1 + b_2 \geq 1$ or $b < 1$ and $b_1 + b_2 = b$, which, in turn implies $b = \min(1, b_1 + b_2)$.

  From this it follows that $c_{ncd}([a_1, b_1], [a_2, b_2]) = [\min(1, a_1 + a_2), \min(1, b_1 + b_2)] = [a, b]$.

* Let $c_{ncd}([a_1, b_1], [a_2, b_2]) = [a, b]$. Then $a = \min(1, a_1 + a_2)$ and $b = \min(1, b_1 + b_2)$. This means that if $a = 1$ then $a_1 + a_2 \geq 1$ and if $a < 1$

then $a_1 + a_2 = a$. Similarly, we get : if $b = 1$ then $b_1 + b_2 \geq 1$ and if $b < 1$ then $b_1 + b_2 = b$.

From this we infer $\langle [a_1, b_1], [a_2, b_2] \rangle \in d_{ncd}([a, b])$.

3.2.3 Properties of P-Strategies

In this section, we define various aspects of p-strategies that will play a key role in the definition of our fixpoint semantics and our model theory for hybrid probabilistic logic programs. First, we need the following very simple property.

Lemma 1 *Let $\rho = \langle c, d \rangle$ be a coherent p-strategy. Then a pair $\langle [a_1, b_1], [a_2, b_2] \rangle \in d([a, b])$* **iff** *$\langle [a_2, b_2], [a_1, b_1] \rangle \in d([a, b])$*

Proof. By commutativity of composition functions if $c([a_1, b_1], [a_2, b_2]) = [a, b]$ then $c([a_2, b_2], [a_1, b_1]) = [a, b]$. Since ρ is a coherent p-strategy, both $\langle [a_1, b_1], [a_2, b_2] \rangle$ and $\langle [a_2, b_2], [a_1, b_1] \rangle$ are in $d[a, b]$. $\square$

The simple lemma above merely assures us that if $\langle [a_1, b_1], [a_2, b_2] \rangle \in d([a, b])$, then so is $\langle [a_2, b_2], [a_1, b_1] \rangle$. Lemma 2 establishes an important and expected property of p-strategies, namely that if lower bounds for the arguments of any composition function are both equal to 0 then so will be the resulting lower bound, and similarly, if two input upper bounds are both 1, then the resulting upper bound is also 1. This matches our general intuition about the probabilities of complex events. Indeed, if it is possible that the probabilities of both e_1 and e_2 are 0, we should deem it possible that the probabilities of $e_1 \vee e_2$ and $e_1 \wedge e_2$ aslo can be 0. Similar reasoning works for upper bounds as well.

Lemma 2 *Let $\rho = \langle c_\rho, d_\rho \rangle$ be a coherent disjunctive or conjunctive p-strategy. Then:*

1. $c_\rho([0, 1], [0, 1]) = [0, 1]$.

2. *More generally,* $(\forall x, y \in [0,1])(\exists z \in [0,1])(c_\rho([x,1],[y,1]) = [z,1]$ *and* $(\forall x, y \in [0,1])(\exists z \in [0,1])c_\rho([0,x],[0,y]) = [0,z])$.

Proof.

- c_ρ is a conjunctive p-strategy.

  1. $(\forall x, y \in [0,1])(\exists z \in [0,1])(c_\rho([x,1],[y,1]) = [z,1])$.

     We know that by the axiom of Identity for conjunctive p-strategies, $c_\rho([x,1],[1,1]) = [x,1]$. Since $y \in [0,1], [1,1] \subseteq [y,1]$. Therefore, by the axiom of Monotonicity we get $c_\rho([x,1],[1,1]) = [x,1] \subseteq c_\rho([x,1],[y,1]) \subseteq [0,1]$.

     From this it is clear that the upper bound of the interval for $c_\rho([x,1],[y,1])$ will be 1, which means that $c_\rho([x,1],[y,1]) = [z,1])$ for some $z \in [0,1]$.

  2. $(\forall x, y \in [0,1])(\exists z \in [0,1])(c_\rho([0,x],[0,y]) = [0,z])$.

     We know that be the axiom of Annihilator for conjunctive p-strategies, $c_\rho([0,x],[0,0]) = [0,0]$. Since $y \in [0,1], [0,0] \subseteq [0,y]$. Therefore, by the axiom of Monotonicity we get $c_\rho([0,x],[0,0]) = [0,0] \subseteq c_\rho([0,x],[0,y]) \subseteq [0,1]$.

     From this it is clear that the lower bound of the interval for $c_\rho([0,x],[0,y])$ will be 0, which means that $c_\rho([0,x],[0,y]) = [0,z])$ for some $z \in [0,1]$.

- c_ρ is a disjunctive p-strategy.

  1. $(\forall x, y \in [0,1])(\exists z \in [0,1])(c_\rho([x,1],[y,1]) = [z,1])$.

     We know that be the axiom of Annihilator for disjunctive p-strategies, $c_\rho([x,1],[1,1]) = [1,1]$. Since $y \in [0,1], [1,1] \subseteq [y,1]$. Therefore, by the axiom of Monotonicity we get $c_\rho([x,1],[1,1]) = [1,1] \subseteq c_\rho([x,1],[y,1]) \subseteq [0,1]$.

From this it is clear that the upper bound of the interval for $c_\rho([x,1],[y,1])$ will be 1, which means that $c_\rho([x,1],[y,1]) = [z,1])$ for some $z \in [0,1]$.

2. $(\forall x, y \in [0,1])(\exists z \in [0,1])(c_\rho([0,x],[0,y]) = [0,z])$.

   We know that be the axiom of Identity for disjunctive p-strategies, $c_\rho([0,x],[0,0]) = [0,x]$. Since $y \in [0,1], [0,0] \subseteq [0,y]$. Therefore, by the axiom of Monotonicity we get $c_\rho([0,x],[0,0]) = [0,x] \subseteq c_\rho([0,x],[0,y]) \subseteq [0,1]$.

   From this it is clear that the lower bound of the interval for $c_\rho([0,x],[0,y])$ will be 0, which means that $c_\rho([0,x],[0,y]) = [0,z])$ for some $z \in [0,1]$. $\square$

It is easy to see, that a composition function performs its duty exactly as planned. Given two intervals μ_1 and μ_2, it returns the interval $\mu = c(\mu_1, \mu_2)$, which (as 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, a 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 simple events based on the entire content of the result of decomposition. The following definitions will help us achieve this goal.

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

Definition 3.2.4 *Let $\rho =< c, d >$. The decomposition projection set πD is:*

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

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

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

Definition 3.2.5 *Let $\rho =< c, d >$ be a p-strategy.*
The maximal interval md for $d([a, b])$ is :

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

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

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

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

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

Proof.

1. Let $md_\rho([a, b]) = [a', b']$. Since ρ is conjunctive strategy, $c_\rho([a, b], [1, 1]) = [a, b]$ (Identity), and since ρ is coherent, $[1, 1] \in \pi D_\rho([a, b])$.

   Since $b' = \max_{[\hat{a},\hat{b}]\in\pi D_\rho([a,b])}(\hat{b})$, and $[1, 1] \in \pi D_\rho([a, b])$, $b' = 1$.

   Since $c_\rho([a, b], [1, 1]) = [a, b]$ and ρ is coherent, $[a, b] \in \pi D_\rho([a, b])$. By the bottomline axiom, $(\forall [\hat{a}, \hat{b}] \in \pi D_\rho([a, b]))(a \leq \hat{a})$. Since $[a, b] \in \pi D_\rho([a, b])$ $a = \min_{[\hat{a},\hat{b}]\in\pi D_\rho([a,b])}(\hat{a})$, and therefore, $a' = a$.

2. Let $md_{\rho'}([a, b]) = [a', b']$. Since ρ' is disjunctive strategy, $c_{\rho'}([a, b], [0, 0]) = [a, b]$ (Identity), and since ρ' is coherent, $[0, 0] \in \pi D_{\rho'}([a, b])$. Therefore, since $a' = \min_{[\hat{a},\hat{b}]\in\pi D_{\rho'}([a,b])}(\hat{a})$, and $[0, 0] \in \pi D_{\rho'}([a, b])$, $a' = 0$.

   Since $c_{\rho'}([a, b], [0, 0]) = [a, b]$ and ρ' is coherent, $[a, b] \in \pi D_{\rho'}([a, b])$. By the bottomline axiom, $(\forall [\hat{a}, \hat{b}] \in \pi D_{\rho'}([a, b]))(b \geq \hat{b})$. Since $[a, b] \in \pi D_{\rho'}([a, b])$ $b = \max_{[\hat{a},\hat{b}]\in\pi D_{\rho'}([a,b])}(\hat{b})$, and therefore, $b' = b$. □

3.2.4 Other p-strategies

A natural question that the reader may ask is what p-strategies exist in addition to those that we have presented above. We present a couple of other example p-strategies below that are hybrids of the ones presented earlier, and then we have a technical discussion about how other p-strategies may be constructed. For the sake of simplicity, we will only discuss composition functions, because decomposition functions can be derived from composition functions using the definition of coherence.

Example 3.2.1 (Mixed-Ignorance-Independence Strategies) Consider a situation where a user considers two events e_1, e_2 whose probabilities are known to be in the ranges $[a_1, b_1], [a_2, b_2]$ respectively. The user in question is not sure if e_1, e_2 are independent, but thinks they might be. As a consequence, he wants the resulting range to be obtained by tightening the range that the ignorance strategy would have returned, by taking his feeling that events e_1, e_2 may be independent into account. He could do this in many ways.

Pessimistic Mixed Strategy: The user may define a pessimistic conjunction strategy c_{pes} such that $c_{pes}([a_1, b_1], [a_2, b_2])$ is computed as follows.

1. Compute $[a, b] = c_{ind}([a_1, b_1], [a_2, b_2])$ and $[a', b'] = c_{igc}([a_1, b_1], [a_2, b_2])$.
2. Let $c_{pes}([a_1, b_1], [a_2, b_2]) = [\min(a, a'), \min(b, b')]$.

This function can be verified to be a conjunctive p-strategy by modifying the proof of Proposition 1 in a straightforward way. In fact, $c_{pes}([a_1, b_1], [a_2, b_2])$ can be directly computed to be $[\max(0, a_1 + a_2 - 1), b_1 b_2]$.

The intuition is that the user is not sure whether the events e_1, e_2 whose conjunction is being considered are independent or not. He chooses to be cautious, and decides to use the smallest values returned for the lower and upper bounds. This approach may be justified in applications where we need to be biased towards assuming lower probabilities of events.

Optimistic Mixed Strategy: On the other hand, the user may tend to assume higher probabilities for complex events, e.g. in the cases of failures between components of a physical system where independence is suspected, but not known, a user a may choose to believe higher probabilities of failure. In a sense, the user is optimistic that the probability that the complex event happens is larger than the pessimistic approach above might suggest. Here, he may use the following optimistic conjunction strategy $c_{opt}([a_1, b_1], [a_2, b_2])$:

1. Compute $[a, b] = c_{ind}([a_1, b_1], [a_2, b_2])$ and $[a', b'] = c_{igc}([a_1, b_1], [a_2, b_2])$.
2. Let $c_{opt}([a_1, b_1], [a_2, b_2]) = [\max(a, a'), \max(b, b')]$.

Here too, $c_{opt}([a_1, b_1], [a_2, b_2])$ can be directly computed to be $[a_1 a_2, \min(b_1, b_2)]$.

Example 3.2.2 (Generalized Mixed Strategies) The reader may have already noted that the "code" given above to merge independence and ignorance according to pessimistic or optimistic approaches can be generalized to merge arbitrary p-strategies, both for conjunction and disjunction. For example, suppose we have two conjunctive p-strategies ρ_1, ρ_2. If

$$\begin{aligned} c_{\rho_1}([a_1, b_1], [a_2, b_2]) &= [a, b] \\ c_{\rho_2}([a_1, b_1], [a_2, b_2]) &= [a', b'] \end{aligned}$$

then we may define a pessimistic mix, $c_{pes}^{\rho_1,\rho_2}$, and an optimistic mix, $c_{opt}^{\rho_1,\rho_2}$, as follows:

$$\begin{aligned} c_{pes}^{\rho_1,\rho_2}([a_1, b_1], [a_2, b_2]) &= [\min(a, a'), \min(b, b')]. \\ c_{opt}^{\rho_1,\rho_2}([a_1, b_1], [a_2, b_2]) &= [\max(a, a'), \max(b, b')]. \end{aligned}$$

In fact, it is easy to verify that for all $[a_1, b_1], [a_2, b_2]$,

$$c_{pes}^{\rho_1,\rho_2}([a_1, b_1], [a_2, b_2]) \leq_t c_{opt}^{\rho_1,\rho_2}([a_1, b_1], [a_2, b_2]).$$

Thus, the pessimistic mix of two p-strategies always tends to produce lower probabilities than the optimistic mix, thus justifying their names.

In addition to the above mixing strategies that allow us to define a set of new p-strategies, we provide below, some general guidance on how yet other p-strategies may be constructed.

Suppose χ is an associative and commutative function (of which there are many!) which takes two intervals $[a_1, b_1], [a_2, b_2]$ as input, and produces an output interval $[a, b]$. For χ to be the composition part of a conjunctive p-strategy. χ must satisfy the Bottom Line and Inclusion Monotonicity axioms as well as the Annihilator and Identity axioms. Out of these four, Bottom Line and Inclusion Monotonicity jointly impose very strong restrictions on which χ's can be used in p-strategies.

Suppose $[a_1, b_1] \subseteq [a_3, b_3]$, and $[a_2, b_2]$ is any subinterval of $[0, 1]$.
Let $[a, b] = \chi([a_1, b_1], [a_2, b_2])$ and $[a', b'] = \chi([a_3, b_3], [a_2, b_2])$. It is easy to see that:

$$\begin{aligned} a_3 &\leq a_1 \\ b_1 &\leq b_3 \\ a' &\leq a \\ b &\leq b' \\ a &\leq a_1 \\ b &\leq b_1 \\ a' &\leq a_3 \\ b' &\leq b_3 \end{aligned}$$

The first two conditions follow as $[a_1, b_1] \subseteq [a_3, b_3]$. The third and fourth inequalities follow because Inclusion Monotonicity tells us that $[a, b] \subseteq [a', b']$. The last four inequalities follow from the Bottom Line axiom which tells us that $[a, b] \leq_t [a_1, b_1]$ and $[a', b'] \leq_t [a_3, b_3]$. Figure 3.1 shows the relationships between these values diagrammatically. An edge from x to x' means x is less than or equal to x'.

In addition, the axiom of identity says that when $[a_2, b_2] = [1, 1]$ then $\chi([a_1, b_1], [a_2, b_2]) = [a_1, b_1]$ and $\chi([a_3, b_3], [a_2, b_2]) = [a_3, b_3]$. Diagrammatically, this means that the top row and the bottom row in Figure 3.1 must coincide when $[a_2, b_2] = [1, 1]$. Similarly, when $[a_2, b_2] = [0, 0]$, the bottom row collapses to one point, viz. 0 because a', a, b, b' are all set to 0 by the Annihilator Axiom.

Thus, to define a function χ in such a way that it is the composition function of a p-strategy, the reader must ensure that the diagram associated with χ looks like that shown in Figure 3.1, and exhibits the extremal properties mentioned in the previous paragraph when $[a_2, b_2] = [1, 1]$ or $[a_2, b_2] = [0, 0]$.

$$\begin{tikzcd}
a_3 \arrow[r] & a_1 \arrow[r] & b_1 \arrow[r] & b_3 \\
a' \arrow[r] \arrow[u] & a \arrow[r] \arrow[u] & b \arrow[r] \arrow[u] & b' \arrow[u]
\end{tikzcd}$$

Figure 3.1: Inequalities induced by Bottomline and Inclusion Monotonicity Axioms

Part II

Hybrid Probabilistic Logic Programs

As mentioned in Section 2.2, 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 cases where the actual relationship between the events is known, or in cases where we are willing to make certain assumptions about this relationship Probabilistic Logic Programs are not as helpful as they could be.

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 goals were:

- 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 rules to have non-atomic heads.
- 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 HPLPs. The first query resolution procedure constructed was provably sound and complete, but 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 results of such subqueries in a table and consulting the table before proceeding with the resolution of

the next sub query 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.

We now define the syntax and semantics of Hybrid Probabilistic Logic Programs. These definitions first appeared in [22].

Chapter 4

Hybrid Probabilistic Programs

4.1 Syntax of Hybrid Probabilistic Programs

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 terms are defined 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 $\vee_{\rho'}$.

Definition 4.1.1 *Let $\mathcal{CONJ}$ be a finite set of conjunctive p-strategies and $\mathcal{DISJ}$ be a finite set of disjunctive p-strategies. Let $\mathcal{S}$ denote $\mathcal{CONJ} \cup \mathcal{DISJ}$.*

- Let $\rho \in \mathcal{CONJ}$. *Connective $\wedge_\rho$ is called a* ρ-annotated conjunction.
- Let $\rho \in \mathcal{DISJ}$. *Connective $\vee_\rho$ is called a* ρ-annotated disjunction.

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

Definition 4.1.2 *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.

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

For instance, the formulas (a $\vee_{igd}$ b) and (a $\wedge_{inc}$ c) are basic formulas involving the ignorance and independence p-strategies. However, (a $\wedge_{inc}$ b) $\wedge_{ind}$ c is not a basic formula, as it involves two different p-strategies. In order to proceed further we have to define the notion of *annotation*. Definitions 4.1.3 – 4.1.5 below were first introduced in [83].

Definition 4.1.3 *An annotation function f of arity n is a total function*

$$f : [0, 1]^n \longrightarrow [0, 1].$$

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

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 [1].

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

[1]We do not formally define computable functions over the real numbers because the theory of computability over real numbers is now well understood [11] and the reader may refer to such treatments for a detailed technical analysis of this issue.

Definition 4.1.4 *An annotation item δ is one of the following:*

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

Definition 4.1.5 *Let δ_1 and δ_2 be annotation items. Then $[\delta_1, \delta_2]$ is called an annotation or an annotation term.*

When δ_1, δ_2 are both constants, then the annotation term $[\delta_1, \delta_2]$ denotes an interval. Otherwise, it denotes a *set* of intervals, obtained by instantiating δ_1, δ_2 in different ways. Following the terminology introduced in [83], if an annotation term has no annotation variables in it, we call it a *c-annotation*. Otherwise it will be called a *v-annotation*.

Example 4.1.1 $[0, 1]$ and $[0.3, 0.6]$ are c-annotations. $[V_1, 1]$ and $[0.5 * V_1, V_1]$ are v-annotations.

Definition 4.1.6 *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.*

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

In this work, hybrid probabilistic annotated basic formulas are the basic syntactic objects that merge together, probabilistic reasoning and logical reasoning. For example, if $(a \wedge_{ig} b) : [0.5, 0.7]$ is a hybrid probabilistic annotated basic formula, this formula says that "If we assume that we have no knowledge of the dependencies or lack thereof between events a and b, then the probability that both events a and b occur lies between 0.5 and 0.7 inclusive." In general, the hybrid probabilistic annotated basic formula $(a \wedge_\rho b) : [0.5, 0.7]$ says that "If we assume that our knowledge of the dependency between a and b is given by the probabilistic-strategy ρ, then the probability that both events a and b occur lies between 0.5 and 0.7 inclusive." Similar rationales can be given for disjunctive basic formulas.

Hybrid rules may now be constructed from hybrid annotated formulas as follows.

Definition 4.1.7 *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 an expression 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 ... the probability of B_k falls within the interval μ_k, then the probability of B_0 falls within the interval μ_0".

Intuitively a basic formula is a statement about probabilities of events. The conjunction in the body of an hp-clause, on the other hand defines a conjunction of such statements, but does not itself represent an event.

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

Example 4.1.2

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

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

Definition 4.1.8 *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 every clause in it is ground, i.e. all its clauses contain neither variables nor variable annotations. For example, the following clauses constitute a simple hp-program using the p-strategies of ignorance and independence.

Example 4.1.3 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 non-stability. 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 $1 - V1$.

Example 4.1.4 Let us consider a rule of the form

$$c : \mu \longleftarrow (a\ \wedge_{inc}\ b) : \mu_1\ \wedge\ (a\ \wedge_{pcc}\ b) : \mu_2.$$

A rule of this sort may be read as "If $(a \wedge b)$'s probability lies in the interval μ_1 when a, b are assumed to be independent, and if $(a\ \wedge\ b)$'s probability lies in the interval μ_2 when a, b are assumed to be positively correlated, then c's probability lies in the interval μ." This rule contains no inconsistency as stated above. Rather, such a rule might reflect some doubt on the part of the author of the rule about the precise relationship between a and b — are they independent? Or are they positively correlated? $\square$

Example 4.1.5 Continuing the stock example, we provide here the rules that formalize the situation described in the Introduction.

$$\begin{aligned} \text{price-drop}(C) : [0.4, 0.9] \quad &\longleftarrow \quad (\text{ch-sells-stock}(C)\ \wedge_{igc}\ \text{ch-retires}(C)) : [0.85, 1]. \\ \text{price-drop}(C) : [0.05, 0.2] \quad &\longleftarrow \quad (\text{ch-sells-stock}(C)\ \wedge_{pcc}\ \text{ch-retires}(C)) : [1, 1]. \end{aligned}$$

The first rule states that if the CEO of the company C sells the stock, retires with probability over 85% and we are ignorant about the relationship between the two events, then the probability that the stock of company C drops is 40% – 90%.

The second rule states that if the CEO retires and sells stock, but we know that the former entails the latter, then the probability that the stock of the company will drop is only 5% – 20% . $\square$

Before proceeding to define the declarative semantics of hp-programs, a comment on the use of p-strategies in hp-programs is in order. A programmer may not know the dependences between events (is there no dependency? are the events independent? are they positively correlated? etc.). In such cases, he can write rules such as those shown in Example 4.1.5 in which he explicitly articulates inferences he is willing to make based on different possible event dependencies — the two rules in Example 4.1.5 do *not* require the programmer to know the dependency between the chairman selling stock and retiring, but only specify the inferences he is willing to make in these two eventualities. If he wishes to infer correlations between events, he may use classical statistical correlation methods[94].

4.2 Model Theory and Fixpoint Semantics for Hybrid Probabilistic Programs

Having completed the definition of the syntax of hp-programs, we are now in a position to develop the declarative semantics of such programs. We will first develop a fixpoint semantics of hp-programs, followed by a model theoretic semantics, and show that the two are essentially equivalent characterizations of hp-programs. Later, in Section 7.4, we will provide a proof procedure for hp-programs.

4.2.1 Fixpoint Semantics

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

Definition 4.2.1 *A function $f : B_L \longrightarrow \mathcal{C}[0,1]$ is called an atomic formula function or atomic function.*

It is easy to see that the set of all atomic functions is a complete lattice. This is because if $(X, \sqsubseteq)$ is any complete lattice, then the set of all functions of the type $2^X \rightarrow 2^X$ is a complete lattice also under the pointwise ordering $f \sqsubseteq g$ iff $(\forall x \in X) f(x) \sqsubseteq g(x)$.

Though atomic functions do not, by themselves, make assignments to basic formulas, they may be extended to do so.

Before proceeding further we first introduce some notation for "splitting" a complex formula into two parts.

Definition 4.2.2 *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:*

1. $\{G_1, \ldots, G_k\} \cup \{H_1, \ldots, H_m\} = \{F_1, \ldots, F_n\}$ *and*
2. $\{G_1, \ldots, G_k\} \cap \{H_1, \ldots, H_m\} = \emptyset$.
3. $k > 0$ *and* $m > 0$.

Definition 4.2.3 *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 $G \in bf_{\mathcal{S}}(B_L)$.*

Given an atomic function f and a hybrid formula function h we say that h is based on f iff $(\forall A \in B_L)(f(A) = h(A))$. Sometimes we will use notation h_f to represent the fact that h is based on f.

From the first condition it follows that $h(F) = h(F')$ for any F and F' which are permutations of one another. This property of models allows us, in fact not to distinguish between the formulas and the sets of atoms they are composed of together with a strategy attached. The second condition states that the probability of a complex formula is bounded by the probabilities of its subformulas. Conversely, the third condition bounds the probability of a subformula by the probability of a formula it is a part of. Clearly, for each atomic function f there exists an entire *family* of hybrid formula functions based on it. This corresponds to our intuition that the knowledge of the probabilities of atomic events does not necessarily allow us to uniquely compute the exact probabilities of complex events. In fact, it is possible to express the fact that we possess specific knowledge of a probability of some complex event. The only requirements we put forth onto the hybrid formula functions is that they provide consistent and maximally tight information about the probability intervals associated with both atomic and complex events.

As mentioned above, each atomic function f produces a family of hybrid formula functions h_f based on it. We let $h[f]$ denote the set of all hybrid formula functions based on atomic function f. Let $\mathcal{HFF}$ denote the set of all hybrid formula functions generated by some arbitrary but fixed set of p-strategies. Let $\mathcal{HH}$ denote the set of all functions $h : bf_{\mathcal{S}}(B_L) \longrightarrow \mathcal{C}[\prime, \infty]$. Clearly $\mathcal{HFF} \subseteq \mathcal{HH}$. The $\leq$-ordering on atomic functions may be extended to basic formulas in the obvious way: $h_1 \leq h_2$ iff $(\forall F \in bf_{\mathcal{S}}(B_L))h_1(F) \supseteq h_2(F)$.

We would like to see if there exists any relationship between the orders on atomic and hybrid formula functions. Let f and g be two atomic functions and let $f \leq g$. One would want to see if the statement $(\forall h \in h[f])(\forall h' \in h[g])(h \leq h')$ will hold. As it happens this statement is not true and the following simple example demonstrates it.

Example 4.2.1 Let $B_L = \{a, b\}$. We define the functions f and g as follows:

$f(a) = [0.4, 0.8] \quad g(a) = [0.6, 0.6]$

$f(b) = [0.5, 0.7] \quad g(b) = [0.6, 0.6]$

Clearly $f \leq g$. Now we consider two hybrid formula functions $h \in h[f]$ and $h' \in h[g]$ defined on formula $a \wedge_{inc} b$ as follows:

$h(a \wedge_{inc} b) = [0.4, 0.4]$

$h'(a \wedge_{inc} b) = [0.36, 0.36]$

We can see that both h and h' satisfy the Composition and Decomposition properties of the hybrid formula functions:

$h(a \wedge_{inc} b) = [0.4, 0.4] \subseteq c_{inc}(h(a), h(b)) = c_{inc}([0.4, 0.8], [0.5, 0.7]) = [0.2, 0.56]$.

$h(a) = [0.4, 0.8] \subseteq md_{inc}(h(a \wedge_{inc} b)) = md_{inc}([0.4, 0.4]) = [0.4, 1]$.

$h(b) = [0.5, 0.7] \subseteq md_{inc}(h(a \wedge_{inc} b)) = md_{inc}([0.4, 0.4]) = [0.4, 1]$.

$h'(a \wedge_{inc} b) = [0.36, 0.36] \subseteq c_{inc}(h'(a), h'(b)) = c_{inc}([0.6, 0.6], [0.6, 0.6]) = [0.36, 0.36]$.

$h'(a) = [0.6, 0.6] \subseteq md_{inc}(h'(a \wedge_{inc} b)) = md_{inc}([0.36, 0.36]) = [0.36, 1]$.

$h(b) = [0.6, 0.6] \subseteq md_{inc}(h'(a \wedge_{inc} b)) = md_{inc}([0.36, 0.36]) = [0.36, 1]$.

As $h(a \wedge_{inc} b) \not\subseteq h'(a \wedge_{inc} b)$ it is clear that $h \not\leq h'$. $\square$

In fact, the example above suggests, that even a weaker statement, $(\forall h \in h[f])(\exists h' \in h[g])(h \leq h')$. To show that it is enough to notice that in this example $h[g] = \{h'\}$. The example above suggests that if there is a relationship between the orders of atomic and hybrid functions, this relationship is more subtle. The following theorem establishes this relationship.

Theorem 2 *Let f, g be two atomic functions and let $f \leq g$. Then*

$$(\exists h \in h[f])(\forall h' \in h[g])(h \leq h')$$

Proof. Consider the function $h \in h[f]$ defined as follows: $h(F *_\rho G) = c_\rho(h(F), h(G))$. We will show that $(\forall h' \in h[g])(h \leq h')$.

Let us consider an arbitrary by fixed function $h' \in h[g]$. We prove that $h \leq h'$ by induction on the size of basic formula F. If F is an atom, then $h(F) = f(F) \leq g(F) = h'(F)$. Now, let $F = G *_\rho H$ and let $h(G) \leq h'(G)$ and $h(H) \leq h'(H)$. By definition of h, $h(F) = c_\rho(h(G), h(H))$. As we know that c_ρ is monotonic we get, $c_\rho(h(G), h(H)) \leq c_\rho(h'(G), h'(F))$. But since h' is a formula function, it satisfies Composition and Commutativity and hence $h'(F) = h'(G *_\rho H) \subset c_\rho(h'(G), h'(H))$ or $c_\rho(h'(G), h'(H) \leq h'(F)$. From the above inequalities we get: $h(F) = c_\rho(h(G), h(H))$ $\leq c_\rho(h'(G), h'(H)) \leq h'(F)$ which is the desired result. ■

In order to define the iterations of T_P operator later in the paper, we need the following theorem.

Theorem 3 *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, $h^* \leq h$.*

Proof.

Let F be some hybrid basic formula. If h is a formula function we will write $h(F) = [h^1(F), h^2(F)]$.

We know that the sequence $H_F^1 = h_1^1(F), h_2^1(F), \ldots$ is ascending and bounded (at least by 1). Therefore, by a well-known property of the sequences of real numbers, H_F^1 has a limit $x_F = \lim_{i \to \infty} h_i^1(F)$. By the same property, the descending sequence $H_F^2 = h_1^2(F), h_2^2(F), \ldots$ (bounded by 0) has a limit $y_F = \lim_{i \to \infty} h_i^2(F)$. Since all h_i are fully defined, $x_F \leq y_F$.

Now we define a function h^* as $h^*(F) = [\lim_{i\to\infty} h_i^1(F), \lim_{i\to\infty} h_i^2(F)]$. To prove the desired result it suffices to show that (i) h^* is an upper bound, (ii) h^* is the *least* upper bound and (iii) h^* is a valid hybrid formula function.

- h^* is an upper bound of H.

  We know that the limit of an ascending sequence is greater than or equal to any member of the sequence. Similarly, the limit of a descending sequence is less than or equal to any member of the sequence. But then, for any basic formula F, it is true that $(\forall i)h^*(F) = [\lim_{i\to\infty} h_i^1(F), \lim_{i\to\infty} h_i^2(F)] \subseteq [h_i^1(F), h_i^2(F)] = h_i(F)$. From this it directly follows that $(\forall i)(h_i \leq h^*)$, i.e., h^* is an upper bound of H.

- h^* is the least upper bound of H.

  Let h an upper bound of H, i.e., let $(\forall i)(h_i \leq h)$. Let F be some basic formula. Let us compare $h^1(F)$ and $h^{*1}(F)$. We know that $h^{*1}(F) = \lim_{i\to\infty} h_i^1(F)$. We also know that $(\forall i)(h_i^1(F) \leq h^1(F)$. Then, by the property of real sequences, $lim_{i\to\infty} h_i^1(F) \leq h^1(F)$. Similarly, we can establish that $h^2(F) \leq \lim_{i\to\infty} h_i^2(F)$. From these two inequalities we see that $h(F) \subseteq h^*(F)$, i.e., $h^* \leq h$.

- h^* is a valid hybrid formula function.

  To show this we have to prove that h^* satisfies the Commutativity, Composition and Decomposition.

  First we show that h^* satisfies the Commutativity property. Let F be some basic formula and let G and G' be basic formulas such that $F \equiv G \oplus_\rho G'$ for some p-strategy ρ. Since all functions h_i are valid hybrid formula functions, they all satisfy the Commutativity postulate, i.e., $(\forall i)(h_i(F) = h_i(G *_\rho G'))$, where $* \in$

$\{\wedge, \vee\}$ depending on whether ρ is conjunctive or disjunctive. Then, the sequences $h_1(F), h_2(F), \ldots$ and $h_1(G *_\rho G'), h_2(G *_\rho G'), \ldots$ coincide and hence the limits of the lower and upper bound sequences are the same as well, i.e. $\lim_{i\to\infty}(h_i^1(F)) = \lim_{i\to\infty}(h_i^1(G *_\rho G'))$ and $\lim_{i\to\infty}(h_i^2(F)) = \lim_{i\to\infty}(h_i^2(G *_\rho G'))$.

However, since $h^*(F) = [\lim_{i\to\infty}(h_i^1(F)), \lim_{i\to\infty}(h_i^2(F))]$ and $h^*(G *_i G') = [\lim_{i\to\infty}(h_i^1(G *_\rho G')), \lim_{i\to\infty}(h_i^2(G *_\rho G'))]$, we get $h^*(F) = h^*(G *_\rho G')$.

Now we proceed to show that h^* satisfies the Composition postulate. Let F be some basic formula and let $F = G *_\rho G'$. We need to show that $h^*(F) \subseteq c_\rho(h^*(G), h^*(G'))$. Remember that c_ρ satisfies the axiom of Separation, i.e., $c_\rho = \langle c_\rho^1, c_\rho^2 \rangle$.

First, we show that $c_\rho^1(h^{*1}(G), h^{*1}(G') \leq h^{*1}(F)$. By a similar argument we will then be able to establish that $h^{*2}(F) \leq c_\rho^1(h^*(G), h^*(G')$. The two statements will give us the desired result.

To show $c_\rho^1(h^{*1}(G), h^{*1}(G') \leq h^{*1}(F)$, we first note that $h^{*1}(F) = lim_{i\to\infty} h_i(F) \geq lim_{i\to\infty} c_\rho^1(h_i^1(G), h_i^1(G'))$. We know this because, since all h_i satisfy the Composition axiom – hence $(\forall i)(h_i^1(F) \geq c_\rho^1(h_i^1(G), h_i^1(G'))$, and therefore the sequence $h_1^1(F), h_2^1(F), \ldots$ *dominates* [2] the sequence $c_\rho^1(h_1^1(G), h_1^1(G')), c_\rho^1(h_2^1(G), h_2^1(G')), \ldots$. Then, the limit of the former sequence has to be greater than or equal to the limit of the latter.

As we know that c_ρ is a *continuous* p-strategy, c_ρ^1 is continuous in both arguments. Therefore $\lim_{i\to\infty} c_\rho^1(h_i^1(G), h_i^1(G')) = c_\rho^1(\lim_{i\to\infty} h_i^1(G), \lim_{i\to\infty} h_i^1(G')))$.

But we know that $c_\rho^1(h^{*1}(G), h^{*1}(G')) = c_\rho^1(\lim_{i\to\infty} h_i^1(G), \lim_{i\to\infty} h_i^1(G')))$. Therefore, $c_\rho^1(h^{*1}(G), h^{*1}(G')) \leq h^{*1}(F)$.

[2] I.e. $(\forall i)(h_i^1(F) \geq c_\rho^1(h_i^1(G), h_i^1(G')))$.

As mentioned above, by a similar argument we can show that $c^2_\rho(h^{*2}(G), h^{*2}(G'))$ $\geq h^{*2}(F)$. From these two inequalities it follows that $h^*(F) \subseteq c_\rho(h^*(G), h^*(G'))$.

Now we prove that h^* satisfies the Decomposition postulate. We have to consider two cases. Let $H = F \wedge_\rho G$. The proof for $H = F \vee_\rho G$ will be similar. By definition of md_ρ, $md_\rho(h(H)) = [h^1(H), 1]$ for all formula functions h. As $md^1_\rho(h(H))$ and $md^2_\rho(h(H))$ we will denote the upper and the lower bounds of the $md_\rho(h(H))$ interval.

Now we consider the sequences

$$h^1_1(F), h^1_2(F), \ldots$$

and

$$md^1_\rho(h_1(H)), md^1_\rho(h_2(H)), \ldots$$

As all h_is are formula functions, they satisfy the Decomposition postulate, i.e., $(\forall i)(h_i(F) \subseteq md_\rho(h_i(H))$. Therefore, the sequence $h^1_1(F), h^1_2(F), \ldots$ dominates the sequence $md^1_\rho(h_1(H)), md^1_\rho(h_2(H)), \ldots$.
Therefore, $\lim_{i\to\infty} md^1_\rho(h_i(H)) \leq \lim_{i\to\infty} h^1_i(F)$.

But as mentioned earlier, $(\forall i)(md^1_\rho(h_i(H)) = h^1_i(H))$.

Therefore $\lim_{i\to\infty} md^1_\rho(h_i(H)) = \lim_{i\to\infty} h^1_i(H)$. From the latter equality we establish that $\lim_{i\to\infty} h^1_i(H) \leq \lim_{i\to\infty} h^1_i(F)$. But we know that $\lim_{i\to\infty} h^1_i(H) = h^{*1}(H)$ and similarly $\lim_{i\to\infty} h^1_i(F) = h^{*1}(F)$, which therefore yields $h^{*1}(H) \leq h^{*1}(F)$. Finally, we note that $md^1_\rho(h^*(H)) = h^{*1}(H)$, i.e., $md^1_\rho(h^*(H)) \leq h^{*1}(F)$. As we know that $md^2_\rho(h^*(H)) = 1$ and $h^*(F) \leq 1$, we get the desired: $h^*(F) = [h^{*1}(F), h^{*2}(F)] \subseteq [h^{*1}(H), 1] = [md^1_\rho(h^*(H)), md^2_\rho(h^*(H))] = md_\rho(h^*(H))$, proving that h^* satisfies Decomposition and therefore proving the statement of the theorem. ■

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

Definition 4.2.4 *Let P be a hybrid probabilistic program. Operator $S_P : \mathcal{HFF} \longrightarrow \mathcal{HH}$ is defined as follows (where F is a basic formula):*

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

$M = \{\mu\sigma | F : \mu \longleftarrow F_1 : \mu_1 \wedge \ldots \wedge F_n : \mu_n$ is a ground instance of some clause in P; *σ is a ground substitution of annotation variables and* $(\forall j \leq n) h(F_j) \subseteq \mu_j\sigma\}$ *if* $M = \emptyset$ $S_P(h)(F) = [0, 1]$.

The operator S_P is very simple. Given $h \in \mathcal{HFF}$ and a basic formula F, it proceeds as follows: (i) First, it finds all ground instances of rules in P such that the head of the rule instance is of the form $F : \mu$ and such that for each $F_j : \mu_j$ 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. Note that in the above definition, it is entirely possible that $S_P(h)(F)$ could be the empty set. In this case, there is an intuitive inconsistency, because the formula function $S_P(h)$ is saying that F's probability lies in the empty set. However, this is absurd, as the empty set cannot contain anything. This will be discussed in further detail in Section 4.2.2.

Example 4.2.2 Consider our stock example. Let h assign the following values to the atoms:

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

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

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

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

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

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

Now, suppose we want to compute $S_P(h)$(price-drop(c)). There are two ground rule instances with price-drop(c) as their head in the set $ground(P)$, where $ground(P)$ is the set of all ground instances 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)) $= [0.4, 0.9] \cap [0.5, 1] = [0.5, 0.9]$.

$\square$

However, the S_P operator is not quite "right." The reason is that in order to determine F's probability, it is not enough to merely look for rule instances whose head is identical to F. For instance, F might be $(p \wedge_{ig} q)$. The probability of $(p \wedge_{ig} q)$ may certainly be influenced by rules with head $p : \mu'$ because such rules may impose additional restrictions on p's probability – and hence on $(p \wedge_{ig} q)$'s probability. Thus, S_P, by itself, does not allow us to accurately infer the probability associated with a formula F. In fact, the result of S_P need not be a *hybrid formula function* (notice that in the definition of S_P we made it range over the set $\mathcal{HH}$ of all functions from basic formulas to intervals). Thus, in order to return the desired probability intervals, S_P needs to be augmented in order to account for the abovementioned influences. Before defining T_P operator, which provides necessary enhancements to S_P, we present a simple monotonicity property of S_P. Note that S_P is monotonic regardless of what p-strategies appear in P.

Lemma 3 *S_P is monotonic, i.e., if h_1, h_2 are two formula functions and $h_1 \leq h_2$ then, $S_P(h_1) \leq S_P(h_2)$.*

Proof.

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

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

$$M_1 = \{\mu | F : \mu \longleftarrow F_1 : \mu_1 \wedge \ldots \wedge F_n : \mu_n$$

$$\text{is a ground instance of some clause in } P \text{ ; } (\forall j \leq n) h_1(F_j) \subseteq \mu_j\}$$

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

$$M_2 = \{\mu | F : \mu \longleftarrow F_1 : \mu_1 \wedge \ldots \wedge F_n : \mu_n$$

$$\text{is a ground instance of some clause in } P \text{ ; } (\forall j \leq n) h_2(F_j) \subseteq \mu_j\}$$

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

Let us now define the T_P operator. Intuitively, the T_P operator builds on top of the S_P operator because the probability interval assignments made by the S_P operator to some formulas may allow us to derive sharper bounds for *other* formulas. However, these sharper bounds may not be found by the S_P operator. The T_P operator defined below takes such derivations into account.

Definition 4.2.5 *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*

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

   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 is not atomic) 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:

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

The intuition underlying the T_P operator is as follows: (i) Consider an atomic formula F: if $S_P(h)(F) = \emptyset$, then this means that an inconsistency (to be made more formal in Section 4.2.2) has occurred. For instance, if we have an hp-program containing two facts

$a : [0, 0]$ and $a : [1, 1]$, then whatever h we pick, $S_P(h)(a) = \emptyset$, reflecting the (in this case flagrant) inconsistency in P. Thus, $T_P(h)$ must also assign $\emptyset$ to F. If $S_P(h)(F) \neq \emptyset$, then it may be case that $S_P(h)$ has assigned too "wide" an interval to F, because it ignores rules that are "associated" with F. As F is atomic, there might be rules whose bodies are satisfied by h, which include F in its head. We must find all such rules, and "split" the rule head into its F part, and the non-F part, say G. Clearly, the rule head must be of the form $(F *_\rho G)$ 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 – is 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 $G \oplus H = F$ (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.

It should also be pointed out, that while the T_P operator is defined to be the intersection of many possible intervals, there are **at most two** intervals which will actually affect the final value of $T_P(h)$ for any particular formula F (one interval to provide the lower bound and one interval to provide the upper bound of $T_P(h)(F)$). Because of this, one can see that while the number of intervals to be intersected to obtain $T_P(h)(F)$ according to the definition above can be large, there is a simple nondeterministic algorithm that would perform this computation. This algorithm would guess how the two relevant intervals are obtained, and will only perform computations to produce these two intervals. The following example demonstrates how T_P is computed.

Example 4.2.3 Let us consider the stock program P and the formula function h from the previous example.

Suppose we want to compute $T_P(h)$(price-drop(c)$\wedge_{pcc}$buy-stock(c)) (i.e., the probability of the fact that the drop in price of stocks will result in purchases of new stock of company c).

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

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

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

Recall, that h(price-drop(c)) $= [0.7, 0.9]$ and h(stable(c)) $= [0.5, 0.6]$. Then, h((price-drop(c) $\wedge_{inc}$ stable(c))) $=$

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

Finally, $T_P(h)$(price-drop(c)$\wedge_{pcc}$buy-stock(c)= $c_{pcc}(T_P(h)$(price-drop(c)), $T_P(h)$(buy-stock(c)) $=$

$c_{pcc}([0.5, 0.9], [0.7, 1]) = [\min(0.5, 0.7), \min(0.9, 1)] = [0.5, 0.7]$. □

Let us consider another example:

Example 4.2.4 In this example we will consider a simple knowledge-base about the possible sales of three items: a, b and c. The unary predicate $s(X)$ is to be interpreted as "item X has been sold". Suppose the program P looks as follows:

$s(a) \vee_{ind} s(b) \vee_{ind} s(c) : [0.4, 0.6] \longleftarrow .$

$s(a) \wedge_{igc} s(b) : [0, 0.5] \longleftarrow .$

$s(a) \wedge_{inc} s(c) : [\min(\frac{V}{2} + 0.1, \frac{W}{2}), \frac{W}{2}] \longleftarrow s(c) : [V, W]$

$s(c) : [0, 0.3] \longleftarrow .$

The program consists of three facts (first, second and fourth clauses) and one rule (third clause). The first clause of the program states that the probability that at least one of the three items had been sold under the assumption of independence between possible sales is between 40% and 60% . The second clause states that the probability that both items a and b have been sold computed under assumption of of ignorance about the relationship of possible sales will be not more than 50% . The fourth clause just states that the probability that item c had been sold is no more than 30% .

Finally, the third clause of the program, states that if we know that the probability that item c had been sold is in the range $[V, W]$, then the probability that both items a and c have been sold, considered under the assumption of independence between possible sales, will be not more than $\frac{W}{2}$ and no less than the minimum of $\frac{V}{2} + 0.1$ and $\frac{W}{2}$.

Now let us look at how we can compute the T_P operator for this program. Let us take $h(F) = [0, 1]$ for all basic formulas F (i.e. our h is the bottom function $\perp$).

In this example we will be tracing all atomic formulas ($s(a)$, $s(b)$ and $s(c)$) as well as a few more complex formulas, such as $s(a) \vee_{ind} s(c)$, $s(a) \wedge_{igc} s(b)$ and $s(a) \wedge_{inc} s(c)$.

- First we have to compute $S_P(h)$. Clearly, we have the following:

  $S_P(h)(s(a)) = S_P(h)(s(b)) = [0, 1]$

  $S_P(h)(s(c)) = [0, 0.3]$

  $S_P(h)(s(a) \vee_{ind} s(c)) = [0, 1]$

  $S_P(h)(s(a) \wedge_{igc} s(b)) = [0, 0.5]$

  $S_P(h)(s(a) \wedge_{inc} s(c)) = [0.1, 0.5]$

Every $S_P(h)$ computation except for the last one is straightforward, since for each formula there is either only one rule (with an empty body) in the program that has it as its head, or there are no such rules at all. In the first case, the probability interval from the head of the rule gets to be the value of $S_P(h)$, in the second case, it will be $[0, 1]$.

The last computation requires more effort. Indeed, the third rule of our program is not ground (because of the variable annotation), therefore it will produce more than one ground instance. However, there is only one ground instance of this rule where body is "satisfied" by h:

$$s(a) \wedge_{inc} s(c) : [0.1, 0.5] \longleftarrow s(c) : [0, 1]$$

since $h(s(c)) = [0, 1]$ (we also point out that $\min(\frac{0}{2} + 0.1, \frac{1}{2}) = \frac{0}{2} + 0.1 = 0.1$). Therefore, $S_P(h)(s(c)) = [0.1, 0.5]$.

- Now let us compute the values of the T_P operator.

  1. $T_P(h)(s(a))$.

     $s(a)$ appears in the heads of 3 rules of interest: first and second rules of the program and in the ground instance of the third rule shown above (4.2.4). This means that

     $$T_P(h)(s(a)) = md_{ind}([0.4, 0.6]) \cap md_{igc}([0, 0.5]) \cap md_{inc}([0.1, 0.5]) =$$

     $$= [0, 0.6] \cap [0, 1] \cap [0.1, 1] = [0.1, 0.6]$$

  2. $T_P(h)(s(b))$.

     $s(b)$ appears in the heads of the first two rules of the program. Therefore:

     $$T_P(h)(s(b)) = md_{ind}([0.4, 0.6]) \cap md_{igc}([0, 0.5]) = [0, 0.6] \cap [0, 1] = [0, 0.6]$$

3. $T_P(h)(s(c))$.

   $s(c)$, besides constituting the head of the fourth rule of the program, is also a part of the head of the first rule the program and the head of the rule (4.2.4). Applying the definition of the T_P operator here we obtain:

   $$T_P(h)(s(c)) = S_P(h)(s(c)) \cap md_{ind}([0.4, 0.6]) \cap md_{inc}([0.1, 0.5]) =$$

   $$= [0, 0.3] \cap [0, 0.6] \cap [0.1, 0.5] = [0.1, 0.3]$$

4. $T_P(h)(s(a) \vee_{ind} s(c))$.

   $s(a) \vee_{ind} s(c)$ appears as part of the head of the first rule of the program. By definition of the T_P operator:

   $T_P(h)(s(a) \vee_{ind} s(c)) = md_{ind}([0.4, 0.6]) \cap c_{ind}(T_P(h)(s(a)), T_P(h)(s(c)))$
   $= [0, 0.6] \cap c_{ind}([0.1, 0.6], [0.1, 0.5]) = [0, 0.6] \cap [0.1 + 0.1 - 0.1 \cdot 0.1,$
   $0.6 + 0.5 - 0.6 \cdot 0.5] = [0, 0.6] \cap [0.19, 0.8] = [0.19, 0.6]$

5. $T_P(h)(s(a) \wedge_{igc} s(b))$.

   $T_P(h)(s(a) \wedge_{igc} s(b)) = S_P(h)(s(a) \wedge_{igc} s(b)) \cap c_{igc}(T_P(h)(s(a)),$
   $T_P(h)(s(b))) = [0, 0.5] \cap c_{igc}([0.1, 0.6], [0, 0.6]) = [0.0.5] \cap [\max(0, 0.1 +$
   $0, -1),$
   $\min(0.6, 0.6)] = = [0, 0.5] \cap [0, 0.6] = [0, 0.5]$

6. $T_P(h)(s(a) \wedge_{inc} s(c))$.

   $T_P(h)(s(a) \wedge_{inc} s(c)) = S_P(h)(s(a) \wedge_{inc} s(c)) \cap c_{inc}(T_P(h)(s(a)),$
   $T_P(h)(s(c))) = = [0.1, 0.5] \cap c_{inc}([0.1, 0.6], [0.1, 0.3]) = [0.1, 0.5] \cap$
   $[0.1 \cdot 0.1, 0.6 \cdot 0.3] = = [0.1, 0.5] \cap [0.01, 0.18] = [0.1, 0.18]$

$\square$

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

Theorem 4 *T_P is monotonic, i.e., if h_1, h_2 are two formula functions and $h_1 \leq h_2$ then, $T_P(h_1) \leq T_P(h_2)$.*

Proof.
Let F be a hybrid basic formula. We proceed by induction on $rank(F)$, i.e. number of atoms in the formula.

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

$$M_1 = \{\mu | (F *_\rho G) : \mu \longleftarrow F_1 : \mu_1 \wedge \ldots \wedge F_n : \mu_n$$

$$\text{where } * \in \{\vee, \wedge\} \text{ and } \rho \in \mathcal{S} \text{ and } (\forall j \leq n) h_1(F_j) \subseteq \mu_j\}$$

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

$$M_2 = \{\mu | (F *_\rho G) : \mu \longleftarrow F_1 : \mu_1 \wedge \ldots \wedge F_n : \mu_n$$

$$\text{is a ground instance of some clause in } P \text{ ; } (\forall j \leq n) h_2(F_j) \subseteq \mu_j\}$$

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

Since $T_P(h_1)(F) = S_P(h_1)(F) \cap (\cap M'_1)$, $T_P(h_2)(F) = S_P(h_2)(F) \cap (\cap M'_2)$, $S_P(h_1)(F) \leq S_P(h_2)(F)$ and $\cap M'_1 \leq \cap M'_2$, we obtain that

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

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

  From Lemma 3 we know that $S_P(h_1)(F) \leq S_P(h_2)(F)$.

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

$$c_\rho(T_P(h_2)(G), T_P(h_2)(H)) \subseteq c_\rho(T_P(h_1)(G), T_P(h_1)(H))$$

  i.e.

$$c_\rho(T_P(h_1)(G), T_P(h_1)(H)) \leq c_\rho(T_P(h_2)(G), T_P(h_2)(H))$$

  From this is follows that

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

$$(\cap\{c_\rho(T_P(h_2)(G), T_P(h_2)(H)) | G \oplus H = F\})$$

  Finally, let $M_1 = \{D_1 *_\rho \ldots *_\rho D_k : \mu \longleftarrow E_1 : \mu_1 \wedge \ldots E_m : \mu_m \in ground(P)|$ $(\forall 1 \leq j \leq m) h_1(E_j) \subseteq \mu_j; \{F_1, \ldots, F_n\} \subset \{D_1, \ldots D_k\}, n < k\}$ and $M_2 = \{D_1 *_\rho \ldots *_\rho D_k : \mu \longleftarrow E_1 : \mu_1 \wedge \ldots E_m : \mu_m \in ground(P)|$ $(\forall 1 \leq j \leq m) h_2(E_j) \subseteq \mu_j; \{F_1, \ldots, F_n\} \subset \{D_1, \ldots D_k\}, n < k\}$ Let $M'_1 = \{\mu|$ $D : \mu \longleftarrow Body \in M_1\}$ and $M'_2 = \{\mu | D : \mu \longleftarrow Body \in M_2\}$

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

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

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

Definition 4.2.6 *1.* $T_P^0 = h_\perp$ *where* $\perp$ *is the atomic function that assigns* $[0, 1]$ *to all ground atoms* A.

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

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

In [22] it was established that if all clauses in P have only constant annotations then $lfp(T_P) = T_P^\omega$, where $lfp(T_P)$ is the least fixed point of T_P. This, however, turns out to not be the case when P has clauses with variable annotations. The following example is from [82].

Example 4.2.5 ([82]) 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 narrow the interval assigned to A by half. As 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. $\Box$

Example 4.2.6 Let us return to the two rule HPP in Example 4.1.5.

$$\begin{array}{rcl} \textsf{price-drop}(C) : [0.4, 0.9] & \longleftarrow & (\textsf{ch-sells-stock}(C) \wedge_{igc} \textsf{ch-retires}(C)) : [0.85, 1]. \\ \textsf{price-drop}(C) : [0.05, 0.2] & \longleftarrow & (\textsf{ch-sells-stock}(C) \wedge_{pcc} \textsf{ch-retires}(C)) : [1, 1]. \end{array}$$

Suppose we have in addition, the two facts:

$$\begin{array}{rcl} \textsf{ch-sells-stock}(ibm) : [1, 1] & \leftarrow & . \\ \textsf{ch-retires}(ibm) : [0.9, 1] & \leftarrow & . \end{array}$$

In this case, the assignment made by T_P^ω to $\textsf{price-drop}(ibm)$ is $[0.4, 0.9]$ as the first rule of the program will fire and the second – won't.

$\Box$

Example 4.2.7 Now, if in addition to the two rules from Examples 4.1.5 and 4.2.6 we add one fact

$$(\textsf{ch-sells-stock}(ibm) \wedge \textsf{ch-retires}(ibm)) : [1, 1] \leftarrow .$$

$T_P^\omega(\textsf{price-drop(ibm)})$ will be equal to $\emptyset$. Indeed, the above fact makes the second rule fire immediately. Decomposing this fact we obtain $T_P^\omega(\textsf{ch-sells-stock}(ibm)) = [1, 1]$

and $T_P^\omega(\textsf{ch-retires}(ibm)) = [1,1]$ which is sufficient to make the first rule fire. Intersecting $[0.4, 0.9]$ and $[0.05, 0.2]$ leads to the assignment of $\emptyset$ to $T_P^\omega(\textsf{price-drop(ibm)})$. $\square$

Definition 4.2.7 *A hybrid probabilistic program P is said to* satisfy the fixpoint reachability condition *iff*

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

Intuitively, if an hp-program P satisfies the fixpoint reachability condition, then for every formula F, if $\mu \supseteq lfp(T_P)(F)$, then there is a finitely long justification of this fact.

4.2.2 Probabilistic Model Theory

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

Definition 4.2.8 *Let h be a hybrid 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, denoted $h \models P$, iff $(\forall p \in P)(h \models p)$. $\square$

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

Recall, from Section 4.2.1, that we can have cases where a hybrid formula function, h, could assign $\emptyset$ to some formula. When $h(F) = \emptyset$, h is "saying" that F's probability lies in the empty set. This corresponds to an inconsistency because, by definition, nothing is in the empty set.

Definition 4.2.9 *Formula function h is called fully defined iff*
$\forall(F \in bf_{\mathcal{S}}(B_L))(h(F) \neq \emptyset)$.

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

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

Proof.

1. $T_P(h) \leq h \Longrightarrow h \models P$.

   Let $F \in bf_{\mathcal{S}}(B_L)$.

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

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

Let $P' \neq \emptyset$.

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

- $(\forall\ 1 \leq j \leq n)(\mu_j \leq h(F_j))$ In this case, we know that

  $h \models F_1 : \mu_1 \wedge \ldots \wedge F_n : \mu_n$. We have to show that

  $h \models F : \mu$, i.e. $h(F) \subseteq \mu$.

  By our assumption, $T_P(h)(F) \leq h(F)$, i.e., $h(F) \subseteq T_P(h)(F)$. By definition of T_P and S_P operators, it is always the case that $T_P(h)(F) \subseteq S_P(h)(F)$. We now show that $S_P(h)(F) \subseteq \mu$.

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

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

This proves the first claim.

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

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

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

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

$$\begin{aligned}
& \ldots \\
F : \mu_k & \longleftarrow \quad \ldots \\
(F *_{\rho_1} G_1) : \nu_1 & \longleftarrow \quad \ldots \\
& \ldots \\
(F *_{\rho_m} G_1) : \nu_m & \longleftarrow \quad \ldots
\end{aligned}$$

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

By definition of T_P, $T_P(h)(F) = \mu_1 \cap \mu_2 \cap \ldots \cap \mu_k \cap md_{\rho_1}(\nu_1) \cap \ldots \cap md_{\rho_m}(\nu_m)$.

Since h satisfies all the bodies of these rules, h must also satisfy all the heads, i.e.,

$(\forall\ 1 \leq j \leq k)(h(F) \subseteq \mu_j)$ and $(\forall\ 1 \leq j \leq l)(h(F *_{\rho_j} G) \subseteq \nu_j)$. From first set of inequalities we obtain: $h(F) \subseteq \mu_1 \cap \mu_2 \cap \ldots \cap \mu_k$.

From second set of inequalities: $h(F *_{\rho_j} G) = c_{\rho_j}(h(F), h(G))$ and therefore $h(F) \subseteq md_{\rho_j}(\nu_j)$. This leads to $h(F) \subseteq md_{\rho_1}(\nu_1) \cap \ldots \cap md_{\rho_m}(\nu_m)$, which combined with previous result gives us desired $h(F) \subseteq T_P(h)(F)$ i.e., $T_P(h)(F) \leq h(F)$.

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

  Let

$$\begin{aligned}
& F : \mu_1 \longleftarrow \ldots \\
& \ldots \\
& F : \mu_k \longleftarrow \ldots
\end{aligned}$$

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

Let now G and H be basic formulas such that $G \oplus H = F$. By definition, $rank(g) < k$ and $rank(H) < k$, therefore, by the induction hypothesis, $h(G) \subseteq T_P(h(G))$ and $h(H) \subseteq T_P(h(H))$. Since $G \oplus H = F$, $G *_\rho H \equiv F$ and therefore $h(F) = h(G *_\rho H) \subseteq c_\rho(h(G), h(H)) \subseteq c_\rho(T_P(h)(G), T_P(h)(F)$ (the last inequality is due to monotonicity property of composition function). Therefore we conclude that

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

Now, let

$$(F *_\rho D_1) : \nu_1 \longleftarrow \ldots$$

$$\ldots$$

$$(F *_\rho D_s) : \nu_s \longleftarrow \ldots$$

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

Combining the three inequalities together we obtain:

$$\begin{aligned} h(F) &\subseteq S_P(F) \cap (\cap\{c_\rho(T_P(h)(G), T_P(h)(H))|G \oplus H = F\}) \cap \\ &\quad (md_\rho(\nu_1) \cap \ldots \cap md_\rho(\nu_s)) \\ &= T_P(h)(F) \end{aligned}$$

which proves the theorem.

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

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

3. Part 3 of this theorem is a direct corollary of Part 2 and Theorem 4 . □

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

Chapter 5

Proof Procedures for HPLPs

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

- The first query processing procedure (Section 5.2), termed *hp-resolution*, builds upon previous approaches of Ng and Subrahmanian [82] by first requiring that programs P be compiled to a new set, $CL(P)$. Queries are then processed by a process akin to linear input resolution, with the difference that clauses from $CL(P)$ may be considered input clauses. This process suffers from the major flaw that usually, construction of $CL(P)$, which is based on computation of $lfp(T_P)$, is prohibitively expensive. Because of that, two more refutation procedures have been introduced.

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

- The third procedure, which will be studied in more detail in Section 6, expands upon HR_P to use tabling, as initially introduced in logic programming by Tamaki and Sato[102]. This procedure assumes caches (or tables) are bounded a priori in size – a situation certainly true in practical implementations where tables cannot grow in an unbounded fashion. Furthermore, table management in probabilistic logic programs is much more complicated than in ordinary logic programming for many reasons. First, a query does not merely have a set of answers. Rather, a query has associated answer substitutions, each of which has an associated probability range. As computation proceeds, these ranges may get refined or sharpened – something that does not happen in classical logic program tables. Second, caches in our framework may contain basic formulas with associated probabilities. Such caches implicitly contain probability ranges for basic formulas implied by the cached formulas, as well as basic formulas that imply the cached formulas. A third difference between our work and classical logic program tabling is that there are often many ways to update a table in the case of probabilistic logic programs. We define cache update strategies, and show several different such strategies. We show how HR_P-refutations may be extended with arbitrary cache update strategies.

Unlike classical resolution, when dealing with annotated conjunctions and disjunctions, unifiers may not be unique, as noted by Ng and Subrahmanian[82]. Before proceeding to describe our different notions of resolution, we summarize some observations of [82] below as they are necessary for the further development of our proof procedures.

5.1 Unification in HPPs

As rules of clauses in hp-programs may contain annotated basic formulas, any notion of unification must be able to handle unification of annotated basic formulas. In this section, we recapitulate from [82, pps 175–179] how this may be done. The contents of this section are not new.

Definition 5.1.1 • Θ *is a unifier of annotated conjunctions*

$C_1 \equiv A_1 \wedge_\rho \ldots \wedge_\rho A_{n_1}$ *and* $C_2 \equiv B_1 \wedge_{\rho'} \ldots \wedge_{\rho'} A_{n_2}$ *iff* $\rho, \rho' \in \mathcal{CONJ}$ *and* $\rho = \rho'$ *and* $\{A_k\Theta | 1 \leq k \leq n_1\} = \{B_k\Theta | 1 \leq k \leq n_2\}$

• Θ *is a unifier of annotated disjunctions*

$D_1 \equiv A_1 \vee_\rho \ldots \vee_\rho B_{n_1}$ *and* $D_2 \equiv B_1 \vee_{\rho'} \ldots \vee_{\rho'} B_{n_2}$ **iff** $\rho, \rho' \in \mathcal{DISJ}$ and $\rho = \rho'$ and $\{A_k\Theta | 1 \leq k \leq n_1\} = \{B_k\Theta | 1 \leq k \leq n_2\}$ □

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

Definition 5.1.2 *Let* $U(C_1, C_2)$ *denote the set of all unifiers of* C_1 *and* C_2. *Let* $\Theta_1, \Theta_2 \in U(C_1, C_2)$.

1. $\Theta_1 \leq \Theta_2$ **iff** *there exists a substitution* γ, *such that* $\Theta_1 = \Theta_2\gamma$.

2. $\Theta_1 \equiv \Theta_2$ **iff** $\Theta_1 \leq \Theta_2$ *and* $\Theta_2 \leq \Theta_1$.

3. *Let* $[\Theta] = \{\Theta' \in U(C_1, C_2) | \Theta \equiv \Theta'\}$.

4. $[\Theta_1] \leq [\Theta_2]$ **iff** *there exists such* γ *that* $[\Theta_1] = [\Theta_2\gamma]$.

5. $[\Theta_1] < [\Theta_2]$ **iff** $[\Theta_1] \leq [\Theta_2]$ *and* $[\Theta_2] \neq [\Theta_1]$.

□

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

Definition 5.1.3 *$\Theta \in U(C_1, C_2)$ is a maximally general unifier (max-gu) of C_1 and C_2* **iff** *there is* **no** *such other unifier $\Theta' \in U(C_1, C_2)$ that $[\Theta] \leq [\Theta']$.* □

The proof of the following result is quite complex and is given in [82, Lemma 12,pps 176-179].

Lemma 4 *([82, Lemma 12,pps 176-179]) If two basic formulas are unifiable then they have a max-gu (not necessarily unique).*

5.2 hp-Resolution

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

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

Definition 5.2.1 *Let P be an hp-program. Then $REDUN(P)$ is defined as*

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

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

Definition 5.2.2 *Let P be an hp-program. Then CL(P) (closure of P) is defined as follows*

- *CL*$^0(P) = REDUN(P)$.
- 1. *For each pair of clauses $F_1 : \mu_1 \longleftarrow Body_1$ and $F_2 : \mu_2 \longleftarrow Body_2 \in$ 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 CL$^{j+1}(P)$.*
  2. *For each clause $F_1 *_\rho F_2 : \mu \longleftarrow Body \in$ CL$^j(P)$ add the following two clauses to 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 $(B_1 *_\rho \ldots *_\rho B_l) : \mu_1 \longleftarrow Body_2 \in$ 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 CL$^{j+1}(P)$.

4. *if A and B are atoms, and 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 $\rho \in \mathcal{CONJ} \cup \mathcal{DISJ}$ to CL$^{j+1}(P)$.

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

$\square$

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

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

Proof. Let C be a clause in CL(P). Then $C \in$ CL$^j(P)$ for some integer $j \geq 0$. We proceed by induction on j.

- **Base Case.**

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

- **Induction Step.**

  Assume that for each clause $C \in$ CL$^j(P)$, $P \models C$. Let $C \in$ CL$^{j+1}(P) -$ CL$^j(P)$. As $C \in$ CL$^{j+1}(P) -$ CL$^j(P)$, C must have been inserted into CL$^{j+1}(P)$ by the means of one of the cases $1 - 4$ from Definition 5.2.2. We have to consider each case separately.

1. Suppose C was inserted by the means of case 1. Then there exist such clauses $C_1 \equiv F_1 : \mu_1 \longleftarrow Body_1$ and $C_2 \equiv F_2 : \mu_2 \longleftarrow Body_2$, such that, $C_1 \in \mathrm{CL}^j(P), C_2 \in \mathrm{CL}^j(P)$, F_1 and F_2 are unifiable via max-gu Θ, and

$$C \equiv (F_1 : \mu_1 \cap \mu_2 \longleftarrow Body_1 \wedge Body_2)\Theta$$

We need to show that $P \models C$. Suppose h is a model of P, i.e., $h \models P$, and $C\gamma$ is a ground instance of C, such that $h \models (Body_1 \wedge Body_2)\Theta\gamma$. By the induction hypothesis, $h \models C_1$ and $h \models C_2$, therefore, $h \models C_1\Theta\gamma$ and $h \models C_2\Theta\gamma$. As $h \models Body_1\Theta\gamma$, we conclude that $h(F_1\Theta\gamma) \subseteq \mu_1$. Likewise we can conclude that $h(F_2\Theta\gamma) \subseteq \mu_2$.

But since Θ is a max-gu of F_1 and F_2, $F_1\Theta\gamma = F_2\Theta\gamma$, and therefore $h(F_1\Theta\gamma) \subseteq \mu_1 \cap \mu_2$, i.e. $h \models F_1\Theta\gamma : \mu_1 \cap \mu_2$.

2. Suppose C was inserted by the means of case 2. Then there exists such a clause $C_1 \equiv (F_1 *_\rho F_2) : \mu \longleftarrow Body \in \mathrm{CL}^j(P)$, that either

$$C \equiv F_1 : md_\rho(\mu) \longleftarrow Body$$

or

$$C \equiv F_2 : md_\rho(\mu) \longleftarrow Body.$$

We will consider the former case, the latter case is symmetric. We need to show that $P \models C$. Let $C\gamma$ be a ground instance of C and let $h \models P$ and $h \models Body\gamma$. By induction hypothesis, $h \models C_1$, and therefore, $h((F_1 *_\rho F_2)\gamma) \subseteq \mu$. By the definitions of md_ρ and h, this yields $h(F_1) \subseteq md_\rho(\mu)$, i.e., $h \models F_1 : md_\rho(\mu)$.

3. Let C be inserted by the means of case 3. In this case, $\mathrm{CL}^j(P)$ will contain two clauses, $C_1 \equiv (A_1 *_\rho \ldots *_\rho A_k) : \mu_1 \longleftarrow Body_1$ and $C_2 \equiv (B_1 *_\rho \ldots *_\rho$

$B_l) : \mu_2 \longleftarrow Body_2$, such that, $k > 1, l \geq 1$, and

$$C \equiv (A_1 *_\rho \ldots *_\rho A_k *_\rho B_1 *_\rho \ldots *_\rho B_l) : c_\rho(\mu_1, \mu_2) \longleftarrow Body_1 \wedge Body_2.$$

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

$$h((A_1 *_\rho \ldots *_\rho A_k *_\rho B_1 *_\rho \ldots *_\rho B_l)\gamma) =$$

$$c_\rho(h((A_1 *_\rho \ldots *_\rho A_k)\gamma), h((B_1 *_\rho \ldots *_\rho B_l)\gamma)) \subseteq c_\rho(\mu_1, \mu_2),$$

which means $h \models C$.

4. Finally, let C be inserted in $\mathrm{CL}^{j+1}(P)$ by the means of case 4. Then, $\mathrm{CL}^j(P)$ will contain 2 clauses, $C_1 \equiv A : \mu_1 \longleftarrow Body_1$ and $C_2 \equiv B : \mu_2 \longleftarrow Body_2$, such that both A and B are atomic, and

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

for some p-strategy ρ.

We have to show $P \models C$. Let $C\gamma$ be a ground instance of C and let $h \models P$ and $h \models (Body_1 \wedge Body_2)\gamma$. By induction hypothesis, $h \models C_1$ and $h \models C_2$, therefore, $h \models C_1\gamma$ and $h \models C_2\gamma$. Since $h \models Body_1\gamma$ and $h \models Body_2\gamma$, we obtain $h \models A\gamma : \mu_1$ and $h \models B\gamma : \mu_2$, i.e. $h(A\gamma) \subseteq \mu_1$ and $h(B\gamma) \subseteq \mu_2$. Hence, $h((A *_\rho B)\gamma) = c_\rho(h(A\gamma), h(B\gamma)) \subseteq c_\rho(\mu_1, \mu_2)$, which means that $h \models C\gamma$ and therefore $h \models C$. □

We now present a refutation procedure for query processing.

Definition 5.2.3 *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_i*s need not be ground.* □

Definition 5.2.4 *Suppose* $C \equiv G : \lambda \longleftarrow G_1 : \lambda_1 \wedge \ldots \wedge G_m : \lambda_m \in$ **CL**(P) *and* $Q \equiv \exists(F_1 : \mu_1 \wedge \ldots \wedge F_n : \mu_n)$ *is a query. Let* C *and* Q *be standardized apart. Let also* G *and* F_i *be unifiable for some* $1 \leq i \leq n$*. Then*

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

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

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

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

Definition 5.2.5 *Let* $Q \equiv \exists(F_1 : \mu_1 \wedge \ldots \wedge F_n : \mu_n)$ *be an initial query, and* P *an hp-program. An hp-deduction of* Q *from* P *is a sequence* $\langle Q_1, C_1, \Theta_1\rangle \ldots \langle Q_r, C_r, \Theta_r\rangle \ldots$ *where,* $Q = Q_1$*, for all* $i \geq 1$ *,* C_i *is a renamed version of a clause in* **CL**(P) *and* Q_{i+1} *is an* hp-resolvent *of* Q_i *and* C_i *via max-gu* Θ_i*.*
If the Θ_i*'s are not restricted to be max-gu's, we call the resulting sequence an unrestricted hp-deduction.* □

Definition 5.2.6 *Let* $Q \equiv \exists(F_1 : \mu_1 \wedge \ldots \wedge F_n : \mu_n)$ *be an initial query, and* P *be an hp-program. An hp-refutation of* Q *from* P *is a hp-deduction*

$$\langle Q_1, C_1, \Theta_1\rangle, \ldots, \langle Q_r, C_r, \Theta_r\rangle,$$

where the hp-resolvent of Q_r *and* C_r *via* Θ_r *is the empty query.* $\Theta_1 \ldots \Theta_r$ *is called the computed answer substitution.* □

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

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

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

Base case: $n = 1$

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

Induction Step.

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

In order to prove a completeness theorem we have to establish first a number of facts. The following two lemmas can be proved by a straightforward application of mgu and lifting lemmas for classical logic programming in Lloyd[69]. Mirror image proofs are given in [82].

Lemma 6 *(***Max-gu Lemma***). Let Q be a query that has an* unrestricted *hp-refutation from an hp-program P. Then, Q has an hp-refutation of the same length and if $\Theta_1, \ldots,$ Θ_m are the unifiers form the unrestricted hp-refutation, and $\Theta'_1, \ldots, \Theta'_m$ are the max-gu's from the hp-refutation, then, for some γ $\Theta_1 \ldots \Theta_m = \Theta'_1, \ldots, \Theta'_m\gamma$.*

Lemma 7 *(***Lifting Lemma***) Let P be an hp-program, Q be a query, Θ be a substitution. Let $Q\Theta$ have an hp-refutation from P. Then Q has an hp-refutation from P of the same length. Also, if $\Theta_1, \ldots, \Theta_m$ are the max-gu's from the refutation of $Q\Theta$ and $\Theta'_1, \ldots, \Theta'_m$ are the max-gu's from the refutation of Q then, for some substitution γ: $\Theta\Theta_1 \ldots \Theta_m = \Theta'_1, \ldots, \Theta'_m\gamma$.*

We can now prove our completeness theorem.

Theorem 7 (Completeness of hp-refutation).
Let P be a consistent hp-program which satisfies the fixpoint reachability condition (see Def. 4.2.7) and Q' be a query. Then, if $P \models \exists(Q')$ then there exists an hp-refutation of Q' from P.

Proof.
Since $P \models \exists(Q')$, there exists such a ground substitution Θ that $P \models Q'\Theta$.. Let $Q \equiv Q'\Theta$. We will prove that Q has an hp-refutation from P. By Lifting Lemma, Q' will also have a refutation from P.
Let $Q \equiv F_1 : \mu_1 \wedge \ldots \wedge F_m : \mu_m$. Since $P \models Q$, it must be the case that $P \models F_i : \mu_i$, $1 \leq i \leq m$.

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

Proof. Let $\langle F : \mu, C_1, \Theta_1 \rangle \ldots \langle Q_l^F, C_l, \Theta_l \rangle$ be the hp-refutation for $F : \mu$. Let $\langle G : \nu, D_1, \Gamma_1 \rangle \ldots \langle Q_k^G, D_k \Gamma_k \rangle$ be the hp-refutation for $G : \nu$. Then, as $F : \mu$ and $G : \nu$ are ground, the following will be the hp-refutation for $F : \mu \wedge G : \nu$:

$$\langle F : \mu \wedge G : \nu, C_1, \Theta_1 \rangle, \langle Q_2^F \wedge G : \nu, C_2, \Theta_2 \rangle \ldots$$

$$\ldots \langle Q_l^F \wedge G : \nu, C_l, \Theta_l \rangle, \langle G : \nu, D_1, \Gamma_1 \rangle \ldots \langle Q_k^G, D_k \Gamma_k \rangle$$

This completes the proof of Claim 1. □

Now all we have to prove is

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

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

Since $P \models F : \mu$, by Theorem 5, $T_P^{\omega}(F) \subseteq \mu$. By definition of T_P^{ω}, there exists such an $\alpha < \omega$ that $T_P^{\alpha}(F) \subseteq \mu$. Consider the smallest such integer. We now proceed by induction on α.

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

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

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

is in $REDUN(P)$, and therefore it is in CL(P). Then, $\langle F : \mu, C, e \rangle$ (e is the empty substitution) is the hp-refutation for $F : \mu$.

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

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

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

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

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

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

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

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

Base Case. F is atomic.

Let $M' = \{\mu' | G : \mu' \longleftarrow Body \in P; T_P^{\alpha-1} \models Body$, where G is unifiable with $F\}$. We notice that $S_P(T_P^{\alpha-1})(F) = \cap\{\mu' | \mu' \in M'\}$.

Let $M'' = \{\mu'' | (G *_\rho H) : \mu'' \longleftarrow Body \in P; T_P^{\alpha-1} \models Body$, where G is unifiable with $F\}$.

We have, by definition of T_P^α ($\alpha > 0$):

$$T_P^\alpha(F) = S_P(T_P^{\alpha-1})(F) \cap (\cap\{md(\mu'') | \mu'' \in M''\}) \subseteq \mu.$$

Two cases are possible.

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

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

   By induction hypothesis, there exists an hp-refutation

$$\langle Body, C_1, \Theta_1 \rangle \ldots \langle Q_k, C_k, \Theta_k \rangle$$

for $Body$. Then

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

is the refutation for $F : \mu$.

Assume now that $M'' \neq \emptyset$. Then there is a unique rule $C' \equiv (G *_\rho H) : \mu' \leftarrow Body \in P$, such, that G unifies with F via max-gu Θ', $T_P^{\alpha-1} \models Body$ and $T_P(T_P^{\alpha-1})(F) = md_\rho(\mu')$. Since $C' \in P$, $C' \in \mathrm{CL}^0(P)$ and therefore the following clause $C'' \equiv G : md_\rho(\mu') \longleftarrow Body$ is in $\mathrm{CL}^1(P)$. By the induction hypothesis, there exists an hp-refutation for $Body$: $\langle Body, C_1, \Theta_1 \rangle \ldots \langle Q_k, C_k, \Theta_k \rangle$. Then

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

is the refutation for $F : \mu$.

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

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

$M' = \{\mu' | G : \mu' \longleftarrow Body \in \mathcal{C}'\}$.

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

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

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

$$\ldots$$

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

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

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

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

Let

$$(D_1 *_{\rho_1} H_1) : \mu''_1 \longleftarrow Body''_1$$

$$\ldots$$

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

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

$$D_1 : md_{\rho_1}(\mu''_1) \longleftarrow Body''_1$$

$$\ldots$$

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

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

$$C_2 \equiv D_1\Theta'' : md_{\rho_1}(\mu''_1) \cap \ldots \cap md_{\rho_r}(\mu''_r) \longleftarrow Body''_1 \wedge \ldots \wedge Body''_r$$

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

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

$$\begin{aligned} G_1\Theta'_1 : \mu'_1 \cap \ldots \cap \mu'_s \cap md_{\rho_1}(\mu''_1) \cap \ldots \cap md_{\rho_r}(\mu''_r) \quad \longleftarrow \quad & Body'_1 \wedge \ldots \wedge Body'_s \wedge \\ & Body''_1 \wedge \ldots \wedge Body''_r \end{aligned}$$

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

Notice that $\mu'_1 \cap \ldots \cap \mu'_s \cap md_{\rho_1}(\mu''_1) \cap \ldots \cap md_{\rho_r}(\mu''_r) = T_P^{\alpha}(F) \subseteq \mu$. Also, by induction hypothesis, each $Body'_j$ and $Body''_j$ has an hp-refutation, therefore by Claim 1 of the theorem, their conjunction has an hp-refutation.

Let $\langle Q_1, C_1, \Theta_1\rangle, \ldots, \langle Q_z, C_z, \Theta_z\rangle$ be such an hp-refutation. Than the following is an hp-refutation for $F : \mu$:

$$\langle F : \mu, C, \Gamma\rangle, \langle Q_1, C_1, \Theta_1\rangle, \ldots, \langle Q_z, C_z, \Theta_z\rangle$$

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

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

Let $\mathcal{C}_1 = \{G : \mu' \longleftarrow Body \in P | T_P^{\alpha-1} \models Body$, where G is unifiable with $F\}$, and $M_1 = \{\mu' | G : \mu' \longleftarrow Body \in \mathcal{C}_2\}$.

Let $\mathcal{C}_2 = \{(D *_\rho E) : \mu'' \longleftarrow Body \in P | T_P^{\alpha-1} \models Body$, where D is unifiable with $F\}$ and $M_2 = \{\mu'' | (D *_\rho E) : \mu'' \longleftarrow Body \in \mathcal{C}_2\}$.

Let

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

$$\ldots$$

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

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

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

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

Let

$$(D_1 *_{\rho_1} E_1) : \mu_1'' \longleftarrow Body_1''$$

$$\ldots$$

$$(E_r *_{\rho_r} E_r) : \mu_r'' \longleftarrow Body_r''$$

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

$$D_1 : md_{\rho_1}(\mu_1'') \longleftarrow Body_1''$$

$$\ldots$$

$$D_r : md_{\rho_r}(\mu_r'') \longleftarrow Body_r''$$

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

$$C_2^F \equiv D_1\Theta'' : md_{\rho_1}(\mu_1'') \cap \ldots \cap md_{\rho_r}(\mu_r'') \longleftarrow Body_1'' \wedge \ldots \wedge Body_r''$$

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

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

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

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

and

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

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

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

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

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

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

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

$$C^{H*_\rho I} \equiv (H' *_\rho I') : c_\rho(\lambda', \lambda'') \longleftarrow Body' \wedge Body''$$

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

$$\langle (H' *_\rho I') : c_\rho(\lambda', \lambda''), C^{H*_\rho I}, \Theta\rangle, \langle Body' \wedge Body'', C_2^H, \Theta_2^H\rangle \ldots$$

$$\langle Q_t^H \wedge Body'', C_t^H, \Theta_t^H\rangle, \langle Body'', C_2^I, \Theta_2^I\rangle \ldots, \langle Q_u^I, C_u^I, \Theta_u^I\rangle$$

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

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

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

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

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

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

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

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

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

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

This completes the proof of the completeness theorem. $\square$

It is important to note that the above theorem only holds when P satisfies the fixpoint reachability condition. Past work on annotated logics [82] makes this assumption, and as this paper generalizes [82, 56], we have not removed this assumption.

The hp-refutation paradigm extends a proof procedure developed in [82] for probabilistic logic programs under the ignorance assumption. In particular, Items (4) and (5) in the definition of $CL(P)$ given in Definition 5.2.2 do not occur in the proof procedure in [82]. The need for these two rules derives directly from the use of arbitrary p-strategies.

This causes the proof procedure of [82] to be much simpler (and easier to implement) than that given in this paper.

5.3 HR_P Refutations for HP-programs

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

Thus, constructing $CL(P)$ before construction of a refutation is attempted, is often completely infeasible in practice. To avoid this *a priori* computation of $CL(P)$, we provide a new procedure that allows that part of $CL(P)$ needed in a refutation to be dynamically computed on an "as-needed" basis. The HR_P refutation framework described here avoids the construction of $CL(P)$.

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

Definition 5.3.1 *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 $A_1, A_2 \in B_L$*

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

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

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

- *Decomposition:*

L-Decomposition	*R-Decomposition*
$\frac{(F*_\rho G):\mu\longleftarrow Body}{F:md_\rho(\mu)\longleftarrow Body}$	$\frac{(F*_\rho G):\mu\longleftarrow Body}{G:md_\rho(\mu)\longleftarrow Body}$

- *Clarification:*

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

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 in P or can be constructed from one or more members 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 this by $P \vdash C$.* □

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

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

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

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

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

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

On the induction step, we consider a clause C added to CL$^n(P)$. By Definition 5.2.2 C was added to CL$^n(P)$ by one of four rules. Since these rules match exactly the four inference rules of HR_P and by induction hypothesis for every clause $C' \in$ CL$^{n-1}(P)$ we know that $P \vdash_{HR_P} C'$, we can obtain the proof of C in HR_P by application of a matching rule to the same clauses. $\square$

Definition 5.3.2 (HR_P-refutations) *Let $Q \equiv \exists(F_1 : \mu_1 \wedge \ldots \wedge F_n : \mu_n)$ be an initial query, and P an hp-program. An hp-refutation via HR_P of Q from P is a finite sequence $\langle Q_1, C_1, \Theta_1\rangle \ldots \langle Q_r, C_r, \Theta_r\rangle$ where,*

- $Q_1 = Q$
- Q_r *is empty*

- $P \vdash C_i$ *for all* $1 \le i \le r$
- Q_{i+1} *is an hp-resolvent of* Q_i *and* C_i *with max-gu* Θ_ρ*, for all* $1 \le i < r$.

The following results tell us that hp-refutations using HR_P are both sound and complete and thus,they constitute the first sound and complete proof procedure for probabilistic logic programs (including those in [82]) that do not require the construction of a program closure. Here is a simple example of HR_P-refutations.

Example 5.3.1 **(**HR_P **refutations)** Consider the HP-program P given by:

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

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

$Q_1 = Q = a[0.9,1] \wedge e : [1,1]$

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

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

$P \vdash C_2 = (b \wedge_{ind} c \wedge_{ind} d) : [0.3,1] \longleftarrow (c \wedge_{ind} d) : [0.3,1].$

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

$P \vdash C_3 = (c \wedge_{ind} d) : [0.3,1]) \longleftarrow$

$Q_4 = f : [0.5,0.9] \wedge e : [1,1]$

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

$Q_5 = b : [1,1] \wedge e : [1,1]$

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

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

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

$Q_7 = e : [1, 1]$

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

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

$P \vdash C_8 = (b \wedge_{ind} c \wedge_{ind} d) : [0.3, 1] \longleftarrow (c \wedge_{ind} d) : [0.3, 1].$

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

$P \vdash C_9 = (c \wedge_{ind} d) : [0.3, 1]) \longleftarrow$

$Q_{10} = f : [0.5, 0.1].$

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

$Q_{11} = b : [1, 1].$

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

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

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

$Q_{13} = \Box$

Here is another example of an HR_P refutation, for a program that contains variable annotations.

Example 5.3.2 Let us consider the program from Example 4.2.4. For convenience we repeat P below:

$s(a) \vee_{ind} s(b) \vee_{ind} s(c) : [0.4, 0.6] \longleftarrow .$
$s(a) \wedge_{igc} s(b) : [0, 0.5] \longleftarrow .$
$s(a) \wedge_{inc} s(c) : [\min(\frac{V}{2} + 0.1, \frac{W}{2}), \frac{W}{2}] \longleftarrow s(c) : [V, W]$
$s(c) : [0, 0.3] \longleftarrow .$

Let us now look at how the refutation of the query $Q = (s(a) \wedge_{inc} s(c)) : [0.15, 0.15]$ will proceed:

$Q_1 = Q = (s(a) \wedge_{inc} s(c)) : [0.15, 0.15]$

$P \vdash C_1 = s(a) \wedge_{inc} s(c) : [0.15, 0.15] \longleftarrow s(c) : [0.15, 0.3]$

$Q_2 = s(c) : [0.15, 0.3]$

$P \vdash C_2 = s(c) : [0.15, 0.3] \longleftarrow s(c) : [0.1, 0.3]$

$Q_3 = s(c) : [0.1, 0.3]$

$P \vdash C_3 = s(c) : [0.1, 0.3] \longleftarrow s(c) : [0, 0.6]$

$Q_4 = s(c) : [0, 0.6]$

$P \ni C_4 = s(c) : [0, 0.3] \longleftarrow .$

$Q_5 = \Box.$

Here is how the derivations of C_1, C_2 and C_3 are done:

- Rule $C_1 = s(a) \wedge_{inc} s(c) : [0.15, 0.15] \longleftarrow s(c) : [0.15, 0.3]$ is a ground instance of the rule

  $$s(a) \wedge_{inc} s(c) : [\min(\frac{V}{2} + 0.1, \frac{W}{2}), \frac{W}{2}] \longleftarrow s(c) : [V, W]$$

  with $V = 0.15$ and $W = 0.3$ ($\min(\frac{0.15}{2} + 0.1, \frac{0.3}{2}) = \frac{0.3}{2} = 0.15$).

- $C_2 = s(c) : [0.15, 0.3] \longleftarrow s(c) : [0.1, 0.3]$ is derived as follows: $C_2' = s(a) \wedge_{inc} s(c) : [0.15, 0.15] \longleftarrow s(c) : [0.1, 0.3]$ is a ground instance of the rule

  $$s(a) \wedge_{inc} s(c) : [\min(\frac{V}{2} + 0.1, \frac{W}{2}), \frac{W}{2}] \longleftarrow s(c) : [V, W]$$

  with $V = 0.15$ and $W = 0.3$. Applying the inference rule of R-Decomposition to C_2' we obtain $C_2'' = s(c) : [0.15, 1] \longleftarrow s(c) : [0.1, 0.3]$. Combining C_2'' with the clause $C_2^* = s(c) : [0, 0.3] \longleftarrow . \in P$ using the inference rule of Clarification we obtain C_2 as $[0, 0.3] \cap [0.15, 1] = [0.15, 0.3]$.

- The derivation of $C_3 = s(c) : [0.1, 0.3] \longleftarrow s(c) : [0, 0.6]$ is similar to the derivation of C_2:

$C_3' = s(a) \wedge_{inc} s(c) : [0.1, 0.3] \longleftarrow s(c) : [0, 0.6]$ is a ground instance of the rule

$$s(a) \wedge_{inc} s(c) : [\min(\frac{V}{2} + 0.1, \frac{W}{2}] \longleftarrow s(c) : [V, W]$$

where $V = 0$ and $W = 0.6$ ($\min(\frac{0}{2} + 0.1, \frac{0.6}{2}) = \frac{0}{2} + 0.1 = 0.1$). Applying the inference rule of R-Decomposition to C_3' we obtain the rule $C_3'' = s(c) : [0.1, 1] \longleftarrow s(c) : [0, 0.6]$. Combining C_3'' with $C_3^* = s(c) : [0, 0.3] \longleftarrow . \in P$ using the inference rule of Clarification we obtain C_3 as $[0, 0.3] \cap [0.1, 1] = [0.1, 0.3]$.

The soundness and completeness of HR_P-refutations follow immediately from the soundness and completeness theorems for HP-refutation and soundness and completeness theorems for HR_P w.r.t. CL(P).

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

Proof. Suppose $\langle Q_1, C_1, \Theta_1\rangle, \ldots, \langle Q_r, C_r, \Theta_r\rangle$ is an HR_P refutation of Q. Let $\mathbf{C} = \{C_i \mid C_i \notin P\}$. By bullet (3) in the definition of HR_P-refutations, it follows that $P \vdash C_i$ for all $C_i \in \mathbf{C}$. By Theorem 8, we know each such C_i is in $CL(P)$, and hence, $\langle Q_1, C_1, \Theta_1\rangle, \ldots, \langle Q_r, C_r, \Theta_r\rangle$ is an hp-refutation. By the soundness of hp-refutation (Theorem 6), the result follows. □

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.

Proof. By the completeness of hp-refutations (Theorem 7), it follows that there exists an hp-refutation of Q' from P. Suppose $\langle Q_1, C_1, \Theta_1\rangle, \ldots, \langle Q_r, C_r, \Theta_r\rangle$ is such an hp-refutation. Then, by definition of hp-refutations, each C_i is in $CL(P)$. But then, by Theorem 9, each C_i is either in P, or is such that $P \vdash C$, and hence,

$$\langle Q_1, C_1, \Theta_1\rangle, \ldots, \langle Q_r, C_r, \Theta_r\rangle$$

is also an HR_P refutation. $\square$

Before concluding this section, we briefly reiterate that HR_P refutations avoid compile-time construction of $CL(P)$ – an expensive and time/space consuming process.

Chapter 6

Tabulation Strategies for Hybrid Probabilistic Programs

We are now ready to study efficient tabled query processing techniques for HPPs. In this section, we first define *caches* and *bounded caches*. Intuitively, a cache contains formulas with established probability ranges. As resolution based processing of a query occurs, we will gain information about certain basic formulas. These will need to be "added" to the cache. For this purpose, we define in this section, a family of *updating strategies* and introduce several example strategies. Later, in Section 6.3, we show how to use these tables and table update strategies hand in hand with the resolution based proof procedure.

6.1 B-Caches and B-Cache Update Strategies

Definition 6.1.1 *A cache is a finite set of annotated basic formulas. If b is an integer, a bounded b-cache is a finite set of annotated basic formulas containing at most b atoms each.*

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

Definition 6.1.2 *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 $T[F] = \mu$.

We fix an integer $b > 0$, a logical language L as defined in Lloyd[69], and a set $\mathcal{S}$ of p-strategies. Let $\mathcal{T}[b, L, \mathcal{S}]$ denote the set of all possible b-caches over $bf_S(B_L)$. Whenever b, L and $\mathcal{S}$ are clear from the context we may use $\mathcal{T}$ instead of $\mathcal{T}[b, L, \mathcal{S}]$.

We are interested in developing a resolution-based query processing procedure that is irredundant in the sense that it does not "re-infer" facts that it has already inferred. In the case of classical logic programs, caches and their utilization are relatively simple: caches contain facts; when performing resolution on an atom A in the query, we check to see if A is subsumed by the cache (Tamaki and Sato[102]). An alternative approach is due to Warren et. al. who check the cache for variants of A [29, 16]. However, in the case of probabilistic logic programs, b-caches are somewhat more complicated.

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

Rather than defining a specific update procedure, we first proceed by defining a notion of an update strategy - a function that takes a b-cache and a basic formula as input, and returns a new "improved" b-cache. We will establish a number of basic properties of any update strategy. Later we will define a number of specific update strategies that are "natural" or "intuitive".

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

Definition 6.1.3 *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 6.1.4 *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 8 (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. If $P \models T$ and $P \models F : \mu$ then $P \models T \uplus_f \{F : \mu\}$.*

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

1. $F' : \mu' \in T$. In this case, since $P \models T$, it has to be $P \models F' : \mu'$.

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

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

Definition 6.1.5 *Let $S = \{F_1 : \mu_1, \ldots, F_n : \mu_n\}$ be a finite set of annotated basic formulas and u- a b-cache update strategy. We define*

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

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

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

Definition 6.1.6 *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 2 *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 $CN(T) = CN(T \uplus_f F : \mu)$.*

Proof. Since $T \models F : \mu$, $CN(T) = CN(T \cup \{F : \mu\})$. Since $CN(T \uplus_f F : \mu) \subseteq CN(T \cup \{F : \mu\})$ and $CN(T) \subseteq CN(T \uplus_f F : \mu)$ we obtain the desired equality.

□

6.2 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 6.2.1 (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.*

$\square$

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 6.2.1 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 $T \uplus_{at} \{A : \mu\}$ for various different annotated atoms (we consider variables in all the formulas to be standardized apart).

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

□

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 6.2.2 (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 to 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 – as every updating formula gets broken down 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 6.2.2 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. $\square$

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

Definition 6.2.3 (Elementary b-cache update) *Let T be a b-cache, F be a basic formula. We define an elementary b-cache update strategy (denoted $\uplus_{e_b}$) as follows:*

1. *$|F|$ = 1. (F is atomic).*

   $T \uplus_{e_b} \{F : \mu\} = T \uplus_{at} \{F : \mu\}$

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. $|F| > b$.

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 see that

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

Proof.

(i). Let F be a basic formula and $\mu \in \mathcal{C}[0,1]$. Three cases are possible:

- F is atomic.

  In this case, by definition of eb, $T \uplus_{eb} F : \mu$ = $T \uplus_{at} F : \mu$ = $T \uplus_{pat} F : \mu$.

- $1 < |F| \leq b$.

  In this case, $T \uplus_{eb} F : \mu$ is computed starting from $T' = T \uplus_{pat} F : \mu$ via a series of iterations which modify/add information about formulas unifiable with F. This means that for all basic formulas G not unifiable with F and for all intervals $\nu \in \mathcal{C}[0,1]$, if $G : \nu \in CN(T')$ (i.e., $T' \models G : \nu$) then $G : \nu \in CN(T \uplus_{eb} F : \mu)$ (i.e., $T \uplus_{eb} F : \mu \models G : \nu$).

  Let H be a basic formula unifiable with F, and let $T' \models H : \nu$. This means that there exists a substitution Θ such that $\langle \nu, \Theta \rangle \in T'[F]$ and $H = F\Theta$. But then, by definition of elementary b-cache update strategy, $T \uplus_{eb} F : \mu$ will contain formula $F\Theta : \mu \cap \nu = H : \mu \cap \nu$. Clearly, $\mu \cap \nu \subseteq \nu$ and therefore, $\{H : \mu \cap \nu\} \models H : \nu$, i.e., $T \uplus_{eb} F : \mu \models H : \nu$.

  From the above we infer that $CN(T \uplus_{pat} F : \mu) \subset CN(T \uplus_{eb} F : \mu)$.

- $|F| > b$.

  Let S be the set of all subformulas of F of size b. By definition of the elementary b-cache update, we get:

  $T \uplus_{\mathsf{e}_b} F : \mu = (T \uplus_{pat} F : \mu) \uplus_{\mathsf{e}_b} S$.

  But by definition of an update strategy, we get $CN(T \uplus_{pat} F : \mu) \subseteq CN((T \uplus_{pat} F : \mu) \uplus_{\mathsf{e}_b} S = CN(S)$

(ii). To prove that $\mathrm{e}1 \equiv pat$ we first note that for any formula F, one of two possible cases holds:

- F is atomic.

  In this case $T \uplus_{\mathsf{e}1} F : \mu = T \uplus_{pat} F : \mu$ by definition.

- F is not atomic.

  In this case $|F| > 1$. Let $F = A_1 *_\rho \ldots *_\rho A_k$. By definition of elementary b-cache update $T \uplus_{\mathsf{e}1} F : \mu = (\ldots (T \uplus_{pat} F : \mu) \uplus_{\mathsf{e}1} A_1 : md_\rho(\mu)) \ldots \uplus_{\mathsf{e}1} A_k : md_\rho(\mu)) = (\ldots((T \uplus_{pat} F : \mu) \uplus_{pat} A_1 : md_\rho(\mu)) \ldots \uplus_{pat} A_k : md_\rho(\mu))$ (the latter equality holds, since all A_i are atomic). Since $(\forall 1 \leq i \leq k) F : \mu \models A_i : md_\rho(\mu)$ we conclude that $CN((T \uplus_{pat} F : \mu) \uplus_{pat} A_i : md_\rho(\mu)) \subseteq CN(T \uplus_{pat} F : \mu)$. On the other hand, by definition of an update strategy, we know that the reverse ($CN((T \uplus_{pat} F : \mu) \uplus_{pat} A_i : md_\rho(\mu)) \supseteq CN(T \uplus_{pat} F : \mu)$) is true. Therefore $CN((T \uplus_{pat} F : \mu) \uplus_{pat} A_i : md_\rho(\mu)) = CN(T \uplus_{pat} F : \mu)$ which implies that

  $$(\ldots((T \uplus_{pat} F : \mu) \uplus_{pat} A_1 : md_\rho(\mu)) \ldots \uplus_{pat} A_k : md_\rho(\mu)) \equiv T \uplus_{pat} F : \mu.$$

$\square$

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

Example 6.2.3 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 6.2.4 (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. *$|F|$ = 1. (F is atomic). $T \uplus_{fb} F : \mu = T \uplus_{at} F : \mu$*

2. *Let $F \equiv F_1 *_\rho \ldots *_\rho F_m$, $m \leq b$. Let $B_1, \ldots B_k$ be <u>all</u> 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. *$|F| > b$. 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 4 *(i)* $(\forall b > 0) fb \geq eb$

(ii) $f1 \equiv e1 \equiv pat$

Proof. *(i)*. Let $b > 0$, $T \in \mathcal{T}$, $F \in bf_{\mathcal{S}}(B_L)$ and $\mu \in \mathcal{C}[0, 1]$. Three cases are possible:

1. $|F| = 1$ (i.e., F is atomic). In this case $T \uplus_{fb} F : \mu = T \uplus_{at} F : \mu = T \uplus_{eb} F : \mu$.

2. $1 < |F| \leq b$. Let $B_1 \ldots B_k$ be all proper subformulas of F. Then $T \uplus_{\mathrm{f}_b} \{F : \mu\} = T \uplus_{\mathrm{e}_b} F : \mu \uplus_{\mathrm{e}_b} B_1 : md_\rho(\mu) \uplus_{\mathrm{e}_b} \ldots \uplus_{\mathrm{e}_b} B_k : md_\rho(\mu)$. Then, by definition of an update strategy, $CN(T \uplus_{\mathrm{e}_b} F : \mu) \subseteq CN(T \uplus_{\mathrm{f}_b} \{F : \mu\})$.

3. $|F| > b$. Let Let $S_B = \{B_1 \ldots B_k\}$ be all subformulas of F; let $S_G = \{G_1, \ldots, G_s\}$ be all subformulas of F of size strictly less than b and $S_H = \{H_1, \ldots H_r\}$ be all subformulas of F of size of exactly b. Clearly $S_B = S_H \cup S_G$.

   Using the commutativity property of b-cache update strategies we can obtain the following:
   $T \uplus_{\mathrm{f}_b} F : \mu = (T \uplus_{pat} F : \mu) \uplus_{\mathrm{e}_b} B_1 : md_\rho(\mu) \uplus_{\mathrm{e}_b} \ldots \uplus_{\mathrm{e}_b} B_k : md_\rho(\mu) = (T \uplus_{pat} F) \uplus_{\mathrm{e}_b} S_H \uplus \mathrm{e}bS_G = T \uplus_{\mathrm{e}_b} F : \mu \uplus_{\mathrm{e}_b} S_G$.

   From this we immediately conclude $CN(T \uplus_{\mathrm{e}_b} F : \mu) \subseteq CN(T \uplus_{\mathrm{f}_b} F : \mu)$.

(ii). Same as the proof of part (ii) of Proposition 3. □

The reader may easily observe that while implementing atomic updates is easy, 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.

6.3 Proof Procedure for HP-Programs with b-cache

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

Informally the desired resolution procedure works as follows. Initially we have query Q, program P, a b-cache update strategy u and our b-cache T is (initially) empty. On each resolution step, we select a basic formula $F : \mu$ from current query and perform a lookup for the probability range of this formula in our current b-cache. To do this we have to compute $T[F]$. Once $T[F]$ is computed we compare it to μ. In case $T[F] \subseteq \mu$ we consider the current resolution step done. Otherwise, we use the above refutation procedure 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 6.3.1 *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 $\hat{Q} \equiv \langle (F_{i_1} \mu_{i_1}, \emptyset), \ldots, (F_{i_n} : \mu_{i_n}, \emptyset) \rangle$.*
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 6.3.2 *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 $\hat{Q} \equiv \langle (G_1 : \mu_1, S_1), \ldots, (G_m : \mu_m, S_m) \rangle$, 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 $\langle \mu, \Theta \rangle \in T[G_1]$, such that, $\mu \subseteq \mu_1$. Let $C \equiv G_1\Theta : \mu \longleftarrow$. Then*

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

   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* $\langle \mu', \Theta' \rangle \in T[G_1]$, *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:*

$$\hat{Q}' \equiv \langle (F_{i_1}\Theta : \lambda_{i_1}, \emptyset), \ldots, (F_{i_n}\Theta : \lambda_{i_n}, S_1\Theta \cup \{G_1\Theta\}),$$

$$(G_2\Theta : \mu_2, S_2\Theta), \ldots, (G_m\Theta : \mu_m, S_m\Theta) \rangle.$$

   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 6.3.3 *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*

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

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 6.3.1 (2-cache supported hp-refutation with elementary update strategy) Let us return to the hp-program shown in Example 5.3.1 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. $Q_1 = \langle (a : [0.9, 1], \emptyset), (e : [1, 1], \emptyset) \rangle$

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

2. $Q_2 = \langle ((b \wedge_{ind} c \wedge_{ind} d) : [0.25, 1], \emptyset), (f : [0.5, 0.9], \{a : [1, 1]\}), (e : [1, 1], \emptyset) \rangle$

   $T_2 = \emptyset; P \vdash C_2 = (b \wedge_{ind} c \wedge_{ind} d) : [0.3, 1] \longleftarrow (c \wedge_{ind} d) : [0.3, 1]$.

3. $Q_3 = \langle ((c \wedge_{ind} d) : [0.3, 1], \{(b \wedge_{ind} c \wedge_{ind} d) : [0.3, 1]\}), (f : [0.5, 0.9], \{a : [1, 1]\}), (e : [1, 1], \emptyset) \rangle$

   $T_3 = \emptyset; P \vdash C_3 = (c \wedge_{ind} d) : [0.3, 1]) \longleftarrow$

4. $Q_4 = \langle (f : [0.5, 0.9], \{a : [1, 1]\}), (e : [1, 1], \emptyset) \rangle$

   $T_4 = (T_3 \uplus_{2e} (c \wedge_{ind} d) : [0.3, 1])) \uplus_{2e} (b \wedge_{ind} c \wedge_{ind} d) : [0.3, 1] = \{c : [0.3, 1], d : [0.3, 1], b : [0.3, 1], (c \wedge_{ind} d) : [0.3, 1], (b \wedge_{ind} c) : [0.3, 1], (b \wedge_{ind} d) : [0.3, 1]\}$

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

5. $Q_5 = \langle (b : [1, 1], \{f : [0.7, 0.9], a : [1, 1]\}), (e : [1, 1], \emptyset) \rangle$

   $T_5 = T_4; T[b] = [0.3, 1] \not\subseteq [1, 1]$;

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

6. $Q_6 = \langle ((c \wedge_{ind} d) : [0.3, 1], \{b : [1, 1], f : [0..7, 0.9], a : [1, 1]\}, (e : [1, 1], \emptyset)\rangle$

   $T_6 = T_5 = T_4; T_6[(c \wedge_{ind} d)] = [0.3, 1] \subseteq [0.3, 1]$

7. $Q_7 = \langle (e : [1, 1], \emptyset)\rangle$

   $T_7 = (T_6 \uplus_{2e} (c \wedge_{ind} d) : [0.3, 1]) \uplus_{2e} \{b : [1, 1], f : [0.7, 0.9], a : [1, 1]\} = \{c : [0.3, 1], d : [0.3, 1], b : [1, 1], (c \wedge_{ind} d) : [0.3, 1], (b \wedge_{ind} c) : [0.3, 1], (b \wedge_{ind} d) : [0.3, 1], f : [0.7, 0.9], a : [1, 1]\}$

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

8. $Q_8 = \langle ((b \wedge_{ind} c \wedge_{ind} d) : [0.25, 1], \emptyset), (f : [0.5, 0.1], \{e : [1, 1]\}\rangle$

   $T_8 = T_7; T_8[(b \wedge_{ind} c \wedge_{ind} d)] = [0.3, 1] \subseteq [0.25, 1]$

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. $Q_{10} = \Box$

$\Box$

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

Proof. Suppose

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

is a b-cache supported hp-refutation of Q w.r.t. HR_P. We proceed by induction on r.

Base Case ($r = 1$)**.** In this case, as $T_1 = \emptyset$, it is immediate that $\langle \hat{Q}_1, C_1, \Theta_1 \rangle$ is an HR_P-refutation of Q and the result follows immediately by the soundness theorem for HR_P-refutations.

Induction Step ($r + 1$)**.** Consider the b-cache supported refutation

$$\langle \hat{Q}_2, C_2, \Theta_2, T_2 \rangle \ldots \langle \hat{Q}_r, C_r, \Theta_r, T_r \rangle$$

— it is easy to see that this may be viewed as a b-cache supported refutation of Q_2 from $P \cup T_2$. Hence, by the inductive hypothesis, $(P \cup T_2) \models (\forall) Q_2 \Theta'$ where $\Theta' = \Theta_2 \cdots \Theta_r$.

But T_2 only contains $\phi_u(\hat{Q}_1, T_1, C_1, \Theta_1)$ and as u is a b-cache supported update policy, it follows that $CN((P \cup T_2)) \supseteq CN(\phi_u(\hat{Q}_1, T_1, C_1, \Theta_1))$. Therefore, $\phi_u(\hat{Q}_1, T_1, C_1, \Theta_1) \models (P \cup T_2)$ which proves the theorem. □

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.

Proof. The proof is immediately obtained from the fact that every HR_P-refutation is a b-cache supported refutation — to see why, observe that the fourth bullet in the definition of a b-cache supported refutation requires either $P \vdash C_i$ or $T \vdash C_i$. The first case is the same as for HR_P refutations, and thus, every HR_P-refutation is a b-cache supported refutation. As HR_P-refutations are complete, so are b-cache supported refutations. □

Chapter 7

Algorithms and Complexity Results

In this section we study the complexity of a variety of problems related to the semantics of HPPs. In particular, we show that for *ground HPPs*, the complexity of the entailment problem (answers to queries to HPPs) for HPPs is polynomial for HPPs with *atomic heads of rules*, and in many cases for HPPs with *at most two atoms in the heads*. However, when *formulas of size three or more* are allowed in the heads of the rules, the complexity of query processing becomes NP-complete. We establish some other complexity results for related problems, such as checking the consistency of an HPP.

In this section only ground Hybrid Probabilistic Programs are considered. The following problems are studied below:

1. **Computation:** We will first study (Section 7.1) the problem of computing the set of all ground basic formulas that are logical consequences of an HPP P.

2. **Entailment:** Second (in Section 7.2), given a query Q and an HPP P as input, we will study the complexity of checking if Q is a logical consequence of P.

3. **Consistency:** Finally, in Section 7.3, given as input, a HPP P, we will examine the complexity of checking if P is consistent.

Obviously, these three problems are closely related to one another.

7.1 Complexity of model computation

Let P be a ground hp-program over the set of atoms $\{A_1, \ldots, A_N\}$.

Definition 7.1.1 *For a clause $C \equiv F : \mu \leftarrow F_1 : \mu_1 *_\rho \ldots *_\rho F_n : \mu_n \in P$ let*

$$act_P(C) = min\{i | T_P^i(F_j) \subseteq \mu_j \ (1 \leq j \leq n)\}.$$

In other words, $act_P(C)$ denotes the minimal step i at which C becomes active (i.e., just after all formulas in the body of C are satisfied). Let $Fired_P(i)$ denote the set $\{C | act_P(C) = i\}$.

The following result provides us with an exact bound on the number of steps required to compute the fixpoint of the T_P operator.

Lemma 9 *1. If for some i we have $Fired_P(i) = Fired_P(i{+}1) = \emptyset$, then $lfp(T_P) = T_P^{i+1}(h_\perp)$*

2. If P consists of m ground clauses then $lfp(T_P) = h_P = T_P^{2m}$.

Proof.

1. Given a basic formula F we will use induction on the number of atoms in F to prove that $T_P^{i+1}(F) = T_P^{i+2}(F)$. The statement of the claim will then follow immediately.

   Base case. Let F be an atomic formula.

   From Definition 4.2.4 it follows that if $Fired_P(i) = Fired_P(i + 1) = \emptyset$ then for any basic formula F, $S_P(T_P^i)(F) = S_P(T_P^{i+1})(F)$.

   Since no new rules are fired in step $i + 1$ the set

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

$$(\forall 1 \leq j \leq n)(T_P^i(F_j) \subseteq \mu_j)\}$$

is equal to the set

$$M^{i+1} = \{\langle \mu, \rho \rangle | (F *_\rho G) : \mu \longleftarrow F_1 : \mu_1 \wedge \ldots \wedge F_n : \mu_n \in P;$$

$$(\forall 1 \leq j \leq n)(T_P^{i+1}(F_j) \subseteq \mu_j)\}.$$

Then, $T_P(T_P^i)(F) = T_P(T_P^{i+1}(F))$, therefore, $T_P^{i+1}(F) = T_P^{i+2}(F)$.

Induction step.

Let F consist of n atoms. Similarly to the atomic case $S_P(T_P^i)(F) = S_P(T_P^{i+1})(F)$ and the set M' in def.4.2.5.2 remains the same at steps i and $i+1$. Also, for any two basic formulas G and H such that $G \oplus_\rho H = F$ by induction hypothesis we get that

$$c_\rho(T_P(T_P^i(G)), T_P(T_P^i(H))) = c_\rho(T_P(T_P^{i+1}(G)), T_P(T_P^{i+1}(G))).$$

Then $T_P(T_P^i)(F) = T_P(T_P^{i+1}(F))$, and therefore, $T_P^{i+1}(F) = T_P^{i+2}(F)$.

2. If P consists of m clauses then $lfp(T_P) = T_P^{2m}$ follows immediately from the first claim of the lemma since for every two steps i, $i+1$ at least one new clause from P should be fired. ■

The following example demonstrates that the bound of $2m$ in Lemma 9.2 cannot be decreased.

Example 7.1.1 Let igc and inc be the conjunctive ignorance and independence. Let us consider the following program P:

$$r_0 : \quad a \wedge_{igc} b_0 : [0.5, 1] \quad \leftarrow$$

$$r_1 : \quad b_1 : [0.5, 1] \quad \leftarrow \quad a \wedge_{inc} b_0 : [0.5, 1]$$

$$r_2 : \;\; b_2 : [0.5, 1] \;\; \longleftarrow \;\; b_0 \wedge_{inc} b_1 : [0.5^2, 1]$$

$$\ldots$$

$$r_{i+1} : \;\; b_{i+1} : [0.5, 1] \;\; \longleftarrow \;\; b_0 \wedge_{inc} b_1 \wedge_{inc} \ldots \wedge_{inc} b_i : [0.5^{i+1}, 1]$$

$$\ldots$$

$$r_n : \;\; b_n : [0.5, 1] \;\; \longleftarrow \;\; b_0 \wedge_{inc} b_1 \wedge_{inc} \ldots \wedge_{inc} b_{n-1} : [0.5^n, 1]$$

P consists of $n+1$ rules $r_0, r_1, ...r_n$. It easy to check that $act_P(r_i) = 2i+1$ for every $0 \leq i \leq n$. Then

$T^1_P(a) = T^1_P(b_0) = [0.5; 1]$, and for $i > 1$ $T^{2i+1}_P(b_i) = [0.5, 1]$ and at the step $2(i+1)$ we have $T^{2(i+1)}_P(b_0 \wedge_{ind} b_1 \wedge_{ind} \ldots \wedge_{ind} b_i) = [0.5^{i+1}, 1]$. Hence, it takes $2(n+1)$ steps to get h_P. □

The following lemma shows how one can compute T^{i+1}_P from T^i_P.

Lemma 10 *Let P be an hp-program.*

1. *For any atomic formula F*

$$T^{i+1}_P(F) = T^i_P(F) \cap (\cap\{md_\rho(\mu) | \langle \mu, \rho \rangle \in M^{i+1}\}) \cap$$

$$(\cap\{\mu | F : \mu \longleftarrow Body \in Fired_P(i)\})$$

where $M^{i+1} = \{\langle \mu, \rho \rangle | (F *_\rho G) : \mu \longleftarrow Body \in Fired_P(i)\}$

2. *For any non-atomic formula F*

$T^{i+1}_P(F) = T^i_P(F) \cap (\cap\{\mu | F : \mu \longleftarrow Body \in Fired_P(i)\}) \cap (\cap\{md_\rho(\mu) | \langle \mu, \rho \rangle \in M^{i+1}\}) \cap$

$(\cap\{c_\rho(T^{i+1}_P(G), T^{i+1}_P(H)) | G \oplus_\rho H = F\})$

Proof. The proof follows directly from definitions 4.2.4 and 4.2.5. ■

Given a basic formula G, we define the *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 $\max\{width(B_1), \ldots, width(B_k)\}$.

We may now define a hierarchy of subclasses of HPPs in terms of the head/body widths of the clauses involved.

Definition 7.1.2 *$HPP_{k,r}$ denotes 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 $HPP_k = \bigcup_{r \geq 0} HPP_{k,r}$.*

Algorithm LFP below shows how we may compute the least fixpoint of T_P for class $HPP_{k,r}$.

Algorithm 1 LFP

Input: $P \in HPP_{k,r}$, $N \geq \max(k, r)$, m - number of clauses in P, $F_1, \ldots F_M$ - all formulas of *width* $\leq N$, lexicographically ordered.
Output: table $t_{2m}(F_k)$, $1 \leq k \leq M$.

BEGIN (algorithm)
1. FOR $j = 1$ TO M DO $t_0(F_j) := [0, 1]$;
2. FOR $i = 0$ TO $2m - 1$ DO
  BEGIN
    3. FOR EACH $C = G : \mu \leftarrow G_1 : \mu_1 \wedge \ldots \wedge G_n : \mu_l \in P$ such that for all $1 \leq j \leq n$ $t_i(G_j) \subseteq \mu_j$ DO
      BEGIN
        4. $t_{i+1}(G) := t_i(G) \cap \mu$;
        5. delete C from P;
        6. FOR EACH H included in G, i.e. $G = H \oplus_\rho H'$ DO
          7. $t_{i+1}(H) := t_i(H) \cap md_\rho(\mu)$;
      END
    8. FOR $j = 1$ TO M DO
    9. FOR $k = j + 1$ TO M DO
    10. IF $F_k = F_j \oplus_\rho F_l$ (for some $l < k$)
        11. THEN $t_{i+1}(F_k) := t_{i+1}(F_k) \cap c_i(t_{i+1}(F_j), t_{i+1}(F_l))$;
  END
END.

The following theorem proves that Algorithm LFP is a correct way of computing the least fixpoint of T_P for programs in the class $HPP_{k,r}$ and establishes its complexity.

Theorem 14 *Let P be any program in $HPP_{k,r}$ with m clauses. Let a be the number of different atoms in P, s be the number of different strategies in P, and $N \geq max\{k, r\}$. Then* Algorithm LFP *computes h_P on all the formulas of $bf_S(B_L)$ of width $\leq N$ in time $O(2m(2sa^N)^2) = O(m(sa^N)^2)$.*

Proof. Let $F_1, ..., F_M$ be a lexicographically ordered sequence of all formulas of width $\leq N$ constructed by atoms and strategies of P. Then $M \leq 2sa^N$. Algorithm LFP constructs a sequence of tables $t_0, t_1, \ldots, t_{2m}$. The correctness of the algorithm follows from the following proposition.

Proposition 5 *For every $i = 0, 1, \ldots, 2m$ and for every $k = 1, \ldots M$ $t_i(F_k) = T_P^i(F_k)$.*

> **Proof.** We prove this proposition using induction on i.
>
> **Base Case**. $i = 0$.
>
> Table t_0 is computed in line 1 of the algorithm. For each formula F_j, $t_0(F_j)$ is assigned $[0, 1]$, which is equal to $T_P^0(F_j)$.
>
> **Induction Step**.
>
> Assume that after $i - 1$ iterations of loop 2–11 table t_i constructed is such that for each F_k ($1 \leq k \leq M$) $t_i(F_k) = T_P^i(F_k)$ and that the set of clauses C deleted via line 5. is equal to $\{C | act_P(C) \leq i\} = \cup_{j=0}^{i} Fired_P(j)$.
>
> We will show that after i iterations of this loop $t_{i+1}(F_k) = T_P^{i+1}(F_k)$ for all ($1 \leq k \leq M$) and that the set of clauses deleted on ith iteration via line 5. is exactly $Fired_P(i + 1)$.

Since $T^i_P(F_k) = t_i(F_k)$, the set of clauses selected from P in line 3. on ith iteration of the loop 2.-11. should belong to $\cup_{j=0}^{i+1} Fired_P(j)$. But by induction hypothesis, after $i-1$ iterations of loop 2 – 11 all clauses in $\cup_{j=0}^{i} Fired_P(j)$ are deleted from P. So, if for some clause C, the condition in line 3 is satisfied on ith iteration of the loop, then $C \in Fired_P(i+1)$. All such clauses are then deleted in executions of line 5 of this iteration of the loop.

Then the assignment statements in lines 4 and 7 imply that after loop 3–7 is completed on ith iteration of the algorithm $t_{i+1}(F_k) = t_i(F_k) \cap (\cap\{\mu | F_k : \mu \longleftarrow Body \in Fired_P(i+1)\}) \cap (\cap\{md_\rho(\mu) | (F \oplus_\rho G) : \mu \longleftarrow Body \in Fired_P(i+1)\})$.

If F_k is atomic the condition in line 10. will never be true for it, therefore the assignment statement in line 11. will never be executed for F_k. Therefore, by Lemma10.1 $T_P^{i+1}(F_k) = t_{i+1}(F_k)$ already after the completion of loop 3–7

We proceed by induction on k to prove that after k executions of loop 8–11 on ith iteration, $T_P^{i+1}(F_k) = t_{i+1}(F_k)$.

Base Case. For all k such that F_k is atomic, the statement is true since because $T_P^{i+1}(F_k)$ was equal to $t_{i+1}(F_k)$ even before the first iteration of the loop, and $t_{i+1}(F_k)$ is never changed inside the loop.

Induction Step. Suppose that for each F_j, ($j < k$), $t_{i+1}(F_j) = T_P^{i+1}(F_j)$ after j iterations of loop 8.–11. Notice that $t_{i+1}(F_j)$ will never appear again on the left side of the assignment statement in any further execution of line 11. Therefore, the value of $t_{i+1}(F_j)$ will not change.

Let M_{11} be the set of all intervals $c_\rho(t_{i+1}(F_j), t_{i+1}(F_l))$ which intersect with

$t_{i+1}(F_k)$ in line 11. Since by induction assumption $t_{i+1}(F_j) = T_P^{i+1}(F_j)$ and $t_{i+1}(F_l) = T_P^{i+1}(F_L)$ we get that $M_{11} \subseteq \{c_\rho(T_P^{i+1}(G), T_P^{i+1}(H)) | F_k = G \oplus_\rho H\}$. Suppose now that for some basic formulas G, H, $F_k = G \oplus H$. Then, there exist such numbers $j, l < k$, that $G = F_j$, $H = F_l$ and on iteration j of loop 8–11 on iteration $k - j$ of loop 9–11, the assignment $t_{i+1}(F_k) := t_{i+1}(F_k) \cap c_\rho(t_{i+1}(F_j), t_{i+1}(F_l))$ in line 11. will be executed. Thus $c_\rho(T_P^{i+1}(G), T_P^{i+1}(H)) = c_\rho(t_{i+1}(F_j), t_{i+1}(F_l)) \in M_{11}$. Therefore $M_{11} = \{c_\rho(T_P^{i+1}(G), T_P^{i+1}(H)) | F_k = G \oplus_\rho H\}$. Then, by Lemma10.2 we get that $t_{i+1}(F_K) = t_i(F_k) \cap (\cap\{\mu | F_k : \mu \longleftarrow Body \in Fired_P(i+1)\}) \cap (\cap\{md_\rho(\mu) | (F \oplus_\rho G) : \mu \longleftarrow Body \in Fired_P(i+1)\}) \cap (\cap\{\mu | \mu \in M_{11}\}) = T_P^{i+1}(F_k)$.

This completes the proof of Proposition 5. ■

Now, from Proposition 1 and Lemma 9 it follows that t_{2m} represents $h_P = lfp(T_P)$. We only have to compute the time complexity of the algorithm. Loop 3–11 will be executed exactly $2m$ times. The running time of loop 3–7 is determined by the running time of the loop 8–11 which requires $O(M^2) = O((2sa^N)^2)$ steps. Combining together this yields the $O(2m(2sa^N)^2)$ bound.■

It is important to note that this theorem tells us that computing the least fixpoint of an HPP is *exponential* in the width of the largest formula of interest. In other words, if we were to develop an implementation of HPPs, and we required that no basic formulas of length greater than δ for some fixed δ are allowed, then the above theorem yields a polynomial result. This is a fairly reasonable assumption, in our opinion. This is stated in the following corollary.

Corollary 1 *Let P be any hp-program, and suppose δ is a fixed bound on the width of basic formulas occurring in P. Then h_P can be computed in polynomial time of size of*

P for all formulas of width $\leq \delta$.

Note: It is important to note that all the results of this subsection are valid for HPPs over p-strategies satisfying axioms (a)-(c) of Definition 3.2.1. Satisfaction of axiom (d) (Separation) is not required.

7.2 Complexity of Entailment

While it is important to know the complexity of computing the entire model of an HPP, it is really the entailment problem which gets solved over and over when queries are asked to the program. In this section we consider the complexity of entailment for HPPs: given a *consistent* program P and a query $F : \mu$, check whether $P \models F : \mu$. The problem of determining the consistency of an HPP is studied in Section 7.3.

In order to carry out our analysis of the entailment problem, we split this subsection into three parts based on the syntax of HPPs.

- In Section 7.2.1, we analyze what happens when we consider HPPs in the class HPP_1 — such HPPs only have atomic annotated atoms in rule heads, though the bodies may contain arbitrarily complex basic formulas.
- In Section 7.2.2, we analyze what happens when we consider HPPs in the class HPP_r for $r \geq 3$.
- In Section 7.2.3, we analyze what happens when we consider HPPs in the class HPP_2.

As usual we fix some standard encoding which is used to represent programs and queries. If P is an HPP, $|P|$ denotes the size of P's representation in this encoding, and similarly, if $F : \mu$ is an annotated basic formula, $|F : \mu|$ denotes the size of its representation. The complexity results in the sections that follow will be relative to $|P|$ and $|F : \mu|$.

7.2.1 Algorithms and Complexity for Entailment w.r.t. HPP_1

Below, we show that if we consider the class HPP_1 containing only atoms in rule heads, then we can specialize Algorithm LFP to a better algorithm, LFP_1 for computing the least fixpoint of T_P.

Algorithm 2 LFP_1

Input: $P \in HPP_1$, m – number of clauses in P, $F_1, \ldots, F_M$ - lexicographical enumeration of all formulas in P, $F = A_1 *_\rho \ldots *_\rho A_n$, $A_i \in B_L, i \in \{1, \ldots, n\}$.
Output: μ' - a subinterval of $[0, 1]$
BEGIN
(1) FOR $j = 1$ TO M DO $t(F_j) := [0; 1]$;
(2) FOR $i = 0$ TO $2m - 1$ DO
BEGIN
(3) FOR EVERY $C = G : \mu \leftarrow G_1 : \mu_1, ..., G_l : \mu_l \in P$ such that $t(G_j) \subseteq \mu_j$ for all $1 \leq j \leq l$ DO
BEGIN
(4) $t(G) := t(G) \cap \mu$;
(5) delete C from P
END
(6) FOR $k = 1$ TO M DO
(7) IF $F_k = B_1 *_\rho ... *_\rho B_r,\ r > 0, B_j$ is an atom for all $1 \leq j \leq r$
(8) THEN $t(F_k) := c_\rho(t(B_1), ..., t(B_r))$;
END;
(9) $\mu' := c_\rho(t(A_1), \ldots, t(A_n))$
END. (algorithm)

The following result specifies that the above algorithm may be directly used to check if an annotated basic formula $F : \mu$ is entailed by an HPP $P \in HPP_1$. Find the value μ' returned by Algorithm LFP_1 — if $\mu' \subseteq \mu$, then $P \models F : \mu$, else it does not.

Theorem 15 *For any program $P \in HPP_1$ and annotated basic formula $F : \mu$ the entailment problem "$P \models F : \mu$" is solvable in time $O(|P|^2 + |F : \mu|)$ via* Algorithm LFP_1.

Proof. Our algorithm takes as input $P \in HPP_1$ and F and produces as output $\mu' = h_P(F) = lfp(T_P)(F)$. Then by the definition of entailment, $P \models F : \mu$ iff $\mu' \subseteq \mu$. Suppose $F = A_1 *_\rho \ldots *_\rho A_n,\ n > 0$.

The algorithm proceeds in two steps. In the first step we use Algorithm LFP_1 to compute h_P for all atoms in P and F. In the second step, *if* $n > 1$, we compute $c_\rho(h_P(A_1), \ldots, h_P(A_n))$ and return the interval μ' obtained as the output (if $n = 1$ then F is equal to some atom A for which we have obtained the value of h_P on the first step. This value will be returned).

Now to compute h_P on the set of atoms, we use lines (1)–(8) of Algorithm LFP_1. Algorithm LFP_1 iteratively constructs table t of size M such that after i iterations of the main loop (1)–(8), $t(F_j) = T_P^i(F_j)$ for all $1 \leq j \leq M$.
The proof of correctness of algorithm $\mathbf{LFP}_1$ is analogous to the proof of correctness of the Algorithm LFP from Theorem 14. This proof uses the following proposition:

Proposition 6 *For any $P \in HPP_1$ and basic formula $F = A_1 *_\rho \ldots *_\rho A_n$*

1. *for all $i \geq 1$ $T_P^i(F) = c_\rho(T_P^i(A_1), \ldots, T_P^i(A_n))$.*
2. *$h_P(F) = c_\rho(h_P(A_1), \ldots, h_P(A_n))$.*

Proof. To prove part 1 of the proposition we notice that from Lemma 10.2 it follows that $T_P^{i+1}(F) = T_P^i \cap (\cap\{c_\rho(T_P^{i+1}(G), T_P^{i+1}(H)) | F = G \oplus_\rho H\})$. Then using double induction on i and on $width(F)$ and the properties of associativity and commutativity of c_ρ we show the required result.

Induction on i. Base case: $i = 0$.

$T^0_P(F) = [0,1] = c_\rho([0,1],\ldots,[0,1]) = c_\rho(T^0_P(A_1),\ldots,T^0_P(A_n))$.

Induction on i. Induction step.

Let $T^i_P(F) = c_\rho(T^i_P(A_1),\ldots,T^i_P(A_n))$ for all $i < k$. We show that the above equality holds for $i = k$.

We know that $T^k_P(F) = T^{k-1}_P(F) \cap (\cap\{c_\rho(T^k_P(G), T^k_P(H)) | F = G \oplus_\rho H\})$ (as $P \in HPP_1$).

We need to show that this is equal to $c_r ho(T^k_P(A_1),\ldots,T^k_P(A_n)$. We use induction on $n = width(F)$ to establish that.

Induction on $n = width(F)$. Base case: $n = 1$

If $n = 1$ then $F = A_1$ is an atom and therefore $T^k_P(F) = T^k_P(A_1) = c_\rho(T^k_P(A_1))$[1].

Induction on $n = width(F)$. Induction step.

Let $n > 1$ and for all $j < n$, and for all formulas $F' = A'_1 *_\rho \ldots *_\rho A'_j$ we have that $T^k_P(F') = c_\rho(T^k_P(A'_1),\ldots,T^k_P(A'_j))$.

We have: $T^k_P(F) = T^{k-1}_P(F) \cap (\cap\{c_\rho(T^k_P(G), T^k_P(H)) | F = G \oplus_\rho H\})$.

Now, for all G and H such that $F = G \oplus H$, $width(G) < width(F) = n$ and $width(H) < width(F) = n$. Let $G = B_1 *_\rho \ldots *_\rho B_r$ and $H = D_1 *_\rho \ldots *_\rho D_s$. We know that $r + s = n$ and $\{B_1,\ldots,B_r, D_1,\ldots D_s\} = \{A_1,\ldots,A_n\}$. By induction hypothesis $T^k_P(G) = c_\rho(T^k_P(B_1),\ldots T^k_P(B_r))$ and $T^k_P(H) = c_\rho(T^k_P(D_1),\ldots T^k_P(D_s))$. By associativity and commutativity

[1]Remember that we assumed that $c_\rho(\mu) = \mu$.

properties of c_ρ we have

$$c_\rho(T_P^k(G), T_P^k(H)) = c_\rho(c_\rho(T_P^k(B_1), \ldots T_P^k(B_r)), c_\rho(T_P^k(D_1), \ldots T_P^k(D_s)))$$

$$= c_\rho(T_P^k(B_1), \ldots T_P^k(B_r), T_P^k(D_1), \ldots T_P^k(D_s)) = c_\rho(T_P^k(A_1), \ldots T_P^k(A_n)).$$

Now we know that for any G and H, such that $F = G \oplus_\rho H$,

$$c_\rho(T_P^k(G), T_P^k(H)) = c_\rho(T_P^k(A_1), \ldots T_P^k(A_n)).$$

But then $\cap\{c_\rho(T_P^k(G), T_P^k(H)) | F = G \oplus_\rho H\}) = c_\rho(T_P^k(A_1), \ldots T_P^k(A_n))$.

By the induction hypothesis of the induction on i we know that $T_P^{k-1}(F) = c_\rho(T_P^{k-1}(A_1), \ldots, T_P^{k-1}(A_n))$.

Then, $T_P^k(F) = T_P^{k-1}(F) \cap (\cap\{c_\rho(T_P^k(G), T_P^k(H)) | F = G \oplus_\rho H\}) = c_\rho(T_P^{k-1}(A_1), \ldots, T_P^{k-1}(A_n) \cap c_\rho(T_P^k(A_1), \ldots T_P^k(A_n)) = c_\rho(T_P^k(A_1), \ldots, T_P^k(A_n))$ (the latter equality is due to monotonicity of T_P operator and monotonicity of c_ρ). This proves part 1 of the proposition.

Part 2 follows immediately from part 1. □

Proposition 6.2 also assures us that $\mu' = c_\rho(h_P(A_1), \ldots, h_P(A_n))$ computed in line (9) of the algorithm and returned as output is indeed equal to $h_P(F)$ which our algorithm claims to compute.

To evaluate the complexity of this algorithm we note that it is determined by the complexity of loop (1)–(8). Its running time is $O(2mM) \leq O(|P|^2)$ because $M \leq |P|$. The running time of line (9) and comparison of μ' and μ is $O(|F : \mu|)$. ■

7.2.2 Algorithms and Complexity for Entailment w.r.t. $HPP_r, r \geq 3$

In this section, we develop algorithms and provide complexity results for checking entailment when we consider hybrid probabilistic programs for HPP_r when $r \geq 3$. The main results in this section are : (i) a *nondeterministic* polynomial algorithm to compute entailment for arbitrary HPPs in $HPP_r, r \geq 3$, and (ii) proof that this problem is NP-complete.

We start our analysis by first considering the class HPP^0 of HPPs that only consist of facts, i.e. all rules in such HPPs have an empty body. Proposition 7 below is an important technical proposition that may be used to develop an algorithm for entailment in class HPP^0. This in turn may be used to show that entailment for arbitrary consistent HPPs (including HPPs not in class HPP^0) is in class NP.

Proposition 7 *Let* $P \in HPP^0$ *be a program consisting of* m *clauses* $C_1, ..., C_m$ *with empty bodies:* $C_k = H_k : [a_k, b_k],\ k = 1, ..., m$*. Then for every formula* $F = B_1 *_\rho ... *_\rho B_n,\ (n \geq 1,\ B_j$ *is an atom,* $1 \leq j \leq n)$*, and a number* $0 \leq x \leq 1$ *:*

1. $P \models F : [0, x]$ *iff there are* $k \geq 1$ *numbers* $x_1, ..., x_k$ *and a partition of* F *into* k *subformulas* $F = F_1 *_\rho ... *_\rho F_k$ *that:*

   (i) *each* F_i *is either some head* H_k *and* $x_i \geq b_k$*, or it is a subformula of some head* $H_k = F_i \oplus_\rho H'_k$ *and* $md_\rho([a_k, b_k]) \subseteq [0, x_i]$*, and*

   (ii) $c^2_\rho(x_1, ..., x_k) \leq x$.

2. $P \models F : [x, 1]$ *iff there are* k *numbers* $x_1, ..., x_k$ *and a partition of* F *into* k *subformulas* $F = F_1 *_\rho ... *_\rho F_k$ *that:*

(i) *each F_i is either some head H_k and $x_i \leq a_k$, or a subformula of some head $H_k = F_i \oplus_\rho H'_k$ and $md_\rho([a_k, b_k]) \subseteq [x_i, 1]$, and*

(ii) $c^1_\rho(x_1, ..., x_k) \geq x$.

Proof.

(1)If-part.

As $P \in HPP^0$, all it's clauses belong to $Fired_P(0)$ and $h_P(F) = T^1_P(F)$. Let given F $A = \{\mu | F : \mu \in P\}$, $B = \{md_\rho(\mu) | (F \oplus_\rho G) : \mu \in P\}$, and $C = \{c_\rho(h_P(G), h_P(H)) | F = G \oplus_\rho H\}$. We will use induction on width of F.

Base case. Let F be an atomic formula. By Lemma 10.1 we get $h_P(F) = [0,1] \cap \bigcap\{\mu | \mu \in A\} \cap \bigcap\{\mu | \mu \in B\}$. Then $h_P(F) = \mu_1 \cap \mu_2$ for some intervals $\mu_1, \mu_2 \in A \cup B \cup \{[0,1]\}$. Let $h_P(F) = [y, z]$. Assume without loss of generality that $\mu_1 = [y_1, z_1], \mu_2 = [y_2, z_2]$, and $z = min\{z_1, z_2\} = z_1$. Since $P \models F : [0, x]$ then $z_1 \leq x$. Now set $F_1 = F$ and $x_1 = z_1$. If $\mu_1 \in A$ then for some k $F_1 = F = H_k$ and $\mu_1 = [y_1, z_1] = [a_k, b_k]$. So, $x_1 \geq b_k$. If $\mu_1 \in B$ then for some k $H_k = F \oplus_\rho H'_k$ and $\mu_1 = md_\rho([a_k, b_k]) = [y_1, z_1] \subseteq [0, x_1]$. In the both cases condition (i) is satisfied. To establish (ii) it's enough to notice that $c^2(x_1) = x_1 = z_1 \leq x$.

Induction step. Suppose that for all formulas of width less than $n \geq 2$ Proposition 7 (1) is valid. Let $F = B_1 *_\rho ... * B_n,\ (n \geq 2)$.

By Lemma 10.2 we get $h_P(F) = [0,1] \cap \bigcap\{\mu | \mu \in A\} \cap \bigcap\{\mu | \mu \in B\} \cap \bigcap\{\mu | \mu \in C\}$. Then $h_P(F) = \mu_1 \cap \mu_2$ for some intervals $\mu_1, \mu_2 \in A \cup B \cup C \cup \{[0,1]\}$. Let $h_P(F) = [y, z]$. Assume without loss of generality that $\mu_1 = [y_1, z_1], \mu_2 = [y_2, z_2]$, and $z = min\{z_1, z_2\} = z_1$. Since $P \models F : [0, x]$ then $z_1 \leq x$. If $\mu_1 \in A \cup B$ we proceed as in the basic case. If $\mu_1 \in C$ then $F = G \oplus_\rho H$ for some formulas G, H of width less than n, and $\mu_1 = c_\rho(h_P(G), h_P(H))$. Let $h_P(G) = [y_G, z_G]$ and $h_P(H) = [y_H, z_H]$. Since $P \models F : [0, x]$ we deduce that $c_\rho(h_P(G), h_P(H)) = c_\rho([y_G, z_G], [y_H, z_H]) \subseteq$

$[0, x]$ and therefore $c_\rho^2(z_G, z_H) \leq x$. ¿From the definition of bounds z_G and z_H it follows that $P \models G : [0, z_G]$ and $P \models H : [0, z_H]$. Then by the induction assumption there exist such two sequence of bounds $u_1, ..., u_s$ and $v_1, ..., v_p$ and two partitions $G = G_1 *_\rho ... *_\rho G_s$ and $H = H_1 *_\rho ... *_\rho H_p$ which satisfy conditions (i) and (ii) of Proposition 7 (1). Uniting these sequences of bounds and partitions we get the sequence of bounds $x_1 = u_1, ..., x_s = u_s, x_{s+1} = v_1, ..., x_{s+p} = v_p$ of the length $k = s + p$ and partition of $F = G_1 *_\rho ... *_\rho G_s *_\rho H_1 *_\rho ... *_\rho H_p$ which satisfies condition (i). To prove (ii) we notice that $c_\rho^2(u_1, ..., u_s) \leq z_G$ and $c_\rho^2(v_1, ..., v_p) \leq z_H$ and due to monotonicity of c_ρ^2 we get $c_\rho^2(x_1, ..., x_k) = c_\rho^2(c_\rho^2(u_1, ..., u_s), c_\rho^2(v_1, ..., v_p)) \leq c^2(z_G, z_H) \leq x$.
The proof of **Only-If** part of proposition follows immediately from Lemma 10. It is enough to notice that from condition (i) it follows that for every F_j in the partition of F $h_P(F_j) \subseteq [0, x_j]$.

The proof of (2) is similar to the proof of (1). ■

Proposition 7 allows us to use the following nondeterministic algorithm to check entailment for hybrid probabilistic programs in class HPP^0.

Algorithm 3 Ent-HPP^0

Input: program $P \in HPP^0$, formula $F = B_1 *_\rho ... *_\rho B_n$, and an interval $[a, b]$.

1. Guess such $k \leq width(F)$, sequence of bounds $x_1, ..., x_k$ and partition $F = F_1 *_\rho ... *_\rho F_k$ which satisfy conditions (i) and (ii) of Proposition 7.(1) for $x = b$.
2. Guess such $k \leq width(F)$, sequence of bounds $x_1, ..., x_k$ and partition $F = F_1 *_\rho ... *_\rho F_k$ which satisfy conditions (i) and (ii) of Proposition 7.(2) for $x = a$.
3. If both attempts are successful then output “Yes”.

The following lemma uses Proposition 7 to establish correctness of Algorithm Ent-HPP^0 and its complexity.

Lemma 11 *(1) For any consistent hp-program $P \in HPP^0$ and any query $F : [a,b]$ algorithm Ent-HPP^0 output "Yes" iff $P \models F : [a,b]$.*

(2) Algorithm Ent-HPP^0 works in nondeterministic polynomial time.

Proof. Let $P \models F : [a,b]$. Then, $P \models F : [0,b]$ and $P \models F : [a,1]$. First step of Algorithm Ent-HPP^0 will then successfully establish that $P \models F : [0,b]$, due to Proposition 7.1, while the second step will successfully establish that $P \models F : [a,1]$ due to Proposition 7.2, i.e. Algorithm Ent-HPP^0 will succeed.

On the other hand if $P \not\models F : [a,b]$, then $h_P(F) \not\subseteq [a,b]$ and therefore either $h_P(F) \not\subseteq [0,b]$ or $h_P(F) \not\subseteq [a,1]$. In the first case, $P \not\models F : [0,b]$ and therefore, by Proposition 7.1, the first step of Algorithm Ent-HPP^0 will fail, and so will the algorithm itself. The case when $h_P(F) \not\subseteq [a,1]$ is symmetric.

Now we need to show that Algorithm Ent-HPP^0 is nondeterministically polynomial. To show this it is sufficient to demonstrate that checking that the selected k and $x_1,\ldots,x_k$ and the partition $F_1 *_\rho \ldots *_\rho F_k$ in step 1 satisfy the conditions of Proposition 7.1 can be done in polynomial in the size of the program time. The second step of Algorithm Ent-HPP^0 is symmetric to the first step.

For each F_i, $1 \leq i \leq k$, it takes time linear in the sum of the sizes of of clauses of all heads in P to establish if F_i satisfies condition (i) of Proposition 7.1. For each clause $H_i : [a_i, b_i]$, it takes linear time in the $|H| + |F_i|$ to check if $F_i = H$ or if $H = F_i \oplus_\rho G$ for some formula G. Once one of the matches is found, checking if $b_i \leq b$ is linear in $|x_i| + |b_i|$, and similarly, checking that $md_\rho([a_i, b_i]) \subseteq [0, x_i]$ is linear in $|a_i| + |b_i| + |x_i|$.

Finally, once the compliance of the partition to the condition (i) of Proposition 7.1 is established, checking the condition (ii) takes time polynomial in $|x_1| + \ldots + |x_k|$ as we assume that $c^2_\rho(x_1,\ldots,x_k)$ is polynomially computable. ■.

Let $P \in HPP$ consist of m clauses $C_1, ..., C_m$ and $C_j = H_j : \mu_j \longleftarrow body_j$. For

$0 \leq i \leq 2m$ we define an hp-program $P_i \in HPP^0$ as $P_i = \{H_j : \mu_j | act(C_j) \leq i\}$. The following important lemma shows us that it is possible to reduce the entailment problem for all HPPs to the entailment problem for the subclass HPP^0.

Lemma 12 *(1) For every $0 \leq i \leq 2m-1$ and any formula F $P_i \models F : T_P^{i+1}(F)$.*

(2) For every $0 \leq i \leq 2m$ and any clause $C_j = (H_j : \mu_j \longleftarrow F_1 : \nu_1 \wedge ... \wedge F_r : \nu_r) \in P$, $C_j \in Fired(i+1)$ iff $P_i \models F_l : \nu_l$ for all $l = 1, ..., r$.

(3) For any query $F : \mu$

$P \models F : \mu$ iff $P_{2m} \models F : \mu$.

Proof. The proof of point (1) follows directly from Lemma 10. Points (2) and (3) follow from (1). ■

We are now ready to present a generic algorithm, Algorithm Ent-HPP, that computes entailment by HPPs.

Algorithm 4 Ent-HPP
Input: program $P \in HPP$, consisting of m clauses $C_1, ..., C_m$ of the form $C_j = H_j : \mu_j \longleftarrow F_1^j : \nu_1 \wedge ... \wedge F_{r_j}^j$, formula $F = B_1 *_\rho ... *_\rho B_n$, and an interval $[a, b]$.
(1) $P_0 := \{H_j : \mu_j | body\ of\ C_j\ is\ empty\}$;
(2) FOR $i = 1$ TO $2m$ DO
  (3) guess $Fired(i)$;
(4) FOR EACH $C_j \in Fired_P(i)$ DO
  (5) FOR $l = 1$ TO r_j DO
    (6) Call Ent-$HPP^0(P_i, F_l^j : \nu_l)$;
  (7) END_DO
  (8) END_DO
  (9) $P_i := P_{i-1} \cup \{H_j : \mu_j | C_j \in Fired(i)\}$
(10) END_DO
(11) Call Ent-$HPP^0(P_{2m}, F : [a, b])$
(12) Output "Yes" if all (nondeterministic) calls of Ent-HPP^0 were successful.

The following result establishes that Algorithm Ent-HPP correctly computes entailment in nondeterministic polynomial time.

Lemma 13 *Algorithm Ent-HPP determines nondeterministically if $P \models F : [a, b]$ in polynomial time.*

Proof. The proof of correctness of the algorithm follows from Lemmas 11 and 12. If for all i, $Fired(i)$ is correctly guessed in line (3), then the loop in lines (4)-(8) can (non-deterministically) confirm this guess. If so, then in line (9), the set P_i is defined correctly. Then after finishing the loop in lines (2)-(10), set P_{2m} consists of the heads of all fired clauses and the invocation of Ent-HPP^0 in line (11) can correctly establish if $P_{2m} \models F : [a, b]$ which (by Lemma 12.3) yields the required result.

To evaluate complexity of the algorithm we notice that the total number of calls of Algorithm Ent-HPP^0 in line (6) does not exceed the number of formulas in bodies of all clauses of P. Let $t_{Ent-HPP^0}(P)$ denote the maximal time among all calls of Ent-HPP^0. Then the length $t_{Ent-HPP}(P, F : [a, b])$ of the shortest successful computation of algorithm Ent-HPP on input $(P, F : [a, b])$ can be bounded as $t_{Ent-HPP}(P, F : [a, b]) \leq O(|P| t_{Ent-HPP^0} + |F : [a, b]|) \leq pol(|P| + |F : [a, b])$ for some polynomial $pol(N)$. ■

An immediate consequence of the above result is the claim that the entailment problem for HPPs is in **NP**.

Corollary 2 *The entailment problem for the class of consistent HPPs belongs to* ***NP****.*

The following result shows that the problem of entailment in HPPs is **NP-complete** even when we only consider head-widths of size 3, and rule bodies are forced to be empty (which occurs when we set body-widths to 0).

Theorem 16 *The entailment problem is* ***NP-complete*** *for the class $HPP_{3,0}$.*

Proof. Upper bound follows by Theorem 2.

To prove the lower bound we show that well-known NP-complete problem 3-Dimensional Matching (3-DM) is reducible in polynomial time to the entailment problem for class $HPP_{3,0}$. Let I be any instance of 3-dimensional matching problem, consisting of three sets W, X and Y of equal size q and a set of triples $M \subseteq W \times X \times Y$. I $\in$3-DM iff there is a complete matching, i.e. such subset $M' \subseteq M$ of size q that $W = \{w | \langle w, x, y\rangle \in M',\ X = \{x | \langle w, x, y\rangle \in M'\}$ and $Y = \{y | \langle w, x, y\rangle \in M'\}$. We use I to construct HPP-program P and query $F : \mu$ such that $I \in$ 3-DM iff $P \models F : \mu$.

As a strategy we choose the conjunctive independence strategy, with c_{inc} defined as follows: $c_{inc}([a, b], [c, d]) = [ac, bd]$. Program P uses atoms of the set $A = W \cup X \cup Y$. For every $\langle w, x, y\rangle \in M$ we include in P the clause:

$w \wedge_{inc} x \wedge_{inc} y : [0; 0.5] \leftarrow$.

Let $F : \mu = \bigwedge_{inc}\{a | a \in A\} : [0, 0.5^q]$.

It is easy to check that for every atom $a \in A$ $h_P(a) = [0, 1]$, for every pair of atoms (a, b) $h_P(a \wedge_{inc} b) = [0, 1]$ and for every triple (a, b, c) which does not constitute the head of some clause in P $h_P(a \wedge_{inc} b \wedge_{inc} c) = [0, 1]$. Of course, if $w \wedge_{inc} x \wedge_{inc} y$ is the head of some clause of P then $h_P(w \wedge_{inc} x \wedge_{inc} y) = [0, 0.5]$. Suppose now that I $\in$3-DM and M' is its complete matching. Then F can be partitioned by q triples corresponding to the heads of clauses of P:

$F = \bigwedge_{inc}(w \wedge_{inc} x \wedge_{inc} y | \langle w, x, y\rangle \in M'\}$.

This implies that $h_P(F) \subseteq c_i([0, 0.5], ..., [0, 0.5]) = [0, 0.5^q]$, and $P \models F : [0, 0.5^q]$.

On the other hand, if$P \models F : [0, 0.5^q]$ then $h_P(F) \subseteq [0, 0.5^q]$. It is possible only if F can be partitioned into q subformulas: $F = F_1 \wedge_{inc} \ldots \wedge_{inc} F_q$ such that each $F_i, 1 \leq i \leq q$, is a head of some clause of P. Indeed, any other partition of F which includes $k < q$ heads of clauses of P provides interval $[0, 0.5^k]$. But $[0, 0.5^k] \not\subseteq [0, 0.5^q]$.

Then the set $M' = \{\langle w_i, x_i, y_i\rangle | F_i = w_i \wedge_{inc} x_i \wedge_{inc} y_i, i = 1, ..., q\}$ is a complete matching and I $\in$3-DM. ■

The proof of Theorem 16 allows us to immediately infer the following result.

Corollary 3 *The entailment problem is* **NP-complete** *for the classes* HPP *and* $HPP_{k,r}$ $(k \geq 3)$.

7.2.3 Algorithms and Complexity for Entailment w.r.t. HPP_2

So far we have have shown that entailment is polynomial for hp-programs in HPP_1 and is NP-complete for hp-programs in HPP_k, $k \geq 3$. We now turn our attention to HPP_2. Here, our results are most interesting — it turns out that for many different types of p-strategies, the entailment problem is polynomially solvable, though the general case of this problem is open.

Recall that given a graph $G = (V, E)$, a *matching*[70] is a set $E' \subseteq E$ such that no two edges in E' share a common vertex. A matching E' is *maximal* iff every edge in $(E - E')$ shares a vertex with some edge in E'. If $V = 2m$, we say a matching E is *complete* iff every vertex $v \in V$ is the endpoint of some edge $e_v \in E'$. It will turn out that entailment is polynomial time equivalent to the following *generalized matching problem* on general (complete) graphs.

Definition 7.2.1 (Generalized weighted matching problem) *Given an edge-weighted, undirected graph* $G = \langle V, E, w \rangle$ *and a goal weight combination function* c*, the generalized weighted matching problem on* G *is a problem of finding 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:
$GWM_{max}(c) = \{G = \langle V, E, w : E \to [0,1] \rangle, B \in [0,1] || V| = 2m\}$ *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*
$GWM_{min}(c) = \{G = \langle V, E, w : E \to [0,1] \rangle, B \in [0,1] || V| = 2m\}$ *and there exists a complete matching* $\{e_1, \ldots, e_m\} \subseteq E$ *such that* $c(w(e_1), \ldots, w(e_m)) \leq B\}$.

To show that entailment in HPPs is polynomial-time equivalent to the above generalized matching problem, we need to show that each problem can be polynomially reduced to the other.

Reducing Entailment to Generalized weighted matching.

In order to reduce the entailment problem, we can use the composition function $c = \langle c^1, c^2 \rangle$ to create a generalized weighted matching which uses the c^1, c^2 as goal weight combination functions. Remember, that since p-strategies are commutative and associative, we can consider them to be functions of any number of arguments.

Theorem 17 *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 $c^1 \in C^1, c^2 \in C^2$.*

Proof. Due to Lemma 12 any problem $''P \models F : \mu\ ?''$ for $P \in HPP_2$ can be reduced in polynomial time to polynomial number of entailment problems of the form $''P' \models F' : \mu'\ ?''$ for programs $P' \in HPP_{2,0}$. So we will suppose that $P \in HPP_{2,0}$.

We will prove the theorem for $F = a_1 \wedge_\rho ... \wedge_\rho a_n$, where $a_i \in B_L$ for $i = 1, ..., n$. The proof for formulas $F = a_1 \vee_\rho ... \vee_\rho a_n$, will be analogous. Without loss of generality one can assume that the width of F is even, i.e. $n = 2m$ (otherwise one can add a new true fact t in the query and in P: $P \models F : \mu \iff P \cup \{t : [1, 1]\} \models F \wedge_\rho t : \mu$).

Using Algorithm LFP of theorem 14 we compute (in polynomial time) the least fixpoint $h_P(a_i \wedge_\rho a_j) = \mu_{i,j} = [a_{i,j}, b_{i,j}]$ for any pair $1 \leq i < j \leq 2m$. Now, we define a new program $P' \in HPP_{2,0}$ consisting of $2m(2m - 1)$ clauses of the form $a_i \wedge_\rho a_j : \mu_{i,j}$. From proposition 7 it follows that $P \models F : \mu \iff P' \models F : \mu$. Now let $c_\rho = \langle c^1_\rho, c^2_\rho \rangle$. Define undirected graph $G = G(P', F) = (V, E)$ as follows:

$V = \{a_1, ..., a_{2m}\}, E = \{(a_i, a_j) | 1 \leq i < j \leq 2m\}$, and attach to it two weight functions: $w_1(a_i, a_j) = a_{i,j}$ and $w_2(a_i, a_j) = b_{i,j}$. Then the theorem follows from

Proposition 8 $P' \models F : [a, b] \iff \{G, w_1, a\} \in GWM_{max}(c^1_\rho)$ *and* $\{G, w_2, b\} \in GWM_{min}(c^2_\rho)$.

Proof.

$\Longrightarrow$. If $P' \models F : [a, b]$ then by proposition 7 there are two partitions of F into heads (or subformulas of heads) of clauses of P': $F = G_1 \wedge_\rho ... \wedge_\rho G_k = H_1 \wedge_\rho ... \wedge_\rho H_l$ such that $c^1_\rho(h_{P'}(G_1), ..., h_{P'}(G_k)) \geq a$ and $c^2_\rho(h_{P'}(H_1), ..., h_{P'}(H_l)) \leq b$ (where for any formula E we set $h_{P'}(E) = [h^1_{P'}(E), h^2_{P'}(E)]$). Now we can assume that $k = l = m$ and all G_i's and H_j's are the heads of clauses of P'. Indeed, if e.g. $G_1 = a_i$ and $G_2 = a_j$ then we can combine them into the head $a_i \wedge_\rho a_j$ because $c_\rho(h_{P'}(a_i), h_{P'}(a_j)) \subseteq h_{P'}(a_i \wedge_\rho a_j) = \mu_{i,j}$ and therefore $c^1_\rho(h^1_{P'}(a_i), h^1_{P'}(a_j)) \leq a_{i,j}$. Then it is easy to check that the first of these partitions $\{G_1, ..., G_m\}$ corresponds to the complete matching $\{e_p = (a_i, a_j) |$ $G_p = a_i \wedge_\rho a_j,\ 1 \leq p \leq m\}$ which solves the matching problem $\{G, w_1, a\} \in GWM_{max}(c^1_\rho)$, and the second partition $\{H_1, ..., H_m\}$ corresponds to the complete matching $\{e'_p = (a_i, a_j) | G_p = a_i \wedge_\rho a_j,\ 1 \leq p \leq m\}$ which solves the matching problem $\{G, w_2, b\} \in GWM_{min}(c^2_\rho)$.

$\Longleftarrow$.

It is easy to see that two complete matchings for problems $\{G, w_1, a\} \in GWM_{max}(c^1_\rho)$ and $\{G, w_2, b\} \in GWM_{min}(c^2_\rho)$ provides two partitions of F which satisfy conditions of (1) and (2) of proposition 7. Then by this proposition $P' \models F : [a, b]$. ■

The above theorem states that any entailment problem for $HPP_{2,0}$ can be reduced to a generalized weighted matching problem. Conversely, any generalized weighted matching problem whose associated goal function satisfies axioms (a)-(d) of Definition 3.2.1 may be polynomially reduced to an equivalent entailment problems for hp-programs of

$HPP_{2,0}$.

Theorem 18 *Any generalized weighted matching problem for goal functions, satisfying axioms (a)-(d) of Definition 3.2.1, is reducible in polynomial time to entailment problems for hp-programs of* $HPP_{2,0}$.

Proof. Suppose that we have a problem of class $GWM_{max}(c^1)$. Then given a graph $G = \langle V, E\rangle$, weight function $w : E \to [0, 1]$, and a goal function c^1, one can define the composition function $c_\rho = (c^1, min)$ and construct the hp-program $P = P(G, w)$ which for every edge $e = (v_i, v_j) \in E$ includes the clause $v_i \wedge_\rho v_j : [w(e), 1]$. Then there exists a complete matching $\{e_1, ..., e_m\} \subseteq E$ such that $c^1(w(e_1), ..., w(e_m)) \geq B$ iff $P \models v_1 \wedge_\rho ... \wedge_\rho v_2m : [B, 1]$.

Analogously, one can reduce a problem of $GWM_{min}(c^2)$. ■

It is well-known [70] that the weighted matching problem is solvable in polynomial time for the sum of edge weights. This allows us to get effective algorithms for almost all of strategies considered in [22].

Corollary 4 *The entailment problem for the class of* HPP_2 *programs over strategies* $\mathcal{S} = \{inc, igc, pcc, ncc, ind, igd, pcd, ncd\}$ *is solvable in polynomial time.*

Let us show the intuition behind this on the example of conjunctive independence strategy inc[2]. As we know $c_{inc}([a, b], [c, d]) = [ac, bd]$, i.e., $c^1_{inc}(x, y) = c^2_{inc}(x, y) = xy$. Then we can reduce $GWM_{max}(c^1_\rho)$ to a weighted matching problem $WM_{max} = (V, E, w')$ as follows. The graph (V, E) of WM_{max} will be the same as the graph of $GWM_{max}(c^1_\rho)$. For each edge $e \in E$, let $w'(e) = \log_2(w(e))$, where $w(e)$ is the weight

[2] The reduction of the GWM problem for functions found in the p-strategies $inc, igc, pcc, ncc, ind, igd, pcd, ncd$ to classical weighted matching problem had been suggested to us by Samir Khuller.

of edge e in $GWM_{max}(c^1_\rho)$. Let $\{e_1, \ldots, e_k\}$ be the solution of WM_{max}. Then we know that $w'(e_1) + \ldots + w'(e_k) = \log_2(w(e_1)) + \ldots + \log_2(w(e_k)) = \log_2(w(e_1) \cdot w(e_2) \cdot \ldots \cdot w(e_k))$ is the maximal weight of a matching in for (V, E, w'). Now, since $\log_2$ is a monotonic function, it follows that $w(e_1) \cdot w(e_2) \cdot \ldots \cdot w(e_k) = c^1_{ind}(w(e_1), \ldots, w(e_k))$ is the maximal value of a matching for (V, E, w), i.e $\{e_1, \ldots, e_k\}$ is the solution of GWM_{max} problem[3]. Reductions for other functions follow similar intuition.

The above result is interesting because it provides polynomial results for programs in HPP_2 for all composition strategies studied in [22] — in fact, we have been unable to find a single composition strategy for which entailment for the class of HPP_2 programs is not polynomial. This leads to an interesting open question.

Open question. Is there polynomial time computable composition function $c = (c^1, c^2)$ satisfying axioms (a)-(d) for which generalized matching problem $GWM_{min}(c^2)$ ($GWM_{max}(c^1)$) is NP-complete? If so, we would know that there are some polynomially computable composition functions for which the entailment problem for programs in HPP_2 is NP-complete.

7.3 Complexity of Consistency

7.3.1 Algorithms and Complexity for Consistency w.r.t. HPP_1

It is easy to see that even a simple HPP_1 program containing two simple facts, viz. $a : [0, 0]$, $a : [1, 1]$, is inconsistent. The complicated interactions between logic and probabilities can engender more devious inconsistencies in HPPs.

[3]The above argument works under the assumption that all weights supplied by w function are non-zero. However, a slight modification of the reduction, which throws zero-weight edges out of the graph for weighted matching problem will work.

The following result tells us that to check if P is consistent, if our language allows n ground atoms, we need to create only all ground basic formulas F containing all the n atoms and check if $h_P(F) \neq \emptyset$ for them. If so, P is guaranteed to be consistent.

Lemma 14 *Let $P \in HPP$ over set $\mathcal{S} = \{\rho_1, \ldots, \rho_m\}$ of p-strategies and let $\mathcal{A} = \{A_1, \ldots, A_n\}$ be all the atoms found in P. Then: P is consistent iff for all formulas F_i of form $F_i = A_1 *_{\rho_i} \ldots *_{\rho_i} A_n$, $i = 1, \ldots, m$, $h_P(F_i) \neq \emptyset$.*

Proof.

If P is consistent then for all basic formulas F (including formulas of the form $A_1 *_{\rho_i} \ldots *_{\rho_i} A_n$, $h_P(F) \neq \emptyset$.

If P is inconsistent then there exists a formula $F = B_1 *_{\rho} \ldots *_{\rho} B_k$ such that $h_P(F) = \emptyset$. But since $\{B_1, \ldots, B_k\} \subseteq \{A_1, \ldots, A_n\}$, $h_P(A_1 *_{\rho} \ldots *_{\rho} A_n) = \emptyset$. This proves the lemma.■

However, for programs from the class HPP_1, the situation with checking consistency is much simpler. As it turns out, for consistency, it is sufficient to check only formulas of length 1, i.e. atoms.

Theorem 19 *Given a program $P \in HPP_1$ its consistency can be established in polynomial time.*

The proof is based on the following lemma which says that when a program $P \in HPP_1$ is inconsistent, then there exists a ground atom A such that $h_P(A) = \emptyset$.

Lemma 15 *Let $P \in HPP_1$. If P is inconsistent, then there exists an atom $A \in B_L$ such that*

1. *A is the head of at least one rule in P and*

2. $h_P(A) = \emptyset$.

Proof. Since P is inconsistent, there exists some formula $F = A_1 *_\rho \ldots *_\rho A_k$ such that $h_P(F) = \emptyset$. If $k = 1$ then $F = A_1$ and the lemma is true.

If $k > 1$ then by Proposition 6 $h_P(F) = c_\rho(h_P(A_1), \ldots, h_P(A_k))$. But the result of c_ρ can be an empty set only if one of the arguments is an empty set. Therefore, for some A_i, $1 \leq i \leq k$, $h_P(A_i) = \emptyset$). ■

Using the lemma, we can now use a result about Algorithm LFP_1 from Theorem 15 to check consistency.

Algorithm 5 Consistency_Check_for_HPP_1

- Run Algorithm LFP_1 until $t(A) = \emptyset$ for some $A \in B_L$. If such an A exists, then P is inconsistent.
- If Algorithm LFP_1 finished successfully, then P is consistent.■

7.3.2 Algorithms and Complexity for Consistency w.r.t. HPP_2

It would be nice if a result similar to Lemma 15 held when we consider programs with complex formulas in the heads. However, the following example shows that even for $P \in HPP_{2,0}$ the shortest inconsistent formula can be longer than the longest head of the rule.

Example 7.3.1 Let P be

$$a \wedge_{ind} b : [0.5, 0.7] \longleftarrow \quad c \wedge_{ind} d : [0.3, 0.48] \longleftarrow$$
$$a \wedge_{ind} c : [0.6, 1] \longleftarrow \quad b \wedge_{ind} d : [0.8, 1] \longleftarrow$$

We can easily compute $h_P = lfp(T_P)$:

$$h_P(a) = [0.6, 1] \qquad h_P(b) = [0.8, 1]$$

$$h_P(c) = [0.6, 1] \qquad h_P(d) = [0.8, 1]$$

$$h_P(a \wedge_{ind} b) = [0.5, 0.7] \qquad h_P(a \wedge_{ind} c) = [0.6, 1]$$

$$h_P(a \wedge_{ind} d) = [0.4, 1] \qquad h_P(b \wedge_{ind} c) = [0.48, 1]$$

$$h_P(b \wedge_{ind} d) = [0.8, 1] \qquad h_P(c \wedge_{ind} d) = [0.48, 0.48]$$

However, $h_P(a \wedge_{ind} b \wedge_{ind} c \wedge_{ind} d) \subseteq c_{ind}(h_P(a \wedge_{ind} b), h_P(c \wedge_{ind} d)) \cap c_{ind}(h_P(a \wedge_{ind} c), h_P(b \wedge_{ind} d)) = [0.5*0.48, 0.7*0.48] \cap [0.6*0.8, 1*1] = [0.24, 0.336] \cap [0.48, 1] = \emptyset$.
$\square$

The following result tells us that the inconsistency problem for HPP_2 is polynomially reducible to the generalized weighted matching problem.

Theorem 20 *Inconsistency problem for HPP_2 is polynomially reducible to $GWM_{\min}$ and $GWM_{\max}$.*

Proof. As in the proof of theorem 17 lemma 12 allows us to reduce the consistency problem for HPP_2 to the consistency problem for $HPP_{2,0}$.

Let now $P \in HPP_{2,0}$ be a program over set $\mathcal{S}$ of p-strategies that uses atoms from the set $\mathcal{A} = \{A_1, \ldots, A_{2m}\}$ and only those. By lemma 14 it is sufficient to check the consistency of all formulas of form $F = A_1 *_\rho \ldots *_\rho A_{2m}$, $\rho \in \mathcal{S}$. As in theorem 17 we construct (in polynomial time) program $P' \in HPP_{2,0}$ which consists of all possible clauses of the form $A_i *_\rho A_j : h_P(A_i *_\rho A_j) \longleftarrow$. From proposition 7 it follows that P is inconsistent iff for some strategy $\rho \in \mathcal{S}$ for formula $F = A_1 *_\rho \ldots *_\rho A_{2m}$ there exist two partitions of F into subformulas $F = G_1 *_\rho \ldots *_\rho G_m$ and $F = H_1 *_\rho \ldots *_\rho H_m$ where all $G_1, \ldots, G_m$ and $H_1, \ldots, H_m$ are heads of clauses in P', such that there exist such numbers $0 \leq b < a \leq 1$ that

$c_\rho(h_P(G_1), \ldots h_P(G_m)) \subseteq [0, b]$ and

$c_\rho(h_P(H_1), \ldots h_P(H_m)) \subseteq [a, 1]$.

To verify the existence of such partitions it is sufficient to optimize two General Weighted Matching Problems. Indeed, let graph $G(P', F) = \langle V, E \rangle$ and two weight functions w_1 and w_2 be defined as in theorem 17. Then using standard dichotomy technique one can find such minimal b' and maximal a' that $\{G, w_2, b'\} \in GWM_{min}(c_\rho^2)$ and $\{G, w_1, a'\} \in GWM_{max}(c_\rho^1)$ and check if $b' < a'$. ■

This is an important result, because we know that polynomial algorithms exist for many generalized weighted matching problems. We may therefore immediately state the following corollary.

Corollary 5 *Let P be any HPP_2 program over the set of p-strategies $\{inc, igc, pcc, igd, ind, pcd, ncd\}$. Then the consistency problem for P is solvable in polynomial time.*

7.3.3 Algorithms and Complexity for Consistency w.r.t. HPP_r, $r \geq 3$

We are now ready to examine the general consistency problem. Checking if an arbitrary HPP is inconsistent may be performed by the nondeterministic algorithm InCon below.

Algorithm 6 InCon
Input. An arbitrary HPP P.

1. Guess the shortest "inconsistent" formula F.
2. Guess two partitions of F into $G_1 *_\rho \ldots *_\rho G_m$ and $H_1 *_\rho \ldots *_\rho H_k$ two sets of numbers: $x_1, \ldots, x_m$ and $y_1, \ldots, y_k$, $0 \leq x_i, y_j \leq 1$ such that $c_\rho^1(x_1, \ldots, x_m) > c_\rho^2(y_1, \ldots y_k)$.
3. Using Algorithm Ent-HPP check that for all $i \in \{1, \ldots m\}$ $P \models G_i : [0, x_i]$ and all $j \in \{1, \ldots k\}$ $P \models H_j : [y_j, 1]$.

4. If all calls to Algorithm Ent-HPP of previous step are successful then output "Yes".

The following result shows that for arbitrary HPPs, the problem of checking if an HPP is inconsistent is NP-complete.

Theorem 21 *The inconsistency problem for HPP is* ***NP-complete****.*

Proof. Algorithm InCon is easily seen to be in NP.

To prove that inconsistency problem is NP-hard we notice that entailment problem for consistent HPPs is reducible in polynomial time to the inconsistency problem. To see why, suppose P is consistent. Then it is easy to check that for any annotated formula $F : \mu$ $P \models F : \mu$ iff the following program P' is inconsistent. $P' = P \cup \{z : [0; 0.1] \leftarrow F : \mu\} \cup \{z : [0.2; 0.3] \leftarrow F : \mu\}$.

Now it follows from Theorem 16 (on NP-completeness of the entailment problem) that inconsistency problem is NP-hard even for class $HPP_{3,0}$. ■

An immediate corollary is that the consistency problem for arbitrary HPPs is co-NP-complete.

Corollary 6 *Consistency problem for HPP is* ***co-NP-complete****.*

7.4 Proof Complexity of HPPs

In this section, we study the problem of proof complexity of Hybrid Probabilistic Programs. This problem can be formulated as follows: given a sound and complete inference system for and HPP program P and a basic formula $F : \mu$, such that $P \models F : \mu$, what is the "size" of the derivation of $P \vdash F : \mu$?

We start by presenting a sound and complete inference system HGR_P. This system is very similar to the HR_P inference system presented in Section 5.3. The two differences between the proof systems are

(i) The newly introduced system HGR_P assumes that P is a ground program.

(ii) HR_P refutations derive hp-clauses which follow from P, while HGR_P refutations will derive basic formulas entailed from P.

After presenting this proof system and showing its soundness and completeness we will show that all valid inferences in HGR_P can be made with proofs that are polynomially bounded in length ! We should note that this is consistent with the preceding NP-completeness result for the entailment problem because the search space may involve exponentially many derivations each of polynomially bounded length. This is an interesting and counterintuitive result — it says that (the answers to) all queries to HPPs have at least one polynomial explanation. This is not true of classical logic proof systems [106] as well as proof systems for most nonmonotonic logics.

7.4.1 System HGR_P

Let us now define the axioms and inference rules of the proof system, HGR_P.

Definition 7.4.1 *Let P be a ground hp-program over set $\mathcal{S}$ of p-strategies. We define the formal system HGR_P as follows:*

1. ***Axioms** of HGR_P are all expressions of the form:*

$$\overline{A : [0, 1]}$$

   where $A \in B_L$.

2. ***Inference Rules**. There are 7 types of inference rule schemas in HGR_P. One type (Program) depends on the clauses of program P while other 6 types of inference rule schemas are independent of clauses in P but do depend on which p-strategies are in $\mathcal{S}$.*

- *Program:* *Let* $F : \mu \leftarrow G_1 : \mu_1, ..., G_k : \mu_k \in P$,

$$\frac{G_1 : \mu_1 \;\; ... \; G_k : \mu_k}{F : \mu}$$

Note: rules corresponding to clauses with empty body $(k = 0)$ *are actually axioms.*

- *A-Composition:* *Let* $A_1, A_2 \in B_L$,$\rho \in \mathcal{S}$

$$\frac{A_1 : \mu_1 \quad\; A_2 : \mu_2}{(A_1 *_\rho A_2) : c_\rho(\mu_1, \mu_2)}$$

- *F-Composition:* *Let* $A_1, \ldots A_k, B_1 \ldots B_k \in B_L$,$\rho \in \mathcal{S}$

$$\frac{(A_1 *_\rho \ldots *_\rho A_k) : \mu_1 \quad\; (B_1 *_\rho \ldots *_\rho B_l) : \mu_2}{(A_1 *_\rho \ldots *_\rho A_k *_\rho B_1 *_\rho \ldots *_\rho B_l) : c_\rho(\mu_1, \mu_2)}$$

- *Decomposition (cut):* *Let* $\rho \in \mathcal{S}$

$$\frac{(F \oplus_\rho G) : \mu}{F : md_\rho(\mu)}$$

- *Clarification:*

$$\frac{F : \mu_1 \quad\; F : \mu_2}{F : \mu_1 \cap \mu_2}$$

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

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

- *Interval Weakening:*

$$\frac{F : \mu \quad\;\; \mu \subseteq \mu_1}{F : \mu_1}$$

3. *A finite sequence* $C_1 \ldots C_r$ *of annotated formulas is called an* HGR_P*-derivation* iff *each formula* $C_j = F_j : \mu_j$ *can be deduced from zero (in the case of axiom), one, or more previous of* $C_1 \ldots C_{j-1}$ *by applying one of the* inference rules *to them. We call formula* C_r the result *of the* HGR_P*-derivation.*

4. *An annotated formula $C = F : \mu$ is derivable in HGR_P iff there exists such an HGR_P-derivation $C_1, \dots C_r$ that $C_r = C$. We denote it by $P \vdash_{HGR_P} C$,or just by $P \vdash C$ in the absence of other inference systems.*

The following theorem states that HGR_P is a sound inference system.

Theorem 22 (soundness of HGR_P) *Let P be an hp-program and let Q be a an hp-formula. If $P \vdash_{HGR_P} Q$, then $P \models Q$.*

Proof.

We prove soundness of the system HGR_P by showing that every axiom and rule of the system preserves truth.

Let h be any model of program P, i.e. $h \models P$.

- Axioms. Let $A \in B_L$. By definition of a formula function $h(A) \subseteq [0, 1]$, therefore, $h \models A : [0, 1]$.

- Program. Let $F : \mu \longleftarrow G_1 : \mu_1 \wedge \dots \wedge G_k : \mu_k$ be a rule of P. Let $h \models G_1 : \mu_1, \dots, h \models G_k : \mu_k$. Then, $h \models G_1 : \mu_1 \wedge \dots \wedge G_k : \mu_k$. Since $h \models F : \mu \longleftarrow G_1 : \mu_1 \wedge \dots \wedge G_k : \mu_k$, by definition of satisfaction we conclude, $h \models F : \mu$.

- A-Composition. Let $A_1, A_2 \in B_L$ and $h \models A_1 : \mu_1$ and $h \models A_2 : \mu_2$. Let ρ be some p-strategy from $\mathcal{S}$. Then, by definition of a formula function, $h(A_1 *_\rho A_2) \subseteq c_\rho(h(A_1), h(A_2))$. By the Monotonicity property of composition functions, and as $h(A_1) \subseteq \mu_1$ and $h(A_2) \subseteq \mu_2$, $h(A_1 *_\rho A_2) \subseteq c_\rho(h(A_1), h(A_2)) \subseteq c_\rho(\mu_1, \mu_2)$ and therefore, $h \models (A_1 *_r hoA_2) : c_\rho(\mu_1, \mu_2)$.

- F-Composition. Let $A_1, \dots A_, B_1 \dots B_l \in B_L$ and $h \models (A_1 *_\rho \dots *_\rho A_k) : \mu_1$ and $h \models B_1 *_r ho \dots *_\rho B_l : \mu_2$ for some p-strategy $\rho \in \mathcal{S}$.

Then, by definition of a formula function, $h(A_1 *_\rho \ldots *_\rho A_k *_\rho B_1 *_\rho \ldots *_\rho B_l) \subseteq c_\rho(h(A_1 *_\rho \ldots *_\rho A_k), h(B_1 *_\rho \ldots *_\rho B_l))$. By the Monotonicity property of composition functions, and since $h(A_1 *_\rho \ldots *_\rho A_k) \subseteq \mu_1$ and $h(B_1 *_\rho \ldots *_\rho B_l) \subseteq \mu_2$ we get $h(A_1 *_\rho \ldots *_\rho A_k *_\rho B_1 *_\rho \ldots *_\rho B_l) \subseteq c_\rho(\mu_1, \mu_2)$, i.e., $h \models (A_1 *_\rho \ldots *_\rho A_k *_\rho B_1 *_\rho \ldots *_\rho B_l) : c_\rho(\mu_1, \mu_2)$.

- Decomposition. Let $h \models H : \mu$ and $H = F \oplus_\rho G$. By the Commutativity property of formula functions, $h(H) = h(F *_\rho G) \subseteq \mu$. Then, by the Decomposition property of formula functions, $h(F) \subseteq md_\rho(h(F *_\rho G))$. Since $h(F *_\rho G) \subseteq \mu$, $md_\rho(h(F *_\rho G)) \subseteq md_\rho(\mu)$[4] Therefore $h \models F : md_\rho(h(F *_\rho G))$ and $h \models F : md_\rho(\mu)$.

- Clarification. Let $h \models F : \mu_1$ and $h \models F : \mu_2$. Then $h(F) \subseteq \mu_1$ and $h(F) \subseteq \mu_2$. Therefore $h(F) \subseteq \mu_1 \cap \mu_2$, hence, $h \models F : \mu_1 \cap \mu_2$.

- Exchange. Let $h \models (A_1 *_\rho \ldots *_\rho A_k) : \mu$ and let $B_1, \ldots, B_k$ be a permutation of $A_1, \ldots, A_k$. By the Commutativity property of formula functions $h(B_1 *_\rho \ldots *_\rho B_k) = h(A_1 *_\rho \ldots *_\rho A_k) \subseteq \mu$. Then $h \models (B_1 *_\rho \ldots *_\rho B_k) : \mu$.

- Interval Weakening. Let $h \models F : \mu$ and $\mu \subseteq \mu_1$. Since $h(F) \subseteq \mu$, also $h(F) \subseteq \mu_1$ and therefore $h \models F : \mu_1$.■

We will combine the proof of completeness of HGR_P with the proof of the existence of polynomially bounded derivations for each valid inference in the next section.

[4]Let $[a, b] \subseteq [a', b']$. If ρ is a conjunctive p-strategy, then $md_\rho([a, b]) = [a, 1] \subseteq [a', 1] = md_\rho([a', b'])$. If ρ is a disjunctive p-strategy then $md_\rho([a, b]) = [0, b] \subseteq [0, b'] = md_rho([a', b'])$.

7.4.2 Proof Complexity

We are now ready to study the complexity of proofs in HGR_P. It is known that all "natural" proof systems for standard classical propositional logic have proofs of exponential size (see e.g. [106]). In this section we show that this is not the fact in our proof system HGR_P.

The first result we prove states that for programs $P \in HPP^0$, HGR_P is both a *complete inference system* and that the length of the proofs in HGR_P is *linearly bounded*. This will be then extended to the entire class of HPPs.

Proposition 9 *For any $P \in HPP_0$ and annotated formula $F : \mu$ if $P \models F : \mu$ there exists such an HGR_P-derivation $C_1, \ldots C_r$ that $C_r = F : \mu$ that $r \leq 10|F| + 6$.*

Proof.

Let $P \in HPP^0$ and $P \models F : [a, b]$. Let us fix two partitions of F, $F = G_1 *_\rho \ldots *_\rho G_k$ and $F = H_1 *_\rho \ldots *_\rho H_l$ that and numbers $x_1, \ldots, x_k, y_1, \ldots, y_l$ which satisfy Proposition 7. Then a canonic proof $C_1, \ldots, C_r = F : [a, b]$ of $F : [a, b]$ can be constructed in the following way:

1. The last formula, $C_r = F : [a, b]$ is obtained by the Interval weakening rule applied to formula $C_{r-1} = F : [a', b']$ for some $[a', b'] \supseteq [a, b]$;

2. The inference rule of Clarification is used only once in the entire proof to obtain a formula $C_{r-1} = F : [a', b']$ from two formulas $C_{r-2} = F : [a', 1]$, $C_{r-3} = F : [0, b']$.

3. Formula $C_{r-2} = F : [a', 1]$ is obtained via the rule of Exchange from formula $C_{r-4} = F' : [a', 1]$ where F' is a formula which is a permutation of atoms in F.

4. Formula $C_{r-3} = F : [0, b']$ is obtained via the rule of exchange from formula $C_{r-5} = F'' : [0, b']$ where F'' is a formula which is a permutation of atoms in F.

5. For some integer $s < r - 5$ such that $C_1, \dots C_s, C_{r-5} = F'' : [0, b']$ is a proof of $F'' : [0, b']$ and $C_{s+1}, \dots C_{r-6}, C_{r-4} = F' : [a', 1]$ is proof of $F' : [a', 1]$.

6. In partition $F' = G_1 *_\rho \dots *_\rho G_k$, for every i if G_i is a head of a clause $G_i : [a'_i, b'_i] \longleftarrow$ the proof includes two formulas:

   (a) $G_i : [a'_i, b'_i]$ (as an axiom);

   (b) $G_i : \mu_i$, $\mu_i = [x_i, 1]$ (obtained from previous formula by the rule of Interval Weakening).

   If G_i is a subformula of some head of a clause $H = G_i \oplus_\rho H' : [a'_i, b'_i] \longleftarrow$ of P the proof includes the following four formulas:

   (a) $H : [a'_i, b'_i]$ (as an axiom);

   (b) $G_i *_\rho H' : [a'_i, b'_i]$ (obtained from previous formula by the rule of Exchange);

   (c) $G_i : [c_i, d_i]$ where $[c_i, d_i] = md_\rho([a'_i, b'_i])$ (obtained by Decomposition rule from previous formula);

   (d) $G_i : \mu_i$, $\mu_i = [x_i, 1]$ ((obtained from previous formula by the rule of Interval Weakening).

   After all the formulas $G_i : \mu_i$ ($i \in \{1, \dots, k\}$) are derived, the proof of $F' : [a', 1]$ will contain the formula $G_1 *_\rho G_2 : c_\rho(\mu_1, \mu_2)$ obtained from $G_1 : \mu_1$ and $G_2 : \mu_2$ by either A-composition or F-composition rule (depending on whether both G_1 and G_2 are atomic) and for all $2 < i \leq k$ the formula $G_1 *_\rho \dots *_\rho G_i : c_\rho(\mu_1, \dots, \mu_i)$ which is obtained by the rule of F-composition from formulas $G_1 *_\rho \dots *_\rho G_{i-1} : c_\rho(\mu_1, \dots, \mu_{i-1})$ and $G_i : \mu_i$.

The last of such formulas, $G_1 *_\rho \ldots *_\rho G_k : c_\rho(\mu_1, \ldots, \mu_k) = F' : [0, a'] = C_{r-4}$.

7. The proof of $C_{r-5} = F'' : [0, b']$ is constructed from the partition $F'' = H_1 *_\rho \ldots *_\rho H_l$ in a manner similar to the construction of the proof of $F' : [0, a'] = C_{r-4}$ on previous step.

We can now compute the length of the proof constructed as above. It is bounded by $5(k + l) + 6 \leq 10|F| + 6$.■

It is possible to generalize the above result to show that HGR_P is a complete inference system for all HPPs P and further that it guarantees polynomially-sized proofs.

Theorem 23 *For any $P \in HPP$ and annotated formula $F : \mu$ if $P \vdash F : \mu$ then there exists such an HGR_P-derivation $C_1, \ldots C_r$ that $C_r = F : \mu$ and $r \leq O(|P|^2 + |F : \mu|)$.*

Proof.

If $P \in HPP$ then we can construct the proof of $F : \mu$ in two stages. At the first stage we will derive all formulas of $P_0, P_1, \ldots P_{2m}$ (as defined in theorem 2) where m is the number of clauses in P. If a clause $C_j \in Fired_P(i + 1) \setminus Fired_P(i)$ and $C_j = D : \nu \longleftarrow D_1 : \mu_1 \wedge \ldots \wedge D_s : \mu_s$ then $P_i \models D_l : \mu_l$ for all $1 \leq l \leq s$ and we include in our proof the proofs of all $D_l : \mu_l$ and using them will derive $D : \nu$ via the Program inference rule that corresponds to the clause C_j.

At the second stage we use the construction from proposition 9 to derive $F : \mu$ using the formulas of P_{2m} derived during the first stage.

The number of formulas of form $D_i : \mu_i$ we derived at the first stage of the proof is less than the length of P. The length of each D_i is less then the length of P as well. So, the total length of the proof is less then $O(|P|^2 + |F|)$. ■

Chapter 8

Hybrid Probabilistic Programs: A summary of results

Hybrid Probabilistic Logic Programs (HPLPs) have been developed with the goal of creating a framework for reasoning with uncertain information in situations where multiple relationships between events can exist. To our knowledge, only Lakshmanan and Sadri [66, 65] have developed a framework which takes into account different dependencies between different events. While our work is similar to their in some aspects, there are also substantial semantical differences in our approach and theirs. We provide a more comprehensive comparison of our work and Probabilistic Deductive Databases [66] of Lakshmanan and Sadri in Chapter 15.

Below we list our main contributions in developing the HPLP framework.

Hybrid Probabilistic LPs: Syntax and Semantics

- Using the notion of probabilistic strategy, derived from the work on ProbView [64], we have defined the syntax of Hybrid Probabilistic Logic Programs. The syntax introduced conjunction and disjunctions connectives annotated with probabilistic strategies, which specify a particular assumption about the relationship between the events (such as independence, positive correlation, or the lack of any information about a possible relationship) is to be used when the probability of a complex event is computed from the probabilities of simple events and when the probability of a simple event is to be computed from the probability of a complex event.

- We introduce hybrid formula functions, which serve as models of our Hybrid Probabilistic Programs (hp-programs). A hybrid formula function assigns for each basic formula a probabilistic interval.

- We provide fixpoint semantics for hp-programs. The semantics is based on the iterations of "immediate consequence" operator T_P which maps hybrid formula functions to hybrid formula functions. We prove the properties of hybrid formula functions and T_P operator which guarantee the existence of the fixpoint.

- We show that our fixpoint semantics and the model theory for hp-programs coincide, i.e. that the least fixpoint of the T_P operator plays the role of "minimal" model of hp-program P.

Proof Procedures for HPLPs

We have described a number of different proof procedures for hp-programs.

- hp-resolution, the first procedure we have defined, given an hp-program P uses a preprocessing step to compile a closure of P – a new hp-program which contains all hp-clauses which logically follow from the rules in P. The resolution of the query proceeds against the closure of P.

  This proof procedure is shown to be sound and complete w.r.t. the semantics of hp-programs given by fixpoint procedure.

However, the preprocessing step has a potentially prohibitive computational cost, and thus this proof procedure cannot be used in practice.

The main reason for defining it was the relative simplicity of the soundness and completeness proofs w.r.t. fixpoint semantics. This allows us for all other proof procedures to prove their soundness and completeness w.r.t. hp-resolution, with the proofs being considerably simpler.

- To avoid the preprocessing step necessary for hp-resolution procedure, we have introduced HR_P resolution, which is a dynamic, inference-rule based proof procedure for hp-programs. Instead of pre-compiling the closure of the hp-program P against which the refutation of the query Q is to proceed, HR_P resolution uses a set of inference rules that allow it to infer only necessary parts of the closure (i.e. only the rules that are to be directly used in the refutation of the query).

  Shown to be sound and complete, HR_P resolution is a more practical proof procedure for hp-programs. Later, we also introduce proof systems HGR_P, based on HR_P inference system. The main difference between the two is that HR_P produces hp-clauses as the results of inference, while proofs in HGR_P, which is applicable to ground hp-programs only produce basic formulas which can be derived from program P. HGR_P systems are used to study proof complexity for hp-programs (see description below).

- We have also studied the use of tabulation procedures in conjunction with our framework. We have defined b-caches – the bounded tables which store information gathered as the resolution of a query proceeds. The information stored in the b-cache can be used in the later stages of query resolution to avoid the necessity of resolving same or similar subqueries.

Unlike in classical logic programming, the information stored in a b-cache is not simply the truth or falsehood of propositions, but currently determined probability intervals for different (not necessarily atomic) formulas. An added complication to a $b$cache management is the fact that addition of information about the probability interval for formula F can trigger the necessity to change information about the probabilities of other formulas (subformulas and superformulas of F, for example).

To deal with that we have introduced a notion of a *b-cache update strategy* - an operation of updating b-cache with new information. We presented examples of different b-cache update strategies and showed that there exists a trade-off between the completeness of the update and computational complexity of the algorithm performing it.

We have introduced b-cache supported proof procedure which works in a manner similar to HR_P-resolution, only at each step the new procedure updates the b-cache with new information (if available) and anytime a new subquery is to be resolved, first b-cache is checked to see whether the successful answer to the query have already been derived.

The b-cache supported proof procedure is also shown to be sound and complete.

Algorithms and Complexity for HPLPs

We have studied a number of problems that traditionally arise in a logic programming framework and provided algorithms and complexity analysis for them.

- We have designed algorithms for computing the least fixpoint of the T_P operator given a program P. For arbitrary hp-programs, the computational complexity of the algorithm computing the least fixpoint of a program P is $O(m(sa^N)^2)$ where m is the number of clauses in P, s is the number of different probabilistic strategies used in P, a is the number of different atoms used in P and N is the maximum size of a formula found in P.

When a program P has only atoms as heads we showed that the least fixpoint is uniquely defined by the probabilistic intervals assigned to the atoms found in P.

- We have considered the complexity of entailment problem for consistent ground hp-programs. For the purpose of complexity analysis we have separated the class of hp-programs into three disjoint classes: HPP_1 – class of hp-programs with atomic heads, HPP_2 – class of hp-programs with heads of size at most 2 and $HPP_{\geq 3}$ – class of hp-programs with heads longer than 2. The complexity analysis for entailment problem for each class was then provided.

  - For class HPP_1, the algorithm for computing the least fixpoint can be used to check entailment. Since for HPP_1 programs least fixpoints are uniquely defined by the probability intervals assigned to atoms, entailment for this class turns out to be polynomial in the size of program+query.

  - For class $HPP_{\geq 3}$ we show that entailment problem is NP-complete.

    NP-hardness is obtained by reducing a well-known NP-complete problem 3-Dimensional Matching in polynomial time to our entailment problem. In fact, entailment is NP-hard even if the bodies of all clauses in an hp-program are empty (i.e., hp-program consists only of facts).

    We also provide an NP algorithm for entailment.

  - Finally, for class HPP_2 of hp-programs entailment problem is shown to be equivalent to a (pair of) Generalized Weighted Matching (GWM) problem on complete graphs where the edge weights get combined by the functions employed by probabilistic strategies.

The complexity of generalized weighted matching in general case is an open problem. However, as Samir Khuller pointed to us, for all functions used by our "standard" set of probabilistic strategies (disjunctive and conjunctive independence, ignorance, positive correlation and negative correlation/ minimal overlap) there exist simple reductions of the corresponding GWM problem to the classical Weighted Matching problem on complete graphs which is known to be polynomially solvable [70].

Therefore, as it turns out, complexity of entailment for the class HPP_2 of hp-programs is dependent on the set of probabilistic strategies used in the program, but is polynomial for the "standard" set of strategies.

- Our approach to the problem of determining consistency of an hp-program is similar to our approach to entailment problem: we studied consistency for each of the three classes of hp-programs considered above.

  - For the class HPP_1 the problem of checking consistency can be solved by a modification of the algorithm which computes the least fixpoint of hp-programs, and therefore the complexity of the consistency check is polynomial.
  - For class $HPP_{\geq 3}$, we showed that inconsistency problem is NP-complete, which makes consistency check co-NP-complete.
  - For class HPP_2 we show that inconsistency problem is polynomially reducible to a pair of GWM problems. This means that whenever GWM problem

is polynomially solvable (as is the case for all "standard" probabilistic strategies), consistency check can also be performed in polynomial time.

- Finally, we studied the complexity of the proofs in proof systems for HPLPs. We have considered an inference system HGR_P for ground hp-programs, which (as mentioned above) is a modification of the inference system used in HR_P resolution procedure which allows to derive formulas instead of hp-clauses.

  We proved that if an hp-program P satisfies basic formula $F : \mu$, then there exists a proof of that fact in HGR_P of polynomial size.

  This result is consistent with the NP-completeness of the general entailment problem, as while existence of short proofs is guaranteed by our result, the search leading to them may take superpolynomial time.

Part III

Managing Uncertainty and Time: Temporal Probabilistic Databases

Chapter 9

Temporal Probabilistic Databases

"Time is what prevents everything from happening at once".

– John Archibald Wheeler.

9.1 Temporal Uncertainty: Motivation and the Need for New Framework

As illustrated by the examples in Sections 1.2.1 and 1.2.2, (Logistics, scheduling and other prediction problems) in a number of applications one has to model statements like " event e occurs at some time in interval $[t_1, t_2]$ with probability between 40% and 50%".

Statements of this kind represent *temporal probabilistic information*. For us temporal probabilistic information possesses the following properties:

- Events are known *exactly*, i.e. there is no uncertainty as to what is supposed to happen or has happened. For example "A letter is sent from Paris to Rome by Jack" contains an exact description of the event.
- Uncertainty exists about *when* the events did/will occur. For example, the state-

ment "A letter is sent from Paris to Rome by Jack some time between 1st and 5th of October, 1999" contains an exact event description with a possible range of dates when it could have occurred. Implicit here is the assumption that only one letter was sent from Paris to Rome by Jack, with only one of the dates (October 1st, 2nd, 3d, 4th or 5th, 1999) being the possible time of the occurrence of this event.

- *Probabilistic information* quantifying the uncertainty about the time of the occurrence of the events is available. For example, "A letter is sent from Paris to Rome by Jack some time between 1st and 5th of October, 1999 with probability distributed uniformly by day" contains the information that the probability that a letter was sent on any given single day between 1st and 5th of October is 20%

As one can see from the above description, temporal probabilistic information has a number of fairly specific features. Therefore, a question arises as to whether designing a specific database framework for managing temporal probabilistic information would be preferable to using generic frameworks like ProbView [64] or PDM [22].

There are some arguments supporting the necessity of the new framework. First, as Dyreson and Snodgrass point in [35], the granularity of time in certain applications with temporal probabilistic information may be very small. It is quite possible that a network monitoring/scheduling application will require granularity of 1 millisecond or less. Therefore, frameworks like ProbView[64] or PDM [22] which physically store the probability of every instance will be forced to deal with potentially thousands or even millions of instances.

When the granularity of time is 1 millisecond, a statement "A network malfunction occurred sometime between 10:30 and 10:44" generates $15 \cdot 60 \cdot 1000 = 900000$

instances to be physically stored in the database, which carries a prohibitive cost. As a consequence, any framework dealing with temporal probabilistic information must store this information compactly, while still providing the possibility of accessing the information at any available level of granularity. For example, regardless of how the information contained in the statement above is stored, the user should be able to obtain information about the probability of a malfunction occurring within the first two seconds after $10:34$.

Another argument for the new framework comes from the observation that in temporal probabilistic information uncertainty exists only in the temporal attribute. Comparing to generic probabilistic frameworks the of Barbara et. al [7] and ProbView [64], where in general more than one attribute could carry a probabilistic distribution over it, this is a simplification. Therefore, one would hope that it would possible to exploit this fact in a framework tailored specifically to manage Temporal Probabilistic information.

Our final argument in favor of a new framework comes from the observation that due to the specific nature of the probabilistic information to be managed, the query language required to manage it has to be fairly specific. Namely, we have to decide what would constitute acceptable semantics of operations such as intersection, union, difference and cartesian product in the presence of temporal probabilistic information. While it is possible that different interpretations of these operations exist, none of the currently available frameworks provides a complete set of operations that would produce acceptable results for temporal probabilistic data. ProbView [64] comes close, and in fact provides the intuition for most operations in the proposed TP-database framework, but the specifics of our data required us to use somewhat different approaches to difference and projection. In Section 2.4 we have explained that some of the operations proposed by Dey and Sarkar [22] for their PDM model have ambiguous semantics. In addition, both relational algebras for ProbView and PDM operate on non-compact data, which as mentioned above is unacceptable. Beyond that, no other probabilistic database framework proposed a full-scale query language.

Dyreson and Snodgrass were the first to realize the need for a specific framework to deal with Temporal Probabilistic Data (Temporal Indeterminacy) [34, 35]. Their approach was directed at compact storage of the information and they were the first, to our knowledge, who used *probability distribution functions* in the framework.

Our work on Temporal Probabilistic Databases was influenced by the approach of Dyreson and Snodgrass. There are, however, some differences in our frameworks. In Chapter 15 we will talk about the parallels and differences between our framework and theirs in more detail (Section 15.3.1). At this point we will only mention that while Dyreson and Snodgrass were augmenting temporal databases with probabilistic mechanisms, we add a temporal dimension to probabilistic databases.

Our presentation of the Temporal Database Framework is broken into 4 chapters. In the rest of this chapter we introduce our data model: *Temporal Probabilistic Relations (TP-relations)*. In addition we introduce *annotated relations*, the flat relational counterparts of TP-relations, which allow us to easily describe the semantics of the relational algebra operations.

Chapter 10 describes *Theoretical Annotated Temporal Algebra (TATA)* – the relational algebra on annotated relations which serves as the denotational semantics for *Temporal Probabilistic Algebra (TPA)* on TP-relations, described in Chapter 11. In the latter chapter we show that TPA operations *correctly implement* the semantics of TATA operations.

Chapter 12 contains a number of query equivalence theorems that hold in TATA and TPA. Finally, Chapter 13 summarizes our TP-database framework.

9.2 Syntax of Temporal Probabilistic Databases

9.2.1 Calendars and Temporal Constraints

Calendars

First, we define the concept of a *calendar* that is used by a TP application. In our architecture, a TP application assumes the existence of an arbitrary but fixed calendar. The definitions in this section are not new, but taken from [63].

Definition 9.2.1 (time unit) *A* time unit *consists of a* name *and a* time-value set. *The time-value set has a linear order, denoted $<_T$, where T is the name of the time unit. As usual, we let $\leq_T$ denote the reflexive closure of the $<_T$ relation. A time unit is either* finite *or* infinite*, depending on whether its time-value set is finite or infinite; an infinite time-value set is assumed to be countable.* □

For instance, the time units named *day*, *month*, and *year* may have the time-value sets $\{1, \ldots, 31\}$, $\{1, \ldots, 12\}$, and {all integers} respectively.

Definition 9.2.2 (linear hierarchy) *A* linear hierarchy of time units*, denoted H, is a finite collection of distinct time units with a linear order $\sqsubseteq$ among those time units. The greatest time unit according to $\sqsubseteq$ may be either finite or infinite, while all other time units in the hierarchy must be finite.* □

For instance, $H_1 =$ *day* $\sqsubseteq$ *month* $\sqsubseteq$ *year*, $H_2 =$ *minute* $\sqsubseteq$ *hour* $\sqsubseteq$ *day* $\sqsubseteq$ *month* $\sqsubseteq$ *year*, and $H_3 =$ *hour* $\sqsubseteq$ *day* $\sqsubseteq$ *month* are all linear hierarchies of time units.

Definition 9.2.3 (time point) *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. We use $(v_1/.../v_n)$ to abbreviate time point t.*

Time points are ordered according to the lexicographic ordering $<_H$ which is defined in the usual way. Thus, time point $t = (v_1, \ldots, v_n) <_H t' = (v'_1, \ldots, v'_n)$ iff there exists an i ($1 \le i \le n$) such that $v_i <_{T_i} v'_i$ and $v_j = v'_j$ for all $j = i + 1, \ldots, n$. Note that if $i = n$, then the ($v_j = v'_j$) statement is vacuously true. When $t <_H t'$, we say that t occurs before t', and conversely, t' occurs after t. If $t = t'$, we say that t occurs simultaneously with t'. □

A time point in linear hierarchy H is simply an instantiation of each time unit in H (a specific point in time with respect to H). For instance, using hierarchy H_1 given above, "March 16, 1997" could be specified by the time point (16/3/1997). By using hierarchy H_2 given above, "3:45pm on March 12, 1997" could be specified by the time point (45, 15, 12, 3, 1997). For hierarchy H_1 given above, time point t *occurs before or simultaneously with* t', denoted $t = (v_{day}, v_{month}, v_{year}) \le_{H_1} t' = (v'_{day}, v'_{month}, v'_{year})$, is true iff $((v_{year} < v'_{year}) \vee (v_{year} = v'_{year} \wedge v_{month} < v'_{month}) \vee (v_{year} = v'_{year} \wedge v_{month} = v'_{month} \wedge v_{day} \le v'_{day}))$.

Definition 9.2.4 (calendar) *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;* 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* valid$_H(t)$ *is true*$\}$. □

For instance, for the Gregorian calendar τ and hierarchy H_1 given above, a suitable validity predicate would state that valid(14/3/1996) = true but valid(29/2/1997) = false. (29/2/1997) is not a valid time point since February of 1997 only contains 28 days. Note that a calendar for the hierarchy *dayOfWeek* $\sqsubseteq$ *day* $\sqsubseteq$ *month* $\sqsubseteq$ *year* should have only one valid time point for each instantiation of (*day*, *month*, *year*) since these three time units uniquely determine the valid time-value for *dayOfWeek*.

Let $next_\tau(t)$ denote the next, consecutive time point after t. Thus, $next_\tau(t)$ denotes the time point $t' \in S_\tau$ where t' occurs after t and for all other $t'' \in S_\tau$ where t'' occurs after t, t'' also occurs after t'.

Constraints

When expressing a statement of the form "The event described by data tuple d occurs at some time point in a set T of timepoints with probability in the interval $[p_1, p_2]$" we must be able to specify the set T of time points. Constraints are a natural way of specifying such sets. In this section, we recapitulate (from [63]) how temporal constraints can be used to specify sets of time points associated with a calendar.

Definition 9.2.5 (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 *must take one of the following forms:*

1. *(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. Here, (T_i op v_i) is called an* atomic time-value constraint.

2. *($t_1 \sim t_2$) where $t_1, t_2 \in S_\tau$ and $t_1 \leq_H t_2$. Here, ($t_1 \sim t_2$) is called an* atomic time-interval constraint. *For convenience, let (t_1) be an abbreviation for ($t_1 \sim t_1$).* □

For example, (*day* $<$ 15), (*month* $\geq$ 8), and (12/3/1997 $\sim$ 10/4/1997) are all atomic temporal constraints, but (1996 = *year*) is not. Also, (*day* $<$ 45) is not an atomic temporal constraint since 45 is not in the time-value set of *day*. Similarly, (15/2/1997 $\sim$ 29/2/1997) is not an atomic temporal constraint since (29/2/1997) is not a valid time point in τ. Furthermore, (10/4/1997 $\sim$ 12/3/1997) is not an atomic temporal constraint since time point (10/4/1997) *occurs after* (12/3/1997).

Definition 9.2.6 (temporal constraint) *A* temporal constraint C over calendar τ *is defined inductively in the following way:*

- *Any atomic temporal constraint over τ is a temporal constraint over τ.*
- *If C_1 and C_2 are temporal constraints over τ, then $(C_1 \wedge C_2)$, $(C_1 \vee C_2)$, and $(\neg C_1)$ are temporal constraints over τ.*

If temporal constraint C is a boolean combination of only atomic time-value constraints, then C is a time-value constraint. *Similarly, if temporal constraint C is solely a boolean combination of atomic time-interval constraints, then C is a* time-interval constraint. $\square$

For instance, $((\mathit{day} > 5 \wedge \mathit{day} < 15) \wedge (\mathit{month} = 4 \vee \mathit{month} \geq 8) \wedge year = 1996)$ and ((12/3/1997 $\sim$ 10/4/1997) $\vee$ (10/7/1997 $\sim$ 10/7/1997)) are temporal constraints but $(\mathit{day} > 5 \neg \wedge \mathit{day} < 15)$ is not.

Definition 9.2.7 (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 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 which is defined in the following way:*

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

□

For instance, the solution set to (*day* $>$ 25) over the Gregorian calendar τ is the set of all time points (*day*, *month*, *year*) $\in\ \tau$ where *day* $>$ 25. Note that (29, 2, 1996) is in this set but (29, 2, 1997) is not since the latter is not a valid time point in S_τ. Also, the solution set to (1/1/1996 $\sim$ 31/12/1996) over the Gregorian calendar would contain 366 time points (one for each calendar day in 1996) while the solution set to (1/1/1997 $\sim$ 31/12/1997) over the same calendar would contain 365 time points.

Definition 9.2.8 (solution set to a temporal constraint) *Let S_τ be the set of all valid time points over calendar τ. Then the* solution set to a temporal constraint C over calendar τ, *abbreviated $sol(C)$, is the set S which is defined inductively in the following way:*

- *If C is an atomic temporal constraint, then $S = sol(C)$.*
- *If C is of the form $(C_1 \wedge C_2)$, then $S = sol(C_1) \cap sol(C_2)$.*
- *If C is of the form $(C_1 \vee C_2)$, then $S = sol(C_1) \cup sol(C_2)$.*
- *If C is of the form $(\neg C_1)$, then $S = S_\tau - sol(C_1)$.*

Each time point $t \in sol(C)$ is called a solution to C. □

For example, the solution set to ((5/8/1997 $\sim$ 10/8/1997) $\vee$ (7/8/1997 $\sim$ 12/8/1997)) would contain eight time points.

The following well known result states that any time-value constraint can be rewritten as an equivalent time-interval constraint (i.e., one which has an equal solution set) and vice-versa.

Proposition 10 (Folk theorem) *Time-value constraints and time-interval constraints have the same expressive power.*

Proof.

Let τ be a calendar and $(name_1, \ldots, name_k)$ be the list of the names of its time units.

To prove this proposition it suffices to show that (i) every atomic time-value constraint can be represented as a time-interval constraint and (ii) every atomic time-interval constraint can be represented as a time-value constraint.

We do just that below:

1. For every atomic time-interval constraint C there exists a time-value constraint C' such that $sol(C) = sol(C')$.

   Let $C = (t_1 \sim t_2)$ be an atomic time-interval constraint. Let $t_1 = (v_1, \ldots, v_k)$ and $t_2 = (v'_1, \ldots, v'_k)$.

   Now, let $sol(C) = \{s_1, \ldots, s_m\}$.

   Let $s = (v^s_1, \ldots v^s_k)$ be a time point. We construct a time-value constraint C_s as follows:

   $$C_s = (name_1 = v^s_1 \wedge name_2 = v^s_2 \wedge \ldots \wedge name_k = v^s_k)$$

   By definition of solution set for a temporal constraint $sol(C_s) = \{s\}$. But then for a time-value constraint

   $$C' = C_{s_1} \vee C_{s_2} \vee \ldots \vee C_{s_k}$$

   the solution set is defined as the union of solution sets of C_{s_i} ($1 \leq i \leq k$), i.e.:

   $$sol(C') = sol(C_{s_1}) \cup \ldots \cup sol(C_{s_k}) = \{s_1\} \cup \ldots \cup \{s_k\} = \{s_1, \ldots, s_k\} = sol(C).$$

Note: C' constructed as described above is going to be a very large constraint. In practice it is always possible to construct much more compact time-value constraints for representing a time period.

2. For every atomic time-interval constraint C there exists a time-value constraint C' such that $sol(C) = sol(C')$.

   Let $C = (name_i opv_i)$ be an atomic time-value constraint. We construct an equivalent time-interval constraint as follows.

   Let t be a timepoint and let C_t be a time-interval constraint defined as follows:

$$C_t = (t \sim t)$$

   By definition of the solution set of a temporal constraint, $sol(C_s) = \{s\}$.

   Now, let $sol(C) = \{t_1, \ldots, t_m\}$. The solution set of a time-interval constraint C' defined as

$$C' = C_{t_1} \vee C_{t_2} \vee \ldots \vee C_{t_m}$$

   will be equal to

$$sol(C') = sol(C_{t_1}) \cup \ldots \cup sol(C_{t_m}) = \{t_1\} \cup \ldots \cup \{t_m\} = \{t_1, \ldots, t_m\} = sol(C)$$

   Note: C' constructed as described above is going to be a very large constraint. In practice it is always possible to construct much more compact time-value constraints for representing a time period. □

Definition 9.2.9 (finite calendar) *Calendar τ is a* finite calendar *iff S_τ is finite.* □

Note that for all temporal constraints C over a finite calendar τ, $sol(C)$ must be a finite subset of S_τ. Given a finite calendar τ, we use t_S^τ to denote the smallest time point of τ (w.r.t. the ordering $<_H$ associated with the calendar) and t_E^τ to denote the largest time point. When τ is clear from context, we will drop the superscript τ and just write t_S and t_E.

In our Temporal Probabilistic Database framework we use only finite calendars.

9.2.2 TP-tuples and TP-relations

In this section, we define the syntax and semantics of a Temporal-Probabilistic relation. Intuitively, a TP-relation is a multiset of TP-tuples. Each TP-tuple consists of a "data" part and a "probabilistic-temporal" part. This latter part is called a TP-case statement and it intuitively specifies the probability with which the "data" part of the tuple is in the relation at different instances of time.

We start the formal description of the syntax of TP-tuples by defining distribution functions which will play an important role in TP-tuples. Distribution functions will allow us to store information about the distribution of probability among the timepoints stored in a tp-tuple. We will proceed by defining TP-cases and once TP-cases are defined in Section 9.2.2 below, we will provide a formal definition of TP-tuples and TP-relations in Sections 9.2.2 and 9.2.2.

Distribution Functions

Consider a simple statement saying that the event described by some data tuple d occurs at some time point in the set $\{1, 2, 3, 4\}$ with probability 0.7. Suppose we are now asked "what is the probability that d is in r at time 2?" There is no way to answer this question without assuming the existence of some probability distribution. In this paper, we wish to allow designers of TP-databases to specify probability distributions for each set of time points.

Definition 9.2.10 (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 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.* □

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 $t_j \in sol(D)$. Partial PDFs are useful when modeling infinite distributions; here, we are considering only a finite portion of the total probability mass. 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 in advance to allocate only a total of 0.9 is determinate.

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

Definition 9.2.11 (PDF; uniform) *The PDF for* the uniform distribution *over calendar τ, denoted 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.*

Notice that for all $t_1, t_2 \in \mathrm{sol}(D)$, $\rho_1 = \rho_2$. In other words, we are equally dividing the probability mass among all of the relevant time points. Also, $\sum_{t_j \in S_\tau}(\rho_j) = 1$ is clearly true since there are $n = |\mathrm{sol}(D)|$ non-zero ρ_js, one for each $t_j \in \mathrm{sol}(D)$, and

$n \cdot \rho_j = |\mathrm{sol}(D)| \cdot \frac{1}{|\mathrm{sol}(D)|} = 1$. Furthermore, we will never divide by zero since by definition of PDFs, $|\mathrm{sol}(D)| \geq 1$.

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 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. For instance if $D = (1/8/1997 \sim 3/8/1997)$ then $n = 2$, $t_0 = 1/8/1997$, $t_1 = 2/8/1997$, and $t_2 = 3/8/1997$.

Definition 9.2.12 (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 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.*

If $p = \frac{1}{3}$, pdf$_{g,p}(D, t_0) = \frac{1}{3} \cdot (\frac{2}{3})^0$, pdf$_{g,p}(D, t_1) = \frac{1}{3} \cdot (\frac{2}{3})^1$, and pdf$_{g,p}(D, t_2) = \frac{1}{3} \cdot (\frac{2}{3})^2$. For all other time points $t_j \in S_\tau$, pdf$_{g,p}(D, t_j) = 0$. Notice that if $p = \frac{1}{2}$, then pdf$_{g,p}(D, t_0) = \frac{1}{2}$ and pdf$_{g,p}(D, t_i)$ will be half of pdf$_{g,p}(D, t_{i-1})$ for each $1 \leq i \leq n$. Let pdf$_{g_c,p}$ be defined in the same way as pdf$_{g,p}$ except pdf$_{g_c,p}(D, t_n) = 1$ if $|\mathrm{sol}(D)| = 1$ or pdf$_{g_c,p}(D, t_n) = 1 - (\sum_{t_j \in \{t_0,\ldots,t_{n-1}\}}(\mathrm{pdf}_{g,p}(D, t_j)))$ otherwise. We call pdf$_{g_c,p}$ the *complete correlate* of pdf$_{g,p}$ since $\mathrm{pdf}_{g_c,p}(D, t_j) = \mathrm{pdf}_{g,p}(D, t_j)$ for all $t_j \in S_\tau - \{t_n\}$ and since pdf$_{g_c,p}$ is a complete PDF. In general, one can construct a complete correlate for any partial PDF in a similar way. Note that when $p = \frac{1}{2}$ and $|\mathrm{sol}(D)| > 1$, pdf$_{g_c,p}$ has the nice property that pdf$_{g_c,p}(D, t_n) = \mathrm{pdf}_{g_c,p}(D, t_{n-1})$.

Definition 9.2.13 (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 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.*

Definition 9.2.14 (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.

Techniques that specify how to associate and store probability distributions with database tuples (events) are provided by Dyreson and Snodgrass [35, p. 8]. Hence, we do not discuss this matter in further detail here, but leverage, where needed from their research.

Throughout the rest of this paper, we will use (δ = "u"), (δ = "g, p"), (δ = "g_c, p"), (δ = "b, p"), and (δ = "po, λ") to represent the distribution functions for pdf_u, $\mathrm{pdf}_{g,p}$, $\mathrm{pdf}_{g_c,p}$, $\mathrm{pdf}_{b,p}$, and $\mathrm{pdf}_{po,\lambda}$ respectively. Furthermore, unless we specifically state otherwise, assume that parameter $p = 0.5$. Thus, ($\delta =$ "g") represents the $\mathrm{pdf}_{g,0.5}$ function.

TP-case statements

We are now ready to define a TP-case statement and its constituent TP-cases.

Definition 9.2.15 (TP-case statement over calendar τ) *A* TP-case statement γ *over calendar* τ *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. *$sol(C_i) \subseteq sol(D_i)$. This ensures that $\delta_i(D_i, t)$ is defined for each time point $t \in sol(C_i)$.*

3. $|\text{sol}(C_i)| \geq 1$. *In other words, C_i and D_i each have at least one solution in S_τ.*

4. *For all $1 \leq j \leq n$, $i \neq j \Rightarrow sol(C_i) \cap sol(C_j) = \emptyset$. 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, sol(C_n) = 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 sol(C_i) = sol(D_i), we let "$(\#)$" be an abbreviation for C_i.* □

The reader may wonder about the occurrence 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 timepoints during which an event could have occurred, while sol(D_i) is the domain of δ_i, i.e. the set of timepoints 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 could have occurred at some time point $t \in sol(C_i)$. This probability lies somewhere in the interval $[\delta_i(D_i, t) \cdot L_i, \delta_i(D_i, t) \cdot U_i]$. So, constraint D_i serves as a domain of the probabilistic distribution function, while constraint C_i actually specifies the timepoints information about which is stored in the database. One can say that C_i in a TP-case represents temporal information about the data, while the quadruple D_i, L_i, U_i, δ_i represent probabilistic information. **This is an important distinction which is critically necessary.** Why ? Suppose that originally, $\text{sol}(C_i) = \text{sol}(D_i) = S = \{1, 2, 3, 4\}$ and $\delta_i =$ "$b, 0.5$". Thus, the probability weights associated with time points 1,2,3,4 are $0.125, 0.375, 0.375, 0.125$. Now suppose we perform a *selection* operation (Section 10.4) which only asks for time points in the set $S' = \{1, 3\} \subset S$. Notice that selection does not change the probabilities stored the relation. If we had no D_i field in our TP-cases, then we would contract the contents of C_i in the result of selection of the set $\{1, 3\}$. We would then have to make sure that our probabilistic data provides a correct way to compute the probability interval for the new set. Notice that merely carrying over the fact that S' has the binomial is not sufficient as a binomial distribution on a set of size two set yields a probability weight of 0.5 to both 2 and 3, stipulating that the probability intervals for both timepoints are the same which is incorrect since in the original TP-case the weights were different. Thus, some mechanism is needed to correctly compute the probabilities of relations resulting from algebraic operations that have been executed.

Thus, in order to accurately compute probabilities, we must do one of two things:

- Carry with us the original set of values over which a probability distribution was defined, or
- 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. What is more important is the fact that its implementation is likely to be extremely expensive as with each change in the content of the temporal constraint, the probabilistic information contained in the TP-case would have to be altered as well. 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 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 $[L_i \cdot \delta_i(D_i, t), U_i \cdot \delta_i(D_i, t)]$.

Even though TP-cases contain two distinct constraint fields, viz. C and D, this distinction can be hidden from the user. Later we will introduce the notion of base relations which will have equal C and D parts of all TP-cases. We will also discuss the presentation operation which will allow us to "hide" unnecessary and sometimes confusing contents of the data as stored in our TP-relations.

The expression

$\{\langle(\#), (month < 6 \wedge year = 1997), 0.4, 0.8, g\rangle,$

$\langle(\#), (month \geq 6 \wedge year = 1997), 0.6, 0.6, u\rangle\}$

is a TP-case statement. However, the expression

$\{\langle(\#), (year = 1996 \wedge year = 1997), 0.4, 0.8, g\rangle,$

$\langle(\#), (year = 1998), 0.0, 0.0, u\rangle\}$

is not a TP-case statement as the solution set to C_1 (and D_1) is empty.

Furthermore,

$\{\langle(\#), (month < 6 \wedge year = 1997), 0.4, 0.8, g\rangle,$

$\langle(\#), (month \geq 3 \wedge year = 1997), 0.6, 0.6, u\rangle\}$

is not a TP-case statement as $(C_1 \wedge C_2)$ is not inconsistent (i.e., the probabilities for the overlapping time points are over specified).

We reiterate that each temporal constraint in a TP-case statement must have a finite number of solutions (since S_τ is finite). We restrict ourselves to finite calendars and solution sets to avoid the complications which arise when trying to determine whether a constraint using negations is infinite or not.

TP-tuples

In this section, we will define the important concept of a Temporal Probabilistic tuple (TP-tuple for short). We will require that all TP-tuples contain a special field called a "hidden field." Intuitively, two TP-tuples with equal "data" parts may represent two different events. The hidden field (similar in function, but easier to implement than the concept of a *path* introduced in [64]) keeps track of how these TP-tuples were derived in order to determine whether or not two TP-tuples refer to the same event.

Definition 9.2.16 (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.* □

In base relations, the contents of the hidden field will be EMPTY (since no fields have been projected out). For intermediate relations, the hidden field holds values of the form "<field>:<value>" for fields which have been projected out. Although these values should be hidden from the user, we shall see that they are important in determining whether two TP-tuples refer to the same event or not.

Definition 9.2.17 (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.* □

For instance, suppose our relational schema is A = (Item, Origin, Dest, H). Then

Item	Origin	Dest	H	C	D	L	U	δ
I1	Rome	Vienna		$(\#)$	$day < 15 \wedge month = 11 \wedge$ $year = 1996$	0.5	0.6	u
				$(\#)$	$day \geq 15 \wedge month = 11 \wedge$ $year = 1996$	0.4	0.4	u

is a TP-tuple which indicates that item "I1" left from "Rome" and will arrive in "Vienna" in November 1996 at some time before the 15th (with $50 - 60\%$ probability) or on/after the 15th (with 40% probability). This TP-tuple is not concerned with I1's arrival before or after November 1996 since no probabilities are assigned to this time range.

If we were sure that I1 did not arrive in Vienna before or after November 1996, we could add the TP-case $\langle(\#), (*), 0, 0, u\rangle$ to the TP-tuple above. If we had no information regarding I1's arrival before or after November 1996 but we were assuming that the distribution function for this time was "u", we could add the TP-case $\langle(\#), (*), 0, 1, u\rangle$ to the TP-tuple above.

Finally, if we had no information whatsoever regarding I1's arrival before or after November 1996, we would not change the TP-tuple above. Here, we are implicitly assigning a probability interval of $[0, 1]$ to each time point t which lies outside of November 1996 since for all $1 \leq i \leq n$, $t \notin \text{sol}(C_i)$.

Some of our definitions in the following sections will rely upon the following operators which manipulate hidden lists.

Definition 9.2.18 (manifest projection) *Let $A = (A_1, \ldots, A_k)$ be a relational schema where A_k is the hidden field ("H") and let $d = (d_1, \ldots, d_k)$ be a (data) tuple over A. Then the* manifest projection of data tuple d, *denoted $\mathcal{P}(d)$, is defined as $(d_1, \ldots, d_{k-1})$. In other words, the tuple $\mathcal{P}(d)$ contains every value in d except the hidden list (d.H). Here, A_1 to A_{k-1} are known as* manifest data fields. □

Intuitively, the manifest projection of a TP-relation simply eliminates the hidden field of the TP-relation. The following definition specifies how hidden lists are concatenated.

Definition 9.2.19 (hidden list concatenation) *The* concatenation of hidden lists $d.\mathrm{H}$ and $d'.\mathrm{H}$*, denoted* $(d.\mathrm{H} \parallel d'.\mathrm{H})$*, is a hidden list* h'' *which can be constructed by lexicographically merging every field-value pair in* $d.\mathrm{H}$ *and* $d'.\mathrm{H}$*. For instance if* $d.\mathrm{H}$ *= "Fld3:Val3, Fld6:Val6" and* d'*.H = "Fld4:Val4, Fld8:Val8, Fld9:Val9", then*
h'' *= "Fld3:Val3, Fld4:Val4, Fld6:Val6, Fld8:Val8, Fld9:Val9".* □

Intuitively, the concatenation of two hidden lists can be obtained by taking the union of the two hidden lists, and then sorting them in lexicographic order.

TP-relations

We may now define a TP-relation in terms of TP-tuples.

Definition 9.2.20 (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*.*

A base TP-relation *is a TP-relation which did not result from a query. In any base relation* r *the following should hold: for each TP-tuple* $tp = (d, \gamma) \in r$*,* (i) $d.\mathrm{H} =$ *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. This key will be used when we describe the TPA's semantics for projection (Section 11.8). □

Note that TP-relations are defined to be multisets rather than sets. This is because, as is the case with classical relational databases, they may contain two or more distinct copies of the same TP-tuple. We address this issue in more detail when we discuss compactness of TP-relations and compaction operations (Sections 9.3.2, 10.1, 11.2).

Recall that a primary key is a minimal set of fields which, taken collectively, allow us to uniquely identify a tuple in a relation [58]. In the worst case, a primary key may need to contain every manifest data field in a relation. In practice, well designed databases use tuple ids, transaction ids, SSNs, timestamps, etc. to help keep the primary keys small.

Definition 9.2.21 (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*.* □

For simplicity, we require all TP-relations in a TP-database to use the same calendar. Throughout our description of the Temporal Probabilistic Databases we assume that all TP-relations are in the same TP-database.

Presentation of TP-relations

Let us return back to the definition of a tp-case statement (Definition 9.2.15. As discussed above, given a TP-tuple (d, γ) and one of its tp-cases $\langle C, D, L, U, \delta \rangle$, the two temporal constraints C and D represent, respectively, the set of timepoints at which an event specified by d could have occurred, and the domain of the probability distribution function δ. This organization of the data allows us to simplify certain relational algebra operations, most importantly, selection on temporal condition (see Section 11.5), as the change in C will not entail any changes in the attributes which store the information about the probability.

However, this data format for TP-tuples has its own drawbacks when it comes to visualizing the information stored in a TP-relation. Consider, for example the following TP-tuple.

Item	Origin	Dest	H	C	D	L	U	δ
I1	Rome	Vienna		$(1/1/1999 \sim 1/5/1999)$	$(1/1/1999 \sim 1/10/1999)$	0.5	0.6	u

If this information is returned to a user in the above format, the user may incorrectly assume that the probability that the item $I1$ will be delivered from Rome to Vienna some time during the first 5 days of 1999 is between 0.5 and 0.6. This, however is not the correct meaning of the tuple above, as 0.5 and 0.6 refer to the lower and upper bounds associated with the solution set of D. The correct probability interval for the event described by our tuple is $[0.25, 0.3]$, but the peculiarities of our format preclude us from storing this interval directly in the tuple.

However, it is unreasonable to sacrifice the readability of the output in order to accommodate the format of the data. While we do need the tp-cases to be stored in the $\langle C, D, L, U, \delta \rangle$ format in order to be able to process the queries of our relational algebra, there is no need to output TP-tuples in their *storage format*, which, as we witnessed, may be confusing.

So, what would make an appropriate output format for a TP-tuple ? As mentioned above, field C in a tp-case indicates timepoints for which the probabilistic information is available. Therefore, it would make sense if, for each tp-case of a TP-tuple, the correct probability interval were to be presented to the user. So instead of a storage format $\langle C, D, L, U, \delta \rangle$ the output could use a special *presentation format* $\langle C, L', U' \rangle$ where $[L', U']$ is the probability interval for C. For example, the TP-tuple above could then be presented as

Item	Origin	Dest	C	L	U
I1	Rome	Vienna	$(1/1/1999 \sim 1/5/1999)$	0.25	0.3

Of course, while the presentation format hides from the user the distribution function δ and its domain D, both are used when computing the probability interval for the presentation format. One more thing the presentation format hides from the user is the hidden field from the data part, which contains "identity" information for the TP-tuple, which need not be displayed.

We formalize the intuition described above by introducing a *presentation* operation. This operation takes as input a TP-tuple in storage format and return back the presentation format of this tuple.

Definition 9.2.22 (presentation operation) *Let $tp = (d, \gamma)$ be a TP-tuple and $\gamma = \gamma_1, \ldots, \gamma_n$, $\gamma_i = \langle C_i, D_i, L_i, U_i, \delta_i \rangle$ for $1 \leq i \leq n$.*

The presentation of tp, denoted $Prs(tp)$ is a tuple (d, γ'), where $\gamma' = \gamma'_1, \ldots, \gamma'_n$ and $\gamma'_i = \langle C_i, L_i \cdot \sum_{t \in sol(C_i)} \delta(D_i, t), U_i \cdot \sum_{t \in sol(C_i)} \delta(D_i, t) \rangle$, $1 \leq i \leq n$.

Given a TP-relation r, its presentation is defined as $Prs(r) = \cup_{tp \in r} Prs(tp)$.

We note that $\sum_{t \in sol(C_i)} \delta(D_i, t)$ is the probability weight of $sol(C_i)$ in $sol(D_i)$. Also, notice that when δ is complete, for base relations (and also for all cases when $C_i = D_i$) the result of presentation is obtained from the input tuple by throwing out the values for δ_i and D_i from the TP-case (as the L and U values will not change).

While defined here, presentation operation will be a part of our Temporal Probabilistic Algebra (TPA). As the result of the presentation operation is **not** a valid TP-relation, it cannot be used inside complex queries. However, it can be used after any TPA query in order to provide the user with convenient format of the result.

In the implementation, presentation operation is also used as a(n optional) filter for the operation of viewing a TP-relation.

Example 9.2.1 Let TP-relation r be

Data	C	D	L	U	δ
d_1	$(5/10/1999 \sim 9/10/1999)$	$(5/10/1999 \sim 9/10/1999)$	0.3	0.45	u
	$(12/10/1999 \sim 15/10/1999)$	$(10/10/1999 \sim 20/10/1999)$	0.5	0.5	$g_{0.5}$
d_2	$(6/10/1999 \sim 8/10/1999)$	$(4/10/1999 \sim 10/10/1999)$	0.4	0.8	$b_{0.5}$

Then $Prs(r)$ is

Data	C	L	U
d_1	$(5/10/1999 \sim 9/10/1999)$	0.3	0.45
	$(12/10/1999 \sim 15/10/1999)$	0.1171875	0.1171875
d_2	$(6/10/1999 \sim 8/10/1999)$	0.3125	0.625

To compute the probability range for the first tp-case in the TP-tuple for d_1 we need not do anything as D and C coincide and the probability distribution function (uniform) is complete. To compute the probability range for the second tp-case of this tuple, we first compute the $\sum_{t \in sol(C)} \delta(D,t)$, which, in our case is

$$pdf_{g,0.5}(D, 12/10/1999) + pdf_{g,0.5}(D, 13/10/1999) +$$
$$pdf_{g,0.5}(D, 14/10/1999) + pdf_{g,0.5}(D, 15/10/1999).$$

This sum is equal to $\frac{1}{8} + \frac{1}{16} + \frac{1}{32} + \frac{1}{64} = \frac{1}{4} - \frac{1}{64} = \frac{15}{64}$. We then multiply the upper and the lower bounds of the probability interval in TP-case (which in our case are both equal to 0.5) by this number to obtain the necessary probability interval: $0.5 \cdot \frac{15}{64} = \frac{15}{128} = 0.1171875$.

Finally, to compute the probability interval for the second tuple, we need to compute the probability weight of the three "middle" points $(2, 3, 4)$ in a binomial distribution of seven points $(0, 1, 2, 3, 4, 5, 6)$. We get

$$pdf_{b,0.5}(D, 6/10/1999) + pdf_{b,0.5}(D, 7/10/1999) + pdf_{b,0.5}(D, 8/10/1999) =$$

$$(\binom{6}{2} + \binom{6}{3} + \binom{6}{4}) \cdot 0.5^6 = (15 + 20 + 15) \cdot \frac{1}{64} = \frac{50}{64} = 0.78125.$$

Then, for lower bound we get $L = 0.78125 \cdot 0.4 = 0.3125$ and for upper bound: $U = 0.78125 \cdot 0.8 = 0.625$. $\square$

9.3 Annotated relations and Semantics of TP-relations

9.3.1 Annotated Relations

An *annotated tuple* over a relational schema $\vec{A}$ is an expression of the form $(\vec{d}, t, L, U)$ where $\vec{d}$ is an (ordinary) tuple over relational schema $\vec{A}$, t is a time point, and L, U are real numbers in the $[0, 1]$ interval. An *annotated relation* is a finite set of annotated tuples. We will often use expressions such as ann_r, $ann_{r'}$, etc. to refer to annotated relations. An annotated tuple (at) provides probabilistic information $([L, U])$ for one data tuple (d) at one point in time (t). Any tp-tuple can be converted into a set of annotated tuples as defined below.

Definition 9.3.1 (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 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 $x = \delta_i(D_i, t)\}$.* $\square$

Intuitively, in the above definition, 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 9.3.1 can be easily extended to convert an annotated relation r into a TP-relation.

Definition 9.3.2 (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 ANN(r), is defined as the multiset* $(\mathrm{ANN}(tp_1) \uplus \ldots \uplus \mathrm{ANN}(tp_n))$ *over τ, where $\uplus$ denotes multiset union operation.* $\square$

It is clear see that ANN(tp) and ANN(r) can often be large and impractical to physically instantiate. This is why we **only use annotation for theoretical purposes** such as illustrating a process or proving equivalences between query expressions. In our implementation, **we never create annotated relations**.

The table below contains an annotated relation ANN(r) for a TP-relation r consisting of one TP-tuple $tp = (d, \gamma)$, where d = ("D1", EMPTY) and $\gamma = \{\gamma_1\}$.

r

Data	H	C	D	L	U	δ
D1		$(\#)$	$day \leq 4 \wedge$ $month = 11 \wedge$ $year = 1996$	0.4	0.8	u

ANN(r)

Data	H	Day	Month	Year	L_t	U_t
D1		1	11	1996	0.1	0.2
D1		2	11	1996	0.1	0.2
D1		3	11	1996	0.1	0.2
D1		4	11	1996	0.1	0.2

However if γ_1's C_1 field was "$day \leq 3 \wedge month = 11 \wedge year = 1996$", then ANN($r$) would no longer contain the last tuple shown above. In general, note that changing C_i only affects the number of annotated tuples in ANN(r), not the probabilities for the remaining time points.

Notice the "0.1" and "0.2" values in the probabilistic fields above. These values were determined by uniformly distributing the available probability $[0.4, 0.8]$ among the four annotated tuples in ANN(r). We were only justified in making this uniformity assumption since $\delta_1 =$ "u". In general, TP-relations will only give us probability intervals for a range of time points, and determining (tight) probability intervals for each time point within that range requires us to apply a distribution function δ_i.

We associate with each annotated relation ann_r the primary key which is associated with r. This key will be used when we define the TATA's projection operation (§10.7).

Note. Any annotated tuple (d, t, L, U) can be trivially converted into a tp-tuple (d, γ) where $\gamma = \gamma_1, \gamma_1 = \langle C, D, L, U, \delta \rangle$, with $sol(C) = sol(D) = \{t\}$ and $\delta =$"u" (uniform). Therefore, any annotated relation ann_r has an associated trivial tp-relation r obtained by replacing each annotated tuple by its tp-conversion.

Below we show a number of examples of annotated relations and annotation operations.

Let r consist of one TP-tuple which contains two TP-cases as shown below.

Item	Origin	Dest	H	C	D	L	U	δ
I1	Rome	Paris		$(\#)$	$day \leq 2 \wedge month = 8 \wedge$ $year = 1997$	0.5	0.7	Φ
				$(\#)$	$day \geq 5 \wedge day \leq 7 \wedge month = 8 \wedge$ $year = 1997$	0.3	0.6	Φ

Note that the variable Φ must be instantiated. If $\Phi =$ "u", ANN(r) will be

Item	Origin	Dest	H	Day	Month	Year	L_t	U_t
I1	Rome	Paris		1	8	1997	0.25	0.35
I1	Rome	Paris		2	8	1997	0.25	0.35
I1	Rome	Paris		5	8	1997	0.10	0.20
I1	Rome	Paris		6	8	1997	0.10	0.20
I1	Rome	Paris		7	8	1997	0.10	0.20

where $[L_t, U_t] = \frac{1}{2} \cdot [0.5, 0.7]$ for the first two tuples and $[L_t, U_t] = \frac{1}{3} \cdot [0.3, 0.6]$ for the remaining tuples in $\mathrm{ANN}(r)$. However if $\Phi =$ "g", ANN(r) will be

Item	Origin	Dest	H	Day	Month	Year	L_t	U_t
I1	Rome	Paris		1	8	1997	0.25	0.35
I1	Rome	Paris		2	8	1997	0.125	0.175
I1	Rome	Paris		5	8	1997	0.15	0.30
I1	Rome	Paris		6	8	1997	0.075	0.15
I1	Rome	Paris		7	8	1997	0.0375	0.075

where $[L_t, U_t] = \frac{1}{2} \cdot [0.5, 0.7]$, $\frac{1}{4} \cdot [0.5, 0.7]$, $\frac{1}{2} \cdot [0.3, 0.6]$, $\frac{1}{4} \cdot [0.3, 0.6]$, and $\frac{1}{8} \cdot [0.3, 0.6]$ for the first through fifth tuples of ANN(r) respectively. Notice that modifying Φ (i.e., the distribution function δ) only affects the L_t and U_t fields of ANN(r).

9.3.2 Probabilistic Interpretations and 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 [98] to the case of TP-relations, by extending the concept of an interpretation in classical logic [98] to handle TP-relations. Before doing this, a preliminary definition is needed.

Definition 9.3.3 (data-identical TP-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 ($d.\mathrm{H}$ = $d'.\mathrm{H}$).* □

Definition 9.3.4 (data-identical annotated tuples) *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 TP-relations under the assumption that all data-identical TP-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.

Similarly to TP-relations, we interpret annotated relations under the same assumption that all data-identical annotated tuples refer to the same event. If at and at' are not data-identical, we assume that they refer to different events. Let d be a data tuple and let t be a time point. Then $ANN(r)[d,t]$ denotes the equivalence class of the pair (d,t), i.e., the multiset of all $at \in ANN(r)$ where $(at.d = d \wedge at.t = t)$.

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 $t \in sol(C_1 \vee \ldots \vee C_n)$. 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. Later, we will describe a variety of compaction operators which convert uncompact TP-relations into compact TP-relations by consolidating probabilistic information for each $r[tp] \subseteq r$ (see Section 11.2). The compaction operations are similar to duplicate elimination in classical relational algebra.

Suppose $at = (d, t, L_t, U_t) \in ANN(r)$, $at' = (d', t', L'_t, U'_t) \in ANN(r)$, and $(d = d' \wedge t = t')$. Here, since $at, at' \in ANN(r)$ refer to the same event at the same point in time, $ANN(r)$ is an *uncompact* annotated relation. If there are no pairs of annotated tuples $at, at' \in ANN(r)$ where $(d = d' \wedge t = t')$, then ANN(r) is a compact annotated relation. Clearly, there exists a connection between compact TP-relations and compact annotated relations. In fact, the following theorem states that the concepts of "compact relation" for TP-relations and annotated relations coincide.

Theorem 24 *A TP-relation r is compact iff its annotated counterpart, $ANN(r)$, is compact.*

Proof.

- r is compact $\Longrightarrow ANN(r)$ is compact.

  As r is compact, for each data tuple d and timepoint t there is at most one tp-tuple $tp = (d, \gamma) \in r$, $\gamma = \gamma_1, \ldots, \gamma_k$, $\gamma_i = \langle C_i, D_i, L_i, U_i, \delta_i \rangle$, for $1 \leq i \leq k$ such that $t \in sol(C_1 \vee \ldots \vee C_k$.

Let d be some data tuple such that there exists a tuple $tp = (d, \gamma) \in r$. We show that $ANN(r[tp])$ is compact. Let $r[tp] = \{tp_1, tp_2, \ldots tp_m\}$. Let C^j be the disjunction of all C-temporal constraints in tp_j. As r is compact, we know that $(\forall 1 \leq i \neq j \leq m)(sol(C^j) \cap sol(C^i) = \emptyset)$. Let t be a timepoint in $sol(C^1 \vee \ldots \vee C^m)$. Then there exists a unique $1 \leq h \leq m$ such that $t \in sol(C^h)$. Consider tp-tuple $tp_h = (d, \gamma')$ where $\gamma' = \gamma'_1, \ldots, \gamma'_k$ and $\gamma'_i = \langle C_i, D_i, L_i, U_i, \delta_i \rangle$ for all $1 \leq i \leq k$. By the definition of a TP-tuple, $(\forall 1 \leq i \neq j \leq k) sol(C_i) \cap sol(C_j) = \emptyset$. This means that for each $t \in sol(C_1 \vee \ldots \vee C_k)$ there will be only one annotated tuple $(d, t, L_t, U_t,)$ added to $ANN(tp)$, i.e., $ANN(tp_h)$ has at most one tuple $(d, t, L_t, U_t,)$ for each timepoint t. The latter means that $ANN(r[tp])$ is compact, since for two tuples $tp', tp'' \in r[tp]$, and two tuples $at' = (d, t', L', U') \in ANN(tp')$ and $at'' = (d, t'', L'', U'') \in ANN(tp'')$ it will never be the case that $t' = t''$. Compactness of $ANN(r[tp])$ for any $tp \in r$ yields the compactness of $ANN(r)$, as for any two not data-identical tuples tp and tp' in r, no two tuples $at \in ANN(r[tp])$ and $at' \in ANN(r[tp'])$ will be data-identical.

- $ANN(r)$ is compact $\Longrightarrow r$ is compact.

  Let r be a tp-relation. Consider an arbitrary annotated tuple $at = (d, t, L_t, U_t) \in ANN(r)$. As $ANN(r)$ is compact, $ANN(r)[d, t] = \{ar\}$, i.e. no other annotated tuple in $ANN(r)$ is data-time identical to at. By definition of the operator $ANN(\cdot)$, there exists a tp-tuple $tp \in r$ such that, $tp = (d, \gamma), \gamma = \gamma_1, \ldots, \gamma_k, \gamma_i = \langle C_i, D_i, L_i, U_i, \delta_i \rangle$, for $1 \leq i \leq k$ and $at \in ANN(tp)$, i.e. $t \in sol(C_1 \vee \ldots C_k)$.

  As $ANN(r)[d, t] = \{at\}$, for any other tp-tuple $tp' = (d, \gamma') \in r$ such that $\gamma' = \gamma'_1, \ldots, \gamma'_l, \gamma'_i = \langle C'_i, D'_i, L'_i, U'_i, \delta'_i \rangle$ we will have $t \notin sol(C'_1 \vee \ldots \vee C'_l)$. Therefore, by definition 11.2.1, r is a compact tp-relation. □

Consistency

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 9.3.5 (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 9.3.6 (satisfaction for TP-tuples) *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. Let $\varpi = \sum_{t \in sol(D_i)} \delta(D_i, t)$. 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. *$L_i \cdot \varpi \leq I_{A,\tau}(d, D_i) \leq U_i \cdot \varpi$, i.e. the probability that $I_{A,\tau}$ assigns to D_i lies in the interval $[L_i, U_i]$.*

2. *$(\forall t \in \mathrm{sol}(C_i))(I_{A,\tau}(d, D_i) \cdot \delta_i(D_i, t) \cdot \varpi = I_{A,\tau}(d, t))$, 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 $\gamma_i \in \gamma$.* $\square$

For example, let us consider the following TP-tuple.

Item	Origin	Dest	H	C	D	L	U	δ
I1	Rome	Vienna		$(\#)$	$day < 15 \wedge month = 11 \wedge$ $year = 1996$	0.5	0.6	u
				$(\#)$	$day \geq 15 \wedge month = 11 \wedge$ $year = 1996$	0.4	0.4	u

Consider the TP-interpretation defined as follows:

$$\begin{aligned} I(\langle \text{I1,Rome,Vienna} \rangle, (d, 11, 1996)) &= 0.04 \text{ when } d < 15. \\ I(\langle \text{I1,Rome,Vienna} \rangle, (d, 11, 1996)) &= \frac{0.4}{16} \text{ when } d \geq 15. \\ I(\langle \text{item,origin,dest}, (d, m, y)) &= 0 \text{ otherwise.} \end{aligned}$$

This TP-interpretation satisfies the TP-tuple above because

$$I(\langle \text{I1,Rome,Vienna} \rangle, \{t \mid t.day < 15 \wedge t.month = 11 \wedge t.year = 1996\}) = 0.04 \times 14 = 0.56$$

which lies between 0.5 and 0.6.

The same TP-interpretations will be used as models for annotated relations.

Definition 9.3.7 (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-interpre-tation $I_{A,\tau}$* satisfies *annotated tuple $at = (d, t, L_t, U_t)$, denoted $I_{A,\tau} \models at$, iff*

$$L_t \leq I_{A,\tau}(d, t) \leq U_t.$$

□

We should note one difference between the definitions of satisfaction for TP-tuples and annotated tuples. For a TP-interpretation to satisfy a TP-tuple tp, one extra requirement has to be met, namely, the TP-interpretation has to assign probability values for each set $sol(C_i)$ in tp in accordance with the distribution function δ_i of the tp-case γ_i.

Now we can define consistency of both TP- and annotated relations

Definition 9.3.8 (consistency and mutual consistency of TP-relations) *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.* $\square$

Definition 9.3.9 (consistency of annotated relations) *An annotated tuple* at *is* consistent *iff* $(\exists I_{A,\tau})(I_{A,\tau} \models at)$*. An annotated relation ANN(r) is* consistent *iff* $(\exists I_{A,\tau})(\forall at \in \mathrm{ANN}(r))(I_{A,\tau} \models at)$*. Annotated relations ANN(r) and ANN(r′) are* mutually consistent *iff* $(\exists I_{A,\tau})((\forall at \in \mathrm{ANN}(r))(I_{A,\tau} \models at) \wedge (\forall at' \in \mathrm{ANN}(r'))(I_{A,\tau} \models at'))$. $\square$

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

$$\sum_{i=1}^{n}(L_i \cdot \Sigma_{t \in sol(C_i)}\delta_i(D_i, t)) \quad \leq \quad 1.$$

If this condition holds, then the TP-tuple is consistent, as the formula above computes the sum of the lower bounds of probability for all timepoints t described in a TP-tuple. If this sum is less than or equal to 1, then an interpretation I which assigns to each timepoint t this lower bound will satisfy the TP-tuple, guaranteeing its consistency.

The rank of a tp-case $\gamma_i = \langle C_i, D_i, L_i, U_i, \delta_i \rangle$ is $rank(\gamma_i) = |C_i|$. Given a TP-tuple $tp = (d, \gamma); \gamma = \gamma_1, \ldots \gamma_k$, $rank(tp) = \sum_{i=1}^{k} rank(\gamma_i)$. Finally, the rank of a TP-relation r is given by: $rank(r) = \sum_{tp \in r} rank(tp)$. Now, assuming that computing the values of δ_i functions takes constant time, this allows us to state the following claim:

Proposition 11 *Checking consistency of a compact TP-relation is linear in the rank of the TP-relation.*

Proof. Suppose r is compact. As r contains no distinct data-identical tuples, r is consistent iff all tp-tuples in r are individually consistent. 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

$$\sum_{i=1}^{n} (L_i \cdot \Sigma_{t \in sol(C_i)} \delta_i(D_i, t)) \leq 1.$$

If this condition holds, then the probabilistic interpretation $I_{A,\tau}$ defined as follows satisfies tp.

$$I_{A,\tau}(d,t) = \begin{cases} L_i \times \delta_i(D_i, t) & \text{if } t \in \text{sol}(C_i), 1 \leq i \leq n \\ 0 & \text{otherwise.} \end{cases}$$

So, checking consistency of a compact relation boils down to the problem of computing $\sum_{t \in sol(C_i)} \delta_i(D_i, t_i)$ for each TP-case in the TP-relation. In the worst case, one will have to compute $\delta_i(D_i, t_i)$ for each $t_i \in C_i$, i.e. for each TP-case $\langle C_i, D_i, L_i, U_i, \delta_i \rangle$ computing $\sum_{t \in sol(C_i)} \delta_i(D_i, t_i)$ is linear w.r.t. $|C_i|$, which in turn means that checking consistency of a TP-tuple tp is linear in $rank(tp)$ and since consistency of each tuple in a compact relation r has to be established independently, it can be done in $O(rank(r))$ time. $\square$

Note that if we assume that pdfs are polynomially computable, then the algorithm discussed in the proof above will work in time $O(rank(r) \cdot P(\tau))$ where $P(\tau)$ is the maximum of the complexities of the pdfs in r.

Now we are ready to establish the results linking satisfaction of TP- and annotated relations and their consistency.

Theorem 25 *Let r be a TP-relation. If $I_{A,\tau}$ satisfies r, then $I_{A,\tau}$ also satisfies* $\mathrm{ANN}(r)$.

Proof. We show that if $I \models r$ then also $I \models ANN(r)$.Let $tp = (d, \gamma) \in r$ be some tp-tuple and $at = (d, t, L, U) \in ANN(r)$. Since $I \models r$, $I \models tp$. We need to show that $I \models at$.

Let $\gamma = \gamma_1, \ldots, \gamma_k$, and $\gamma_i = \langle C_i, D_i, L_i, U_i, \delta_i \rangle$ for $1 \leq i \leq k$. Since $at = (d, t, L, U) \in ANN(r)$, there exists some $1 \leq j \leq k$ such that, $t \in sol(C_j)$ and $[L, U] = [\delta(D_j, t)L_i, \delta(D_j, t)U_i]$.

Since $I \models tp$, $I \models \langle d, D_j, [L_j, U_j], \delta_j \rangle$. By the definition of satisfaction, $I(d, D_j) \in [L_j, U_j]$ and $I(d, t) = \delta_j(D_j, t)I(d, D_j)$. But then $L = \delta_j(D_j, t)L_j \leq \delta_j(D_j, t)I(d, D_j)$ $= I(d, t) \leq \delta_j(D_j, t)U_j = U$, i.e., $I(d, t) \in [L, U]$. From the latter it follows that $I \models (d, t, L, U) = at$. $\square$

An immediate consequence of this theorem is

Corollary 7 *If TP-relation r is consistent, then so is* $\mathrm{ANN}(r)$.

The converse of Theorem 25 is not true, however, i.e., it may be the case that a TP-interpretation satisfies ANN(r), but does not satisfy r. This is shown in the following example.

Example 9.3.1 (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 since the details of the distribution get lost when annotating a relation, which 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 $I_{A,\tau}(d, t) = L_t$. This means that the lower bounds contained in $\mathrm{ANN}(r)$ are tight, and hence, $\mathrm{ANN}(r)$ correctly captures the implied probability intervals for data tuple d at time t.

Theorem 26 *Let r be a compact TP-relation containing a TP-tuple (d, γ), and suppose $(d, t, L_t, U_t) \in \mathrm{ANN}(r)$. Then there is a TP-interpretation $I_{A,\tau}$ satisfying r such that $I_{A,\tau}(d, t) = L_t$.*

Proof. Let $I_{A,\tau}(d, t)$ be computed as follows. As r is compact, all tuples of the form $(d, -, -, -) \in ANN(r)$ are derived from $ANN(tp)$ where $tp = (d, \gamma)$ where $\gamma = \langle C_1, D_1, L_1, U_1, \delta_1\rangle, \ldots, \langle C_m, D_m, L_m, U_m, \delta_m\rangle$. Let $ANN(tp)$ consist of $(d_1, t_1, L_1, U_1), \ldots, (d_k, t_k, L_k, U_k)$. We know that $\Sigma_{i=1}^{k} L_i \leq 1$. Without loss of generality, suppose t is a solution of C_i for some $1 \leq i \leq m$. Clearly, exactly one such i exists. Let $\{t'_1, \ldots, t'_r, t'_{r+1}, \ldots, t'_{r+s}\}$ be all *other* solutions of D_i, and let $\mathrm{sol}(C_i) = \{t, t'_1, \ldots, t'_r\}$. For all $1 \leq i \leq r$, (d, t'_i, L'_i, U'_i) is in $ANN(r)$. We know that $L_i \leq L + L'_1 + \cdots + L'_r \leq U_i$. We now construct an interpretation $I_{A,\tau}$ that satisfies r and which assigns L to $I_{A,\tau}(d,t)$ as follows. We know that $x \cdot I_{A,\tau}(d,y) = \delta_i(D_i,y)$ for some $x > 0$. By setting $I_{A,\tau}(d,t) = L$, we know that $x = \frac{\delta_i(D_i,t)}{L}$. Therefore, we may set

$$I_{A,\tau}(d,t'_i) \quad = \quad \delta_i(D_i,t'_i) \cdot \frac{L}{\delta_i(D_i,t)}.$$

The same procedure may be used to extend $I_{A,\tau}$ to other tuples that are not data-identical to $(d,-,-,-)$. □

Theorems 25 and 26 jointly tell us that as far as lower bounds are concerned, r and $\mathrm{ANN}(r)$ 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 26 is to hold.

Definition 9.3.10 (tightening) *Let $tp = (d,\gamma)$, $\gamma = \gamma_1,\ldots,\gamma_n$, $\gamma_i = \langle C_i, D_i, U_i, L_i, \delta_i\rangle$ be a tp-tuple. A tightening of tp is a tp-tuple $tp' = (d,\gamma')$, $\gamma' = \gamma'_1,\ldots,\gamma'_m$, $\gamma'_i = \langle C'_i, D'_i, U'_i, L'_i, \delta'_i\rangle$, such that*

1. $\cup_{i=1}^{n} C_i = \cup_{i=i}^{m} C'_i$.

2. *For all TP-interpretations $I_{A,\tau}$, $I_{A,\tau} \models (d,\gamma)$ iff $I_{A,\tau} \models (d,\gamma')$.*

3. *For all* $t \in \cup_{i=1}^{n} C_i$, *let* $(d, t, L, U) \in ANN(tp)$ *and* $(d, t, L', U') \in ANN(tp')$. *Then* $L' = L$ *and* $U' \leq U$.

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, upper bounds are reachable by satisfying TP-interpretations.

Theorem 27 *Let* r *be a compact, tight, TP-relation containing a TP-tuple* (d, γ)*, and suppose* $(d, t, L_t, U_t) \in \mathrm{ANN}(r)$*. Then there is a TP-interpretation* $I_{A,\tau}$ *satisfying* r *such that* $I_{A,\tau}(d, t) = U_t$*.*

Proof. Similar to the proof of Theorem 26. □

Theorems 26 and 27 jointly tell us that the conversion of a TP-relation r to annotated form preserves bounds when r is tight and compact. Later, in Section 11.10, we will describe a procedure for tightening TP-relations.

9.4 Review of Definitions.

Before proceeding to describe the relational algebras that operate on Annotated and TP-Relations, let us summarize the definitions given in this chapter.

- We have defined TP-Relations: the data model for storing temporal probabilistic information. Our units of storage, TP-tuples consist of two parts: data part and a tp-case statement.

The data part of a TP-tuple is a collection of relational fields which describe some instantaneous event. The tp-case statement contains information about the times at which the event described by the data part could have occurred together with the probabilistic information.

The definition of TP-tuples used the notions of *calendars* and *temporal constraints* which have also been described here following their treatment in [63]. Also described in the chapter are *probabilistic distribution functions*, which are used in TP-tuples as well.

- As the format in which TP-tuples are stored contains information sufficient to compute the probability of the event occurring not only for some time in current set of timepoints, but also in any of its subsets, displaying the TP-tuples to the user in the format they are stored may be somewhat confusing. To combat that, we have introduced *presentation format* for TP-tuples. A TP-tuple in presentation format contains the description of an event (data part), information about the timepoint at which it could have occurred and the probability that it occurs at one of these timepoints. TP-tuples in their *storage format* can be converted into presentation format via the *presentation operation* also defined here.

- TP-tuples are fairly complex structures representing possibly information about a large number of timepoints. Defining relational algebra on TP-Relations requires understanding of how the structure of TP-tuples changes. Following [64] and [22] we look for analogs of TP-relations which would better resemble classical relations. Our hope is that defining relational algebra on such relations would provide the necessary intuition for defining relational algebra on TP-relations (in a sense it would serve as the denotational semantics).

In fulfillment of this plan we have defined *annotated tuples* and *annotated relations*. A single tuple of an annotated relation contains data part, temporal part, which describes exactly one timepoint and the probability that the event described by the data part occurs at the timepoint stored in the temporal part. We have defined an Annotation operation ANN which converts TP-tuples into annotated relations and extended this operation to map TP-relations to annotated relations.

- We have defined model-theoretic semantics for both TP- and annotated relations. Based on this semantics we have defined the consistency of TP- and annotated relations and established the relationship between the consistency of TP-relations and their annotated counterparts. Also, based on the model theory we have defined the notion of *tight* TP-tuples and TP-relations. Tight TP-tuples contain probabilistic intervals which are *exact* in the sense that model theory allows any probability value contained in these intervals to be reachable.

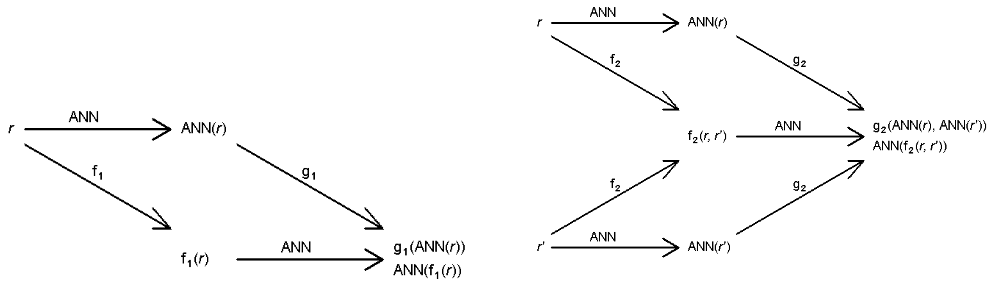
Having defined both TP-Relations and annotated relations, we can now proceed with the second part of our plan. In the next chapter we will define Theoretical Annotated Temporal Algebra (TATA), the relational algebra which works on annotated relations. We will then proceed to define the relational algebra on TP-Relations, called Temporal Probabilistic Algebra (TPA), in Chapter 11 using (and preserving) the semantics of TATA operations.

Chapter 10

Theoretical Annotated Temporal Algebra (TATA)

In this chapter, we define the Theoretical Annotated Temporal Algebra and provide definitions for compaction, intersection, union, selection, difference, cartesian product, projection, and join on annotated relations.

Figure 10.1 shows what we hope to accomplish through this section. We know that every TP-relation can be converted into a (potentially very large) annotated relation. As annotated relations are explicit representations of TP-relations, all of the above operations on annotated relations can be explicitly defined and justified — this is what we will do in this section. Then, in Chapter 11, we will show how these operations can be implemented in the TP-Algebra in such a way that the TP-Algebra operations implement



(a) Unary Operators (b) Binary Operators

Figure 10.1: Commutativity between operators of the TATA and TPA algebras

r_1

Data	H	C	D	L	U	δ
D1		$(\#)$	$(1/8/1997 \sim 3/8/1997)$	0.64	0.88	g
		$(\#)$	$(5/8/1997 \sim 8/8/1997)$	0.40	0.80	u

$ANN(r_1)$

Data	H	Day	Month	Year	L_t	U_t
D1		1	8	1997	0.32	0.44
D1		2	8	1997	0.16	0.22
D1		3	8	1997	0.08	0.11
D1		5	8	1997	0.10	0.20
D1		6	8	1997	0.10	0.20
D1		7	8	1997	0.10	0.20
D1		8	8	1997	0.10	0.20

Figure 10.2: TP- and annotated relations used in TATA and TPA examples (part 1).

the annotated algebra operations on the implicit (smaller) TP-relations, rather than their larger annotated counterparts.

The definitions in this section will produce a new annotated relation ann''_r based on input from consistent annotated relations ann_r, ann'_r. Oftentimes, these definitions will refer to annotated tuples at, at' which are assumed to be of the form $at = (d, t, L_t, U_t)$ and $at' = (d', t', L'_t, U'_t)$.

Note: Our examples illustrating the Theoretical Annotated Temporal Algebra and the TP-Algebra will be based on the relations shown in Figures 10.2, 10.3, 10.4 and 10.5.

r_2

Data	H	C	D	L	U	δ
D1		$(\#)$	$(2/8/1997 \sim 3/8/1997)$	0.20	0.50	g
		$(\#)$	$(6/8/1997 \sim 9/8/1997)$	0.40	0.80	u

$ANN(r_2)$

Data	H	Day	Month	Year	L_t	U_t
D1		2	8	1997	0.10	0.25
D1		3	8	1997	0.05	0.125
D1		6	8	1997	0.10	0.20
D1		7	8	1997	0.10	0.20
D1		8	8	1997	0.10	0.20
D1		9	8	1997	0.10	0.20

Figure 10.3: TP- and annotated relations used in TATA and TPA examples (part 2).

r_3

Data1	Data2	H	C	D	L	U	δ
D1	D2		$(\#)$	$(2/8/1997 \sim 2/8/1997)$	0.20	0.40	u
D1	D3		$(\#)$	$(2/8/1997 \sim 3/8/1997)$	0.60	0.80	g

$ANN(r_3)$

Data1	Data2	H	Day	Month	Year	L_t	U_t
D1	D2		2	8	1997	0.20	0.40
D1	D3		2	8	1997	0.30	0.40
D1	D3		3	8	1997	0.15	0.20

Figure 10.4: TP- and annotated relations used in TATA and TPA examples (part 3).

r_3

Data1	Data2	H	C	D	L	U	δ
D1	D2		$(\#)$	$(2/8/1997 \sim 3/8/1997)$	0.20	1.00	g
D1	D3		$(\#)$	$(3/8/1997 \sim 3/8/1997)$	0.50	0.50	u
D4	D5		$(\#)$	$(1/8/1997 \sim 1/8/1997)$	0.70	0.80	u

$ANN(r_4)$

Data1	Data2	H	Day	Month	Year	L_t	U_t
D1	D2		2	8	1997	0.10	0.50
D1	D2		3	8	1997	0.05	0.25
D1	D3		3	8	1997	0.50	0.50
D4	D5		1	8	1997	0.70	0.80

Figure 10.5: TP- and annotated relations used in TATA and TPA examples (part 4).

10.1 Compaction of an annotated relation

The first operation we define will be compaction as this operator is needed to define other operators. Compaction is the TP analog of duplicate elimination in the relational algebra. Combination functions described below will play an important part in implementation of compactions.

10.1.1 Combination functions

Suppose we are trying to determine the probability that a single event e is true at time point t. Occasionally, we may have multiple sources of information where each source provides a different probability interval for e at time t. Here, combination functions can be used as a generic mechanism for combining these intervals into a single $[L, U]$ result.

Definition 10.1.1 (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*.* □

Combination functions will be useful when describing TATA and TPA semantics for intersection (Sections 10.2, 11.3) and union (Sections 10.3, 11.4). For instance, after a union merges all tuples from two relations, the resulting relation may contain more than one tuple for a single event. Here, we could compact (merge) these tuples into a single tuple by applying a combination function.

Definition 10.1.2 (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 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 10.1.3 (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 10.1.1 (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 $\bigcap_{[L,U]\in S}[L,U]$.

□

Proposition 12 *Every function listed in Example 10.1.1 is an equity combination function.*

Proof.

If S is such that $\cap_{[L,U]\in S}[L,U] \neq \emptyset$ then the values of all functions defined in the example 10.1.1 will coincide on S, as they are all defined to be equal to the intersection. Clearly,$\cap_{[L,U]\in S}[L,U]$ satisfies Identity as $[L,U] \cap [L,U] = [L,U]$. Also, since $\cap_{[L,U]\in S}[L,U] = [\max_{[L,U]\in S}(L), \min_{[L,U]\in S}(U)]$ and $[\max_{[L,U]\in S}(L), \min_{[L,U]\in S}(U)] \leq [\max_{[L,U]\in S}(L), 1]$ the Bottomline is satisfied. We also notice that when $S = \{[L,U]\}$, $\cap_{[L,U]\in S}[L,U] \neq \emptyset$, i.e. Identity need not be considered when $\cap_{[L,U]\in S}[L,U] = \emptyset$.

Let now $\cap_{[L,U]\in S}[L,U] = \emptyset$. We prove Bottomline for all six functions:

- Optimistic Equity.

  $\chi_{eq}(S) = [\max_{[L,U]\in S}(L), \max{[L,U] \in S(U)}] \leq [\max_{[L,U]\in S}(L), 1]$ i.e. χ_{eq} satisfies Bottomline.

- Enclosing Equity.

  $\chi_{ec}(S) = [\min_{[L,U]\in S}(L), \max{[L,U] \in S(U)}] \leq [\max_{[L,U]\in S}(L), 1]$ i.e. χ_{ec} satisfies Bottomline.

- Pessimistic Equity

  $\chi_{ep}(S) = [\min_{[L,U]\in S}(L), \min{[L,U] \in S(U)}] \leq [\max_{[L,U]\in S}(L), 1]$ i.e. χ_{ep} satisfies Bottomline.

- Rejecting Equity

  $\chi_{er}(S) = [0, 0] \leq [\max_{[L,U]\in S}(L), 1]$ i.e. χ_{er} satisfies Bottomline.

- Skeptical Equity.

  $\chi_{er}(S) = [0, 1] \leq [\max_{[L,U]\in S}(L), 1]$.

- Quasi-Independence Equity.

  $\chi_{eqi}(S) = [\Pi_{[L,U]\in S}L, \Pi_{[L,U]\in S}U] \leq [\max_{[L,U]\in S}(L), 1]$. □

10.1.2 Compaction

Definition 10.1.4 (Compaction of an annotated relation) *A function κ from annotated relations to annotated relations is called a* compaction operation *if it satisfies the following axioms:*

- Compactness : $\kappa(ann_r)$ *is compact for all annotated relations* ann_r.

- No Fooling Around (NFA) : *If ann_r is compact, then $\kappa(ann_r) = ann_r$.*

- Conservativeness : *If $at = (d, t, L_t, U_t) \in \kappa(ann_r)$, then $\exists at' = (d, t, L'_t, U'_t) \in ann_r$.* □

The Compactness axiom assures us that the result of a compaction operation will be a compact relation. The NFA axiom states that applying the compaction operation to a compact relation should not change the relation. The Compactness and NFA axiom jointly guarantee that compaction operations are idempotent, i.e. $\kappa(\kappa(ann_r)) = \kappa(ann_r)$. 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 class of compaction strategies involves the use of a combination function (as defined in Section 10.1.1).

Definition 10.1.5 (χ-compaction of an annotated relation) *Let χ be a combination function. Then the* χ-compaction of annotated relation ann_r, *denoted $\kappa_\chi(ann_r)$, is defined as*

$$\kappa_\chi(ann_r) = \{at = (d, t, L_t, U_t) \mid [L_t, U_t] = \chi(\{[L_1^{(d,t)}, U_1^{(d,t)}], \ldots, [L_k^{(d,t)}, U_k^{(d,t)}]\})\},$$

where $ann_r[d, t] = \{at_1^{(d,t)}, \ldots at_k^{(d,t)}\}$ and $at_i^{(d,t)} = (d, t, L_i^{(d,t)}, U_i^{(d,t)})$. □

Intuitively, in combination function based compactions, χ is applied to the multiset of all $[L_i, U_i]$s associated with (d, t). The resulting $[L, U]$ then becomes the only probability interval associated with (d, t). The following proposition states that an operation defined in this manner is indeed a compaction operation.

Proposition 13 *Let χ be any combination function. Then $\kappa_\chi(.)$ is a compaction operation.*

Proof.

Let χ be a combination function and let ann_r be an annotated relation. We show that κ_χ satisfies the properties of compaction operations.

1. Compactness. By definition 10.1.5 $\kappa_\chi(ann_r)$ contains *only one* annotated tuple for each pair (d, t) where d is data tuple and t is a timepoint. Therefore, $\kappa_\chi(ann_r)$ is compact.

2. No Fooling Around. Let ann_r be compact. Then for any data tuple d and timepoint t, $ann_r[d, t]$ will contain *at most* one tuple. Let $ann_r[d, t] = \{(d, t, L_t, U_t)\}$. Then, by definition 10.1.5, $\kappa_\chi(ann_r)$ will contain a tuple $(d, t, \chi(\{[L_t, U_t]\})$. As χ is a combination function, it satisfies the Identity property of combination functions, i.e. $\chi(\{d, t, L_t, U_t\}) = [L_t, U_t]$, i.e., $(d, t, L_t, U_t) \in \kappa_\chi(ann_r)$ or $ann_r \subseteq \kappa_\chi(ann_r)$.

   To prove that $\kappa_\chi(ann_r) \subseteq ann_r$ we observe that by definition 10.1.5, **the only** tuples in $\kappa_\chi(ann_r)$ are those inserted there by applying χ to the sets $\{[L_t, U_t]\}$ of probabilistic intervals from $ann_r[d, t]$ equivalence classes. As we have shown above, **all** such tuples will be the tuples from the original relation ann_r.

3. Conservativeness. We notice that if there exists a tuple $at = (d, t, L_t, U_t) \in \kappa_\chi(ann_r)$, then by definition 10.1.5 there had to be a *nonempty* set $ann_r[d, t] \subset ann_r$, which means that for any such tuple in $\kappa_\chi(ann_r)$ there is always at least one tuple $at' = (d, t, L'_t, U'_t)$ in ann_r. $\square$

Theorem 28 indicates that $\kappa_\chi(ann_r)$ operations also possess another important property: Reasonableness. Intuitively, this property is similar to the converse of Conservativeness — every tuple in ann_r leads to a corresponding tuple in the result of the compaction.

Theorem 28 *If* $at' = (d, t, L'_t, U'_t) \in ann_r$*, then* $\exists at = (d, t, L_t, U_t) \in \kappa_\chi(ann_r)$.

Proof. Let χ be a combination function and ann_r be an annotated relation. Let $at' = (d, t, L', U') \in ann_r$. Then $ann_r[d, t] \neq \emptyset$. Let $ann_r[d, t] = \{at'_1, \ldots, at'_k\}$, $at'_i = (d, t, L'_i, U'_i)$ for $1 \leq i \leq k$. By definition of combination function-based compaction, $\kappa_\chi(ann_r)$ will contain the tuple $at = (d, t, L, U)$ where $[L, U] = \chi(\{[L'_1, U'_1], \ldots, [L'_k, U'_k]\})$. □

Another possible class of compaction strategies involves the use of a *p-strategy* ρ (i.e., a probabilistic conjunction or disjunction strategy as defined in Section 3.1). These compactions, denoted κ_ρ, are defined in the same way as $\kappa_\chi(ann_r)$ except we let $[L_t, U_t] = ([L_1^{(d,t)}, U_1^{(d,t)}] \otimes_\rho \ldots \otimes_\rho [L_k^{(d,t)}, U_k^{(d,t)}])$ when ρ is a conjunctive p-strategy, and let $[L_t, U_t] = ([L_1^{(d,t)}, U_1^{(d,t)}] \oplus_\rho \ldots \oplus_\rho [L_k^{(d,t)}, U_k^{(d,t)}])$ when ρ is a conjunctive p-strategy.

Proposition 14 *Let ρ be any (conjunctive or disjunctive) p-strategy. Then $\kappa_\rho(ann_r)$ is a compaction operation.*

Proof. We need to show that $\chi_\otimes$ satisfies the three axioms defining compaction operators.

1. Compactness. By the definition of a p-strategy based compaction, for any given pair (d, t), κ_ρ computes $ann_r[d, t]$ consisting of all tuples in ann_r of the form $(d, t, -, -)$. All tuples in $ann_r[d, t]$ are combined into one by the construction of the definition of a p-strategy based compaction.

2. No Fooling Around. If ann_r is compact, then for all (d, t), ann_r contains at most one tuple of the form $(d, t, -, -)$. In this case, the definition of κ_ρ returns that tuple unchanged.

3. **Conservativeness.** Suppose $\kappa_\rho(ann_r)$ contains a tuple of the form $(d, t, -, -)$. Then it is easy to see that by the definition a of p-strategy based compaction there must exist at least one tuple in ann_r of the form $(d, t, -, -)$. $\square$

10.2 Intersection of two annotated relations

The intersection of annotated relations ann_r and ann'_r extracts information which is common to both relations. In our algebra, we break intersection into two suboperations: First, a multiset intersection will extract all tuples from both ann_r and ann'_r which contain "common information". Then, we will use one of our previously-defined compaction operators to compact the result of this multiset intersection. Finally, intersection will be defined as a combination of these suboperations. Note that intersection (and multiset intersection) is only defined when both relations have the same schema.

Definition 10.2.1 (multiset intersection of two annotated relations) *The* multiset intersection of annotated relations ann_r and ann'_r*, denoted* $ann_r \cap ann'_r$*, is defined as* ann''_r = $\{at \in ann_r \mid (\exists at' \in ann'_r)(d = d' \wedge t = t')\} \cup \{at' \in ann'_r \mid (\exists at \in ann_r)(d = d' \wedge t = t')\}$. $\square$

Intuitively, ann''_r will contain all $at \in ann_r$ and all $at' \in ann'_r$ where at and at' refer to the same event at the same point in time. Recall that "$(\exists at \in ann_r)$" and "$(\exists at' \in ann'_r)$" are shorthand for "$(\exists (d, t, L_t, U_t) \in ann_r)$" and "$(\exists (d', t', L'_t, U'_t) \in ann'_r)$" respectively. For greater clarity and conciseness, our definitions will make use of this implicit notation. For example, $ann''_r = \mathrm{ANN}(r_1) \cap \mathrm{ANN}(r_2)$ will be

Data	H	Day	Month	Year	L_t	U_t
D1		2	8	1997	0.16	0.22
D1		3	8	1997	0.08	0.11
D1		6	8	1997	0.10	0.20
D1		7	8	1997	0.10	0.20
D1		8	8	1997	0.10	0.20
D1		2	8	1997	0.10	0.25
D1		3	8	1997	0.05	0.125
D1		6	8	1997	0.10	0.20
D1		7	8	1997	0.10	0.20
D1		8	8	1997	0.10	0.20

Clearly, ann''_r above is uncompacted. To obtain a compact annotated relation, we may use any κ_χ compaction operator. Using a κ_χ compaction operator when defining intersection makes sense because the two different relations r and r' may both contain data tuple d at some time point t, but with different probabilities. In this case, using a conjunction strategy is not appropriate because we are not combining probabilities of different events — we are combining two different probabilities assigned to the same even by two different sources (relations r and r'). This is exactly what combination functions χ were designed to support.

Definition 10.2.2 (intersection of two annotated relations) *The* intersection of annotated relations ann_r *and* ann'_r *under the* χ *combination function, denoted* $ann_r \cap_\chi ann'_r$*, is defined as* $\kappa_\chi(ann_r \cap ann'_r)$. □

For example, $\mathrm{ANN}(r_1) \cap_{eq} \mathrm{ANN}(r_2)$ will be

Data	H	Day	Month	Year	L_t	U_t
D1		2	8	1997	0.16	0.22
D1		3	8	1997	0.08	0.11
D1		6	8	1997	0.10	0.20
D1		7	8	1997	0.10	0.20
D1		8	8	1997	0.10	0.20

10.3 Union of two annotated relations

Just like intersection, the union of two annotated relations will be presented as a combination of two suboperations: multiset union, which combines the information from two relations together and compaction, which compacts the result. As always, union is only defined when both relations have the same schema.

Definition 10.3.1 (multiset union of two annotated relations) The *multiset union of annotated relations* ann_r and ann'_r, denoted $ann_r \cup ann'_r$, is defined as $ann''_r = ann_r \uplus ann'_r$. □

Intuitively, ann''_r will contain all $at \in ann_r$ and all $at' \in ann'_r$. For example, $ann''_r = \mathrm{ANN}(r_1) \cup \mathrm{ANN}(r_2)$ will be

Data	H	Day	Month	Year	L_t	U_t
D1		1	8	1997	0.32	0.44
D1		2	8	1997	0.16	0.22
D1		3	8	1997	0.08	0.11
D1		5	8	1997	0.10	0.20
D1		6	8	1997	0.10	0.20
D1		7	8	1997	0.10	0.20
D1		8	8	1997	0.10	0.20
D1		2	8	1997	0.10	0.25
D1		3	8	1997	0.05	0.125
D1		6	8	1997	0.10	0.20
D1		7	8	1997	0.10	0.20
D1		8	8	1997	0.10	0.20
D1		9	8	1997	0.10	0.20

As in the case of intersection, ann''_r may overspecify probabilistic information. We can consolidate this information by using a κ_χ compaction operator. The reason for using the operator κ_χ instead of a conjunction strategy is exactly for the same reason that we used the κ_χ compaction operator when defining intersection (see discussion preceding Definition 10.2.2).

Definition 10.3.2 (union of two annotated relations) *The* union of annotated relations *ann_r and ann'_r under the χ combination function, denoted $ann_r \cup_\chi ann'_r$, is defined as $\kappa_\chi(ann_r \cup ann'_r)$.* □

For example, $\mathrm{ANN}(r_1) \cup_{eq} \mathrm{ANN}(r_2) = \kappa_{eq}(ann''_r)$ will be

Data	H	Day	Month	Year	L_t	U_t
D1		1	8	1997	0.32	0.44
D1		2	8	1997	0.16	0.22
D1		3	8	1997	0.08	0.11
D1		5	8	1997	0.10	0.20
D1		6	8	1997	0.10	0.20
D1		7	8	1997	0.10	0.20
D1		8	8	1997	0.10	0.20
D1		9	8	1997	0.10	0.20

Note that although ANN(r_1) and ANN(r_2) are both consistent, $\mathrm{ANN}(r_1) \cup_{eq} \mathrm{ANN}(r_2)$ above is an inconsistent annotated relation (since for some data tuple d, the sum of the L_t values exceeds 1.0). This occurs since ANN(r_1) and ANN(r_2) are not mutually consistent (Section 9.3.2). In general, if consistent annotated relations ann_r and ann'_r are also mutually consistent, then $ann_r \cup_{eq} ann'_r$ will always be consistent.

10.4 Selection on an annotated relation

We represent a selection condition over calendar τ by the symbol $\mathcal{C}$. If $\mathcal{C}$ is of the form (F *op* v) or ($t_1 \sim t_2$), then $\mathcal{C}$ is an atomic condition over τ. Let $\mathcal{C}$ be an atomic condition, let $T_1 \sqsubseteq \cdots \sqsubseteq T_m$ be a linear hierarchy H of time units over τ, and suppose TP-relation r is over relational schema $A = (A_1, \ldots, A_k)$. Then one of the following cases must hold:

- If $F = A_i$ for some $1 \leq i < k$, then $\mathcal{C}$ is a *data condition*.
- If $F = T_j$ for some time unit T_j in H or if $\mathcal{C}$ is of the form ($t_1 \sim t_2$), then $\mathcal{C}$ is a *temporal condition*.

- If $F =$ "L" or $F =$ "U", then $\mathcal{C}$ is a *probabilistic condition*.

- Otherwise, $\mathcal{C}$ is an *inapplicable condition*. In this case, $\sigma_{\mathcal{C}}(r)$ and $\sigma_{\mathcal{C}}(ann_r)$ are not defined. Notice that selections on the hidden field (i.e., $F = A_k$) are not permitted. Throughout this paper, we will assume that $\mathcal{C}$ is not an inapplicable condition.

Definition 10.4.1 (selection on an annotated relation; atomic condition) *The* selection of atomic condition $\mathcal{C}$ on annotated relation ann_r*, denoted* $\sigma_{\mathcal{C}}(ann_r)$*, is defined in the following way:*

- *If $\mathcal{C}$ is a data condition,* $ann''_r = \{at \in ann_r \mid d$ *satisfies* $\mathcal{C}\}$.
  In this case, our selection is based on the classical relational algebra.

- *If $\mathcal{C}$ is a temporal condition,* $ann''_r = \{at \in ann_r \mid t \in \mathrm{sol}(\mathcal{C})\}$.

- *If $\mathcal{C}$ is a probabilistic condition,* $ann''_r = \{at \in ann_r \mid ([L, U] = [L_t, U_t])$ *satisfies* $\mathcal{C}\}$. □

For example if $\mathcal{C} = (2/8/1997 \sim 7/8/1997)$, $\sigma_{\mathcal{C}}(\mathrm{ANN}(r_1))$ and $\sigma_{\mathcal{C}}(\mathrm{ANN}(r_2))$ will be respectively

Data	H	Day	Month	Year	L_t	U_t
D1		2	8	1997	0.16	0.22
D1		3	8	1997	0.08	0.11
D1		5	8	1997	0.10	0.20
D1		6	8	1997	0.10	0.20
D1		7	8	1997	0.10	0.20

and

Data	H	Day	Month	Year	L_t	U_t
D1		2	8	1997	0.10	0.25
D1		3	8	1997	0.05	0.125
D1		6	8	1997	0.10	0.20
D1		7	8	1997	0.10	0.20

If $\mathcal{C} = (L \neq 0.10)$, $\sigma_{\mathcal{C}}(\mathrm{ANN}(r_1))$ and $\sigma_{\mathcal{C}}(\mathrm{ANN}(r_2))$ will be (respectively)

Data	H	Day	Month	Year	L_t	U_t
D1		1	8	1997	0.32	0.44
D1		2	8	1997	0.16	0.22
D1		3	8	1997	0.08	0.11

and

Data	H	Day	Month	Year	L_t	U_t
D1		3	8	1997	0.05	0.125

Later, we will describe how to perform selections with non-atomic selection conditions (Section 11.5).

10.5 Difference of two annotated relations

As in the classical relational algebra, difference is only defined when both relations have the same schema. There are many possible ways of defining difference, but we have chosen to base our definition on the intuition that if two relations r and r' represent the information derived from two different sources, then $r - r'$ should represent the information about the world that r has and r' does not.

Definition 10.5.1 (difference of two annotated relations) *The* difference of annotated relations ann_r *and* ann'_r, *denoted* $ann_r - ann'_r$, *is defined as* $ann''_r = \{at \in ann_r \mid (\forall at' \in ann'_r), (d \neq d' \vee t \neq t')\}$. $\square$

Thus, ann''_r will not include $at \in ann_r$ if there exists an $at' \in ann'_r$ which refers to the same event at the same point in time. For example, $\mathrm{ANN}(r_1) - \mathrm{ANN}(r_2)$ and $\mathrm{ANN}(r_2) - \mathrm{ANN}(r_1)$ will be respectively

Data	H	Day	Month	Year	L_t	U_t
D1		1	8	1997	0.32	0.44
D1		5	8	1997	0.10	0.20

and

Data	H	Day	Month	Year	L_t	U_t
D1		9	8	1997	0.10	0.20

Suppose $at_1 = (d, t, 0.2, 0.4) \in ann_r$ and $at_2 = (d, t, 0, 1) \in ann'_r$. Then by definition, ann''_r will not contain at_1. Now suppose we removed from ann'_r all annotated tuples where $[L'_t, U'_t] = [0, 1]$. Here, ann''_r will contain at_1. Apparently, we cannot simply throw out tuples where $[L'_t, U'_t] = [0, 1]$.

Intuitively, if we do not have an annotated tuple for data tuple d at time t, then "we do not know anything about (d, t)'s probability". In this case, (d, t) is implicitly assigned a probability interval of $[0, 1]$. On the other hand, at_2 indicates that "we know that we do not know anything about (d, t)'s probability". This distinction is subtle yet important; by keeping these two cases distinct, we allow both the *closed world assumption* (where (d, t) is implicitly assigned a probability interval of $[0, 0]$) and the *open world assumption*.

10.6 Cartesian product of two annotated relations

Each tuple in the result of a cartesian product reflects the conjunction of two events. Suppose that at time t, events e_1 and e_2 have probability intervals $[L_1, U_1]$ and $[L_2, U_2]$ respectively. In order to compute the probability interval $[L, U]$ for the event $(e_1 \wedge e_2)$ at time t, we must apply a probabilistic conjunction strategy α, i.e., $[L, U] = [L_1, U_1] \otimes_\alpha [L_2, U_2]$ (Section 3.1). This allows users to ask queries such as "Compute the cartesian product of annotated relations ann_r and ann'_r under the assumption that there is no information about dependencies between events in these relations."

Definition 10.6.1 (cartesian product of two annotated relations) *The* cartesian product of annotated relations ann_r *and* ann'_r *under the* α *probabilistic conjunction strategy, denoted* $ann_r \times_\alpha ann'_r$*, is defined as* $ann''_r = \{(d'', t, L''_t, U''_t) \mid (\exists at \in ann_r) \wedge (\exists at' \in ann'_r) \wedge (d'' = (\mathcal{P}(d), \mathcal{P}(d'), h'')) \wedge (h'' = (d.\mathrm{H} \parallel d'.\mathrm{H})) \wedge (t = t') \wedge ([L''_t, U''_t] = [L_t, U_t] \otimes_\alpha [L'_t, U'_t])\}$. □

Note that cartesian products only combine annotated tuples which refer to the same time point. It computes the combined data tuple d'' by merging (i) manifest data fields from ann_r (i.e., $\mathcal{P}(d)$), (ii) manifest data fields from ann'_r (i.e., $\mathcal{P}(d')$), and (iii) $h'' = d.\mathrm{H} \parallel d'.\mathrm{H}$ (i.e., the hidden list concatenation of $d.\mathrm{H}$ and $d'.\mathrm{H}$). It then computes the combined probability interval by applying user selected conjunction strategy α. This is highly appropriate because when computing Cartesian Products, we are looking at the probability that the concatenation of the two data tuples is in the (ordinary set theoretic) cartesian product of the two relations at a given instant of time. This is therefore a conjunctive event, and hence, the use of a conjunctive p-strategy when performing cartesian products.

For example, $\mathrm{ANN}(r_1) \times_{ig} \mathrm{ANN}(r_2)$ will be

r_1.Data	r_2.Data	H	Day	Month	Year	L_t	U_t
D1	D1		2	8	1997	0.00	0.22
D1	D1		3	8	1997	0.00	0.11
D1	D1		6	8	1997	0.00	0.20
D1	D1		7	8	1997	0.00	0.20
D1	D1		8	8	1997	0.00	0.20

but $\mathrm{ANN}(r_1) \times_{pc} \mathrm{ANN}(r_2)$ will be

r_1.Data	r_2.Data	H	Day	Month	Year	L_t	U_t
D1	D1		2	8	1997	0.10	0.22
D1	D1		3	8	1997	0.05	0.11
D1	D1		6	8	1997	0.10	0.20
D1	D1		7	8	1997	0.10	0.20
D1	D1		8	8	1997	0.10	0.20

10.7 Projection on an annotated relation

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 10.7.1 (projection on an annotated relation) *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 $ann''_r = \{(d'', t, L_t, U_t) \mid (\exists at \in ann_r) \wedge (d'' = (\pi_{\mathcal{F}}(\mathcal{P}(d)), h'')) \wedge (h'' = (d.\mathrm{H} \parallel$ "A_1:$d.\mathrm{A}_1$,..., A_n:$d.\mathrm{A}_\mathrm{n}$"$))\}$.* □

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 and it gracefully ignores fields in $\mathcal{F}$ which do not appear in $\mathcal{P}(d)$'s schema.

For example if $\mathcal{F}$ = "Data1" and if our primary key for ANN(r_3) was "Data1,Data2", then $ann''_r = \pi_{\mathcal{F}}(\mathrm{ANN}(r_3))$ will be

Data1	H	Day	Month	Year	L_t	U_t
D1	Data2:D2	2	8	1997	0.20	0.40
D1	Data2:D3	2	8	1997	0.30	0.40
D1	Data2:D3	3	8	1997	0.15	0.20

Notice that if we did not have the hidden field h'', then we would not be able to tell whether (D1) refers to event (D1,D2) or to event (D1,D3). In other words, the hidden field helps us to prevent a loss of information. Now suppose that after a projection, we wanted (D1) to refer to all events where Data1 = D1. For the example above, this would mean that event (D1) should refer to the compound event ((D1,D2) $\vee$ (D1,D3)). This interpretation is not directly supported by our algebras since our framework only allows instantaneous events.

However, our algebra is rich enough to express it indirectly by (i) setting each h'' in ann''_r to EMPTY and then (ii) invoking a disjunctive p-strategy based compaction (Section 10.1) on the result of step (i). The resulting annotated relation may be inconsistent. All tuples in the result of this operation denote instantaneous events.

To help reduce the size of the hidden field, projection only retains field-value pairs for fields which appear in the relation's primary key. Thus when the primary key is small, h'' will also be small.

10.8 Join of two annotated relations

For simplicity, this paper will only consider the "natural join" operation.

Definition 10.8.1 (join of two annotated relations) *Let selection condition $\mathcal{C}$ be defined as $((ann_r.\mathcal{L}_1 = ann'_r.\mathcal{L}_1) \wedge \ldots \wedge (ann_r.\mathcal{L}_n = ann'_r.\mathcal{L}_n))$ where "$\mathcal{L}_1 \ldots \mathcal{L}_n$" is the list of all manifest data fields which occur in the schema for both ann_r and ann'_r. Then the* join of annotated relations *ann_r and ann'_r under the α probabilistic conjunction strategy, denoted $ann_r \bowtie_\alpha ann'_r$, is defined as $\pi_{\mathcal{F}}(\sigma_{\mathcal{C}}(ann_r \times_\alpha ann'_r))$ where $\mathcal{F}$ is the list of all manifest data fields which occur in the schema for either ann_r or ann'_r after removing duplicate field names.* □

For example, $ann''_r = \mathrm{ANN}(r_3) \bowtie_{pc} \mathrm{ANN}(r_4)$ will be

Data1	Data2	H	Day	Month	Year	L_t	U_t
D1	D2		2	8	1997	0.10	0.40
D1	D3		3	8	1997	0.15	0.20

Notice that all of the hidden fields in ann''_r above are EMPTY. This occurs since in our example, $\mathcal{F}$ = "Data1,Data2" so when we perform a projection, the list of manifest data fields not appearing in $\mathcal{F}$ (i.e., the "$A_1, \ldots, A_n$" list in Definition 10.7.1) is empty.

Although our definition of join in this section only corresponds to a natural join, it can easily be extended to handle other types of join. For instance, an implementation which uses an SQL-like interface may allow users to explicitly specify appropriate values for $\mathcal{C}$ and $\mathcal{F}$.

10.9 Granularity of operations in TATA.

Complex expressions in our algebra can be used to perform operations involving a number of different combination functions. For instance, a user who wants to take a join of two annotated relations ann_{r_1} and ann_{r_2} assuming independence on the first day of each month and positive correlation on all other days may do so by writing a complex query. First, he selects from each relation, all first days of the month, leading to relations $ann_{r_1^1}$ and $ann_{r_2^1}$ respectively. He also selects from each of the two annotated relations ann_{r_1} and ann_{r_2} all tuples not dealing with the first day of the month, leading to relations $ann_{r_1^2}$ and $ann_{r_2^2}$ respectively. He now takes the join of $ann_{r_1^1}$ and $ann_{r_2^1}$ using independence, and the join of $ann_{r_1^2}$ and $ann_{r_2^2}$ using positive correlation, and then takes the multiset union of the two.

The TATA expression describing such a join will be

$$(\sigma_{day=1}(ann_{r_1}) \bowtie_{ind} \sigma_{day=1}(ann_{r_2}) \cup \sigma_{day\neq 1}(ann_{r_1}) \bowtie_{pc} \sigma_{day\neq 1}(ann_{r_2})).$$

Similarly, if different combination functions are desired for compacting different parts of a TP-relation, complex expressions in TATA may be used. For example, $\kappa_{\chi_1}(\sigma_{month=10}(r)) \cup \kappa_{\chi_2}(\sigma_{month\neq 10}(r))$ specifies that compaction for parts of r referring to the month of October should be performed under a different combination strategy than the compaction for the rest of the relation.

In general, the person querying the data must - using his knowledge of the data - specify the combination strategies and/or dependencies he plans to use. However, the algebraic operations - as shown above - allow him to express queries taking into account, dependencies he has identified, even if different types of dependencies apply to different parts of the data. With sufficient amount of information provided one can construct TATA expressions which would employ different assumptions for different tp-tuples (i.e., make the granularity of the assumption to be a single tuple). However, there expressions will naturally be long.

Similar comments apply to the TP-algebra defined below.

Chapter 11

Temporal Probabilistic Algebra (TP-Algebra, TPA)

As mentioned several times thus far, for every data-tuple d and every time-point t, the TATA algebra *explicitly* represents the probability that a data-tuple d is in a given relation at time t. As pointed out by Dyreson and Snodgrass[35], this leads to a completely unacceptable explosion in the size of annotated relations and leads to major scalability problems. In this section, we will show that TP-relations, which implicitly and succinctly represent temporal probabilistic data, can be very efficiently manipulated by algebraic operations that correctly implement (as defined below) all the operations on the TP-algebra. In other words, we can use the TP-representation to efficiently implement the TATA algebra operations.

In this chapter, we provide definitions for TP-compression, compaction, intersection, union, selection, difference, cartesian product, projection, join, and tightening of TP-relations. With the exception of TP-compression and tightening (which have no analogs in TATA), we will show that each of these operations *correctly implements* the corresponding operation in the TATA. The advantage is immediate: as TP-relations are relatively small when compared to their annotated counterparts, a huge savings, both in space (of storing TP- vs. annotated relations) and time (in terms of the time to process these operations) will result.

Definition 11.0.1 (correctly implements) *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* $\mathrm{ANN}(r\ op^T\ r') = \mathrm{ANN}(r)\ op^A\ ann'_r$ *for every pair of TP-relations r, r'.* □

Note that as usual, intersection, union, and difference are only defined when both TP-relations have the same schema, selections are only defined when $\mathcal{C}$ is not an inapplicable condition, and projections are only defined when field list $\mathcal{F}$ is projectable.

The definitions in this section will produce a new TP-relation r'' based on input from consistent TP-relations r, r'. Oftentimes, these definitions will refer to TP-tuples tp, tp' which are assumed to be of the form $tp = (d, \gamma)$ and $tp' = (d', \gamma')$. Here, let γ contain n TP-cases of the form $\gamma_i = \langle C_i, D_i, L_i, U_i, \delta_i \rangle \in \gamma$ and let γ' contain n' TP-cases of the form $\gamma'_j = \langle C'_j, D'_j, L'_j, U'_j, \delta'_j \rangle \in \gamma'$.

11.1 TP-compression of a TP-relation

Let $N(r)$ and $N(tp)$ denote the number of TP-cases in TP-relation r and TP-tuple tp respectively. Intuitively, when we apply a TP-compression function to r, we may be able to decrease $N(r)$ and thus store temporal probabilistic information more compactly.

Definition 11.1.1 (TP-compression function) *A* TP-compression function $\Xi(r)$ *is a function which takes TP-relation r as input, and returns as output a TP-relation r'' where* (i) $N(r'') \leq N(r)$ *and* (ii) *there exists a bijection between $ANN(r)$ and $ANN(r'')$ which maps each $(d, t, L_t, U_t) \in ANN(r)$ to a $(d, t, L_t, U''_t) \in ANN(r'')$ such that $L_t \leq U''_t \leq U_t$.* □

In other words, $ANN(r'')$ and $ANN(r)$ must have the same data tuples, time points, and lower bounds, but we allow TP-compressions to tighten upper bounds. Note that there are many functions which satisfy the definition of a TP-compression function given above. For instance, the following TP-compression function combines TP-cases which share the same distribution.

Definition 11.1.2 (TP-compression of a TP-relation; same-distribution) *The* same-distribution TP-compression *of TP-relation r, denoted $\Xi^{sd}(r)$, is equal to the multiset S which can be constructed in the following way: Initially, let $S = r$. Then for each $(d, \gamma'') \in S$ and for each pair of TP-cases $\gamma_i = \langle C_i, D_i, L, U, \delta\rangle, \gamma_j = \langle C_j, D_j, L, U, \delta\rangle \in \gamma''$, where $\mathrm{sol}(D_i) = \mathrm{sol}(D_j)$, remove γ_i, γ_j from γ'' and add TP-case $\langle (C_i \vee C_j), D_i, L, U, \delta\rangle$ to γ''.* □

Another possible TP-compression function takes advantage of the uniform distribution's regularity.

Definition 11.1.3 (TP-compression of a TP-relation; u-based) *The* u-based TP-compression of TP-relation *r, denoted $\Xi^{u}(r)$, is equal to the multiset S which can be constructed in the following way: Initially, let $S = r$. Then for each $(d, \gamma'') \in S$ and for each pair of TP-cases $\gamma_i = \langle C_i, D_i, L_i, U_i, u\rangle, \gamma_j = \langle C_j, D_j, L_j, U_j, u\rangle \in \gamma''$ where $n_i = |\mathrm{sol}(D_i)|$, $n_j = |\mathrm{sol}(D_j)|$, and $([L_t, U_t] = \frac{1}{n_i} \cdot [L_i, U_i] = \frac{1}{n_j} \cdot [L_j, U_j])$, remove γ_i, γ_j from γ'' and add TP-case $\langle (C_i \vee C_j), (D_i \vee D_j), L_{ij}, \min(1, U_{ij}), u\rangle$ to γ'' where $n_{ij} = |\mathrm{sol}(D_i \vee D_j)|$ and $[L_{ij}, U_{ij}] = n_{ij} \cdot [L_t, U_t]$.* □

Note that when $U_{ij} > 1$, the upper bounds in ANN(r'') will be tighter than the ones in ANN(r).

Definition 11.1.4 (TP-compression of a TP-relation; hybrid) *The* hybrid TP-compression of TP-relation *r, denoted $\Xi^{hy}(r)$, is defined as $\Xi^{sd}(\Xi^{u}(r))$.* □

The following theorem indicates that the above functions satisfy our definition for a TP-compression.

Theorem 29 *$\Xi^{sd}(r)$, $\Xi^{u}(r)$ and $\Xi^{hy}(r)$ are all TP-compression functions.*

Proof.

To see that $\Xi^{sd}(r)$ is a TP-compression function, we must demonstrate that $N(\Xi^{sd}(r) \leq N(r)$ and that there exists a bijection, ϕ from $ANN(r)$ to $ANN(sd_(r))$. In fact we will show that a stronger statement holds: $ANN(r) = ANN(sd_(r))$.

1. $N(\Xi^{sd}(r)) \leq N(r)$.

   Let $r = \{tp_1, \ldots, tp_s\}$. Clearly, $N(r) = N(tp_1) + \ldots + N(tp_s)$.

   By definition of Ξ^{sd}, for every tp-tuple $tp \in r$, exactly one tp-tuple tp' is added to $\Xi^{sd}(r)$. Therefore $|r| = |\Xi^{sd}(r)|$. Let now $\Xi^{sd}(r) = \{tp'_1, \ldots, tp'_s\}$ where $tp'_i = \Xi^{sd}(tp_i)$, $1 \leq i \leq s$. As $N(\Xi^{sd}(r)) = N(tp'_1) + \ldots + N(tp'_s)$, it suffices to show that for all $1 \leq i \leq s$ $N(tp'_i) \leq N(tp_i)$.

   The latter statement is clearly true as by definition of Ξ^{sd} either $tp_i = tp'_i$ (in the case when no tp case statements were deleted) or, for every two tp-case statements deleted from the tp-tuple tp_i during its conversion into tp'_i only one tp-case statement is added.

2. $ANN(r) = ANN(\Xi^{sd}(r))$.

   Let $tp \in r$. We will show that $ANN(tp) = ANN(\Xi^{sd}(tp))$.

   Let $tp' = \Xi^{sd}(tp)$. Let $tp = (d, \gamma), \gamma = \gamma_1, \ldots, \gamma_n, \gamma_i = \langle C_i, D_i, L_i, U_i, \delta_i, 1 \leq i \leq n$ and let $tp' = (d, \gamma'), \gamma' = \gamma'_1, \ldots, \gamma'_m, \gamma'_i = \langle C'_i, D'_i, L'_i, U'_i, \delta'_i, 1 \leq i \leq m$.

   Consider some tp-case γ'_j from tp'. By the definition of Ξ^{sd}, one of two cases is possible:

(a) There exists some $1 \leq k \leq n$, $\gamma'_j = \gamma_k$ (i.e., during the application of Ξ^{sd} tp-case γ_k did not change). In this case clearly $ANN((d, \gamma_k)) = ANN((d, \gamma'_j))$.

(b) γ'_j is not equal to any $\gamma_i \in \gamma$, but, there exist $1 \leq k_1, \ldots k_q \leq n$ such that $D_{k_1} = \ldots = D_{k_q} = D'_j$, $L_{k_1} = \ldots = L_{k_q} = L'_j$, $U_{k_1} = \ldots = U_{k_q} = U'_j$, $\delta_{k_1} = \ldots = \delta_{k_q} = \delta'_j$ and $C_{k_1} \vee \ldots \vee C_{k_q} C'_j$.

Let now $at = (d, t, L, U) \in ANN(d, (\gamma_{k_1}, \ldots, \gamma_{k_2}))$. Clearly, there exists some $1 \leq h \leq q$, such that $t \in sol(C_{k_h})$ and $[L, U] = [\delta_{k_h}(D_{k_h}, t) \cdot L_{k_h}, \delta_{k_h}(D_{k_h}, t) \cdot U_{k_h}]$. Since $t \in sol(C_{k_h})$, $t \in sol(C_{k_1} \vee \ldots \vee C_{k_q}) = sol(C'_j)$. Therefore, $at' = (d, t, \delta'_j(D'_j, t) \cdot L'_j, \delta'_j(D'_j, t) \cdot U'_j) \in ANN(d, \gamma'_j)$. But since $D_{k_h} = D'_j$, $L_{k_h} = L'_j$, $U_{k_h} = U'_j$ and $\delta_{k_h} = \delta'_j$, $[L, U] = [\delta'_j(D'_j, t) \cdot L'_j, \delta'_j(D'_j, t) \cdot U'_j]$ and therefore $at = at'$. Hence, $ANN(d, (\gamma_{k_1}, \ldots, \gamma_{k_2})) \subseteq ANN(d, \gamma'_j)$.

The proof that $ANN(d, (\gamma_{k_1}, \ldots, \gamma_{k_2})) \supseteq ANN(d, \gamma'_j)$ is analogous.

This establishes that $ANN(d, (\gamma_{k_1}, \ldots, \gamma_{k_2})) = ANN(d, \gamma'_j)$.

Combining the above results, we conclude that $ANN(tp) \supseteq ANN(\Xi^{sd}(tp))$.

To show that $ANN(tp) \subseteq ANN(\Xi^{sd}(tp))$, it suffices to notice that for any $1 \leq i \leq n$ and tp-case γ_i there exists such $1 \leq j \leq m$ and tp-case γ'_j that $sol(C_i) \subseteq sol(C'_j)$.

The proofs for $\Xi^u(r)$ and $\Xi^{hy}(r)$ are similar. $\square$

More sophisticated TP-compression operators are also possible. For instance, let $p \in (0, 1)$ be a probability and let $t_1, \ldots, t_n$ be a list of (consecutive) time points in S_τ where for each $1 \leq i < n$, $t_{i+1} = next_\tau(t_i)$. Then if γ'' contains n TP-cases of the form $\langle (\#), (t_i), L''_{t_i}, U''_{t_i}, u \rangle$ where ($[L''_{t_{i+1}}, U''_{t_{i+1}}] = p \cdot [L''_{t_i}, U''_{t_i}]$ for all $1 \leq i < n$) and ($U''_{t_1} \div p \leq 1$), apply an operator which replaces these n TP-cases with the TP-case $\langle (\#), (t_1 \sim$

$t_n), L'', U'', \delta''\rangle$ where $[L'', U''] = [L''_{t_1} \div p, U''_{t_1} \div p]$ and $\delta'' =$ "g, p".

The aforementioned operator performs a g-based TP-compression. Note that for any distribution function δ, one can define a corresponding δ-based TP-compression operator. Also note that one can obtain optimal TP-compression (i.e., the smallest possible value for $N(r''))$ by allowing a TP-compression operator to dynamically create new distribution functions which fit the resulting data.

11.2 Compaction of a TP-relation

As in the case of the TATA, the compaction operation in the TPA will be used to define the operations of intersection, union and projection.

Definition 11.2.1 (Compaction of a TP-relation) *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* $\mathrm{ANN}(\kappa(r)) = \mathrm{ANN}(r)$.
- Conservativeness : *If* $at = (d, t, L_t, U_t) \in \mathrm{ANN}(\kappa(r))$, *then* $\exists at' = (d, t, L'_t, U'_t) \in \mathrm{ANN}(r)$. □

The Compactness, NFA, and Conservativeness axioms for TP-relations are similar in spirit and intuition to the same concepts defined earlier for annotated relations. However, one difference is that while each annotated relation has a unique compaction, given an annotated relation ann_r, there may be more than one TP-relation r such that $ann_r = ANN(r)$. Hence, there may be more than one TP-relation r'' which is a compaction of r. **Any such tp-relation r'' is a valid result of compaction**. We will develop and present a compaction algorithm below, which constructs a compaction of a TP-relation.

There are many different compaction operations on TP-relations in TPA (similarly to TATA). Below, we present the TP analogs of χ-compactions and p-strategy based compactions.

Definition 11.2.2 (χ-compaction of a TP-relation) *Let χ be a combination function. Then the* χ-compaction of TP-relation r*, denoted $\kappa_\chi(r)$, is defined as*

$$ANN(\kappa_\chi(r)) = \{(d, t, L_t, U_t) \mid [L_t, U_t] = \chi(\{[L_1^{(d,t)}, U_1^{(d,t)}], \ldots, [L_k^{(d,t)}, U_k^{(d,t)}]\})\},$$

where $(d, t, L_t, U_t) = at$ and $ANN(r)[d, t] = \{at_1^{(d,t)}, \ldots at_k^{(d,t)}\}$ and $at_i^{(d,t)} = (d, t, L_i^{(d,t)}, U_i^{(d,t)})$. □

The following lemma states that an operation defined in this manner is indeed a compaction operation.

Proposition 15 *Let χ be a combination function. Then $\kappa_\chi(r)$ is a compaction operation.*

Proof. Similar to the proof of Proposition 13. □

Note that some parts of Definitions 11.2.1 and 11.2.2 were presented in a declarative manner — via the contents of the annotation of the result. Algorithm Compute-Compaction shown below provides a mechanism to efficiently compute compactions without resorting to annotation. This algorithm can perform compactions using either a combination function χ or a p-strategy ρ. The boxed line in this algorithm shows exactly where a combination function or p-strategy is applied to compact data-identical tuples. Note that when f is a p-strategy, the application of f to a set X of intervals merely represents the iterative application of f to pairs of intervals in X — as p-strategies are associative and commutative, this is well defined.

Algorithm 7 Algorithm Compute-Compaction(r,f)

Input: TP-relation r and combination function or p-strategy f
Output: TP-relation $r'' = \kappa_f(r)$
01. $r'' := \emptyset$; // Initialize the resulting relation
02. $r' := r$; // Obtain a working copy of the initial relation
03. // For each (maximal) multiset S of data-identical TP-tuples in r
04. **while** $(r' \neq \emptyset)$ **do** $\{$
05. Select a TP-tuple $tp \in r'$;
06. $S := r'[tp]$; // Extract the next equivalence class from r'
07. $r' := r' - S$; $\gamma'' := \emptyset$;
08. $\Gamma_S := \biguplus_{(d,\gamma)\in S} \gamma$; $\Gamma_I := \Gamma_S$;
09. **foreach** $\gamma_i = \langle C_i, D_i, L_i, U_i, \delta_i \rangle \in \Gamma_I$ $\{$
10. Remove γ_i from Γ_I and Γ_S; $C_S := \bigvee_{\langle C,D,L,U,\delta\rangle \in \Gamma_S} C$;
11. **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'';
12. **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; $\}$
13. $C_S := \bigvee_{\langle C,D,L,U,\delta\rangle \in \Gamma_S} C$;
14. // Note: Each $t \in \mathrm{sol}(C_S)$ will refer to more than one TP-case
15. **foreach** $t \in \mathrm{sol}(C_S)$ $\{$
16. $X := \emptyset$; // X will contain the probability intervals to be combined
17. $\Gamma_t := \{\langle C, D, L, U, \delta \rangle \in \Gamma_S \mid t \in \mathrm{sol}(C)\}$;
18. **foreach** $\langle C, D, L, U, \delta \rangle \in \Gamma_t$ $\{$
19. $x_t := \delta(D, t)$; $[L_t, U_t] := [L \cdot x_t, U \cdot x_t]$;
20. $X := X \cup \{[L_t, U_t]\}$; $\}$
21. $\boxed{[L''_t, U''_t] := f(X);}$
22. Add TP-case $\langle (\#), (t), L''_t, U''_t, u \rangle$ to γ''; $\}$
23. Add TP-tuple (d, γ'') to r''; $\}$
24. **return** r'';
End-Algorithm

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

Theorem 30 *Let χ be a combination function.*

Then Algorithm Compute-Compaction(r, χ) *correctly computes the $\kappa_\chi(r)$ compaction operation.*

Proof.

Let us look at the relation r'' which is returned by Compute-Compaction(r, χ). The main loop of the algorithm (lines 04.–23.) works in the following way:

- In lines 05.–08. of the algorithm the initial relation r is broken into the equivalence classes by the relation of data-identicity. Lines 05.–0.6 select the next unprocessed equivalence class, and lines 07.–08. prepare the data for the next step. In particular, Γ_I (and initially Γ_S) are set to be equal to the set of all tp-cases found in the tp-tuples of the current equivalence class.

- The **foreach** loop in lines 09.–12. then separates all timepoints which are in solution of some tp-case $\gamma_i \in \Gamma_I$ into two categories: timepoints that are referred to only by γ and timepoints referred to by more than one tp-case. The timepoints referred to only by one tp-case are handled by line 11. in which a new tp-case that contains only such timepoints from γ_i and adds it to the final tp-tuple.

- Line 13. collects together in one temporal constraint C_S all the timepoints that are referred to by more than one original tp-case.

- In the **foreach** loop in lines 15.–22. each timepoint t from $sol(C_S)$ gets processed. Lines 17.–20. collect together the set X of all intervals $[L, U]$ which are the probabilities associated with the timepoint t by all original tp-cases which refer to t.

- In line 21. the combination function χ is applied to the set X to obtain the interval $[L''_t, U''_t]$, which will become the part of the tp-case $((t), (t), L''_t, U''_t, u)$ which is added to the final tp-tuple tp''.

- In line 23. the tp-tuple tp'' constructed for the current equivalence class is added to the resulting relation r''.

Now, let some annotated tuple $at = (d, t, L, U) \in ANN(r'')$. Then there exists a tp-tuple $tp \in r''$, such that $at \in ANN(tp)$. Let us examine the values of L and U.

Since $at \in ANN(tp)$, $tp = (d, \gamma)$, $\gamma = \gamma_1, \dots \gamma_n$, $\gamma_i = \langle C_i, D_i, L_i, U_i, \delta_i \rangle$, $1 \le i \le n$ and there exists $1 \le j \le n$, $t \in sol(C_j)$ and $[L, U] = [\delta_j(D_j, t) \cdot L_j, \delta_j(D_j, t) \cdot U_j]$.

Let us look at how tp-case γ_j could have been added to tp. From the analysis of the algorithm above, it is clear that there are only two possiblilties:

1. γ_i had been added to tp in line 11. Let us look at the equivalence class of tp in r (although tp itself is not in r it must be data-identical to some tp-tuples of r and hence $r[tp]$ makes sense even if $tp \notin r$). After line 08. is executed while the algorithm is processing $r[tp]$, Γ_I will contain the (multi)set of all tp-cases from the tuples in $r[tp]$. Since γ_i had been added to tp in line 11., it means that there exists such a tp-case $\gamma' = \langle C', D', L', U' \rangle \in \Gamma_I$[1] that $sol(C_j) \subseteq sol(C')$, $D_j = D'$, $L_j = L'$, $U_j = U'$ and $\delta_j = \delta'$. Also this means that no other tp-case in Γ_I refers to any timepoints in $sol(C_j)$, in particular, no other tp-case in Γ_I refers to timepoint t. Clearly then $ANN(r)[d, t] = ANN(r[tp])[d, t]$ will contain one and only one annotated tuple: $at' = (d, t, \delta(D', t) \cdot L', \delta(D', t) \cdot U')$. But from the equalities established above, $at' = (d, t, \delta_j(D_j, t) \cdot L_j, \delta_j(D_j, t) \cdot U_j) = (d, t, L, U) = at$. Now we notice that $\chi(\{[L, U]\}) = [L, U]$ as it is required by the definition of χ-compaction, since χ as a combination function satisfies Identity postulate.

2. γ_i had been added to tp in line 22. In this case in the set Γ_I built in line 08. for the equivalence class of tp in r there will be more than one tp-case which refers to t. From the analysis of the algorithm above we conclude that in this case $t \in sol(C_S)$ after C_S had been computed in line 13. Let us consider the iteration of the **foreach** loop 15.–22. which processes t. In line 17. Γ_t gets assigned the value of the set of all tp-cases $\gamma_i = \langle C'_i, D'_i, L'_i, U'_i, \delta'_i \rangle$ that $t \in sol(C'_i)$. **foreach** loop in lines 18.–20

[1] here and further when we mention Γ_I we refer to its initial value assigned in line 08.

builds the set $X = \{[L_i^*, U_i^*]\}$ where $[L_i^*, U_i^*] = [\delta_i(D_i', t) \cdot L_i', \delta_i(D_i', t) \cdot U_i']$ are the probabilistic intervals assigned to d, t by each tp-case γ_i.

In line 21. $[L, U] = \chi(X)$ is computed. Now, all we have to prove is that $X = \{[L_1^{(d,t)}, U_1^{(d,t)}], \ldots, [L_k^{(d,t)}, U_k^{(d,t)}]\}$ such that $\mathrm{ANN}(r)[d, t] = \{at_1^{(d,t)}, \ldots at_k^{(d,t)}\}$ and $at_i^{(d,t)} = (d, t, L_i^{(d,t)}, U_i^{(d,t)})$. Let $\Gamma_t = \{\gamma_1', \ldots \gamma_s'\}$. Since Γ_t contains all tp-cases of $r[tp]$ (and hence, all tp-cases of r) which refer to timepoint t for the data d, $ANN(r)[d, t] = ANN(r[tp])[d, t]$ will be equal to the set: $\{at_1', \ldots at_s'\}$, where $at_i' = (d, t, \delta_i(D_i, t) \cdot L_i, \delta_i(D_i, t) \cdot U_i) = (d, t, L_i^*, U_i^*), 1 \leq i \leq s$. But then we also know that $X = \{[L_1^*, U_1^*], \ldots [L_1^*, U_1^*]\}$. This proves the theorem. □

We define p-strategy based compactions of TP-relations in the same way as $\kappa_\chi(r)$ except we let $[L_t, U_t] = ([L_1^{(d,t)}, U_1^{(d,t)}] \otimes \ldots \otimes [L_k^{(d,t)}, U_k^{(d,t)}])$ and let $[L_t, U_t] = ([L_1^{(d,t)}, U_1^{(d,t)}] \oplus \ldots \oplus [L_k^{(d,t)}, U_k^{(d,t)}])$ when defining $\kappa_\otimes(r)$ and $\kappa_\oplus(r)$ respectively.

Proposition 16 *Let ρ be a p-strategy. Then $\kappa_\rho(r)$ is a compaction operation.*

Proof. Similar to the proof of Proposition 14. □

As p-strategy based compaction of TP-relations is defined declaratively, we need an explicit algorithm (mentioned above) to compute it. The following result states the correctness of this algorithm.

Theorem 31 *Let ρ be a (conjunctive or disjunctive) p-strategy. Then Algorithm Compute-Compaction(r, ρ) correctly computes the $\kappa_\rho(r)$ compaction operation.*

Proof.

Similar to the proof of Theorem 30. □

Thus far, we have separately defined compaction operators on annotated relations and on TP-relations. From here and until the end of this section we will use superscript ^{A} to denote compaction operators for annotated relations, and superscript ^{T} to denote compaction operators for TP-relations. The following definition specifies when a compaction operator on the annotated side corresponds to a compaction operator on the TP-side.

Definition 11.2.3 (compatible pair of compactions) *A pair $\langle \kappa^A(.), \kappa^T(.)\rangle$, where κ^A is a compaction operator on annotated relations and κ^T is a compaction operator on TP-relations is a* compatible pair *iff for every TP-relation r, $\kappa^A(\mathrm{ANN}(r)) = \mathrm{ANN}(\kappa^T(r))$.*

□

The following two theorems say that for any arbitrary combination function χ and for any arbitrary p-strategy ρ, $\langle \kappa^A_\chi(.), \kappa^T_\chi(.)\rangle$ and $\langle \kappa^A_\rho(.), \kappa^T_\rho(.)\rangle$ are compatible pairs.

Theorem 32 *Let χ be any combination function. Then $\langle \kappa^A_\chi(.), \kappa^T_\chi(.)\rangle$ is a compatible pair.*

Proof.

By definition 11.2.2,

$ANN(\kappa^T_\chi(r)) = \{at = (d, t, L, U) | [L, U] = \chi(\{[L_1,^{(d,t)}, U_1,^{(d,t)}], \ldots,$

$[L_k,^{(d,t)}, U_k,^{(d,t)}]\})$, where $(\forall 1 \leq i \leq k)(at_i^{(d,t)} = (d, t, L_i^{(d,t)}, U_i^{(d,t)})$ and

$\{at_1^{(d,t)}, \ldots at_k^{(d,t)}\} = ANN(r)[d, t]\}$. But by definition 10.1.5 $\kappa^A_\chi(ANN(r))$ is equal to the same expression. Therefore,

$ANN(\kappa^T_\chi(r)) = \kappa^A_\chi(ANN(r))$. □

Theorem 33 *Let ρ be any p-strategy. Then $\langle \kappa^A_\rho(.), \kappa^T_\rho(.)\rangle$ is a compatible pair.*

Proof.

By the definition of a p-strategy based compaction of TP-relations $ANN(\kappa_\rho^T(r)) =$ $\{at = (d, t, L, U) | [L, U] = [L_1^{(d,t)}, U_1^{(d,t)}] *_\rho \ldots *_\rho [L_k^{(d,t)}, U_k^{(d,t)}],$ where $(\forall 1 \leq i \leq k)(at_i^{(d,t)} = (d, t, L_i^{(d,t)}, U_i^{(d,t)}); \{at_1^{(d,t)}, \ldots at_k^{(d,t)}\} = ANN(r)[d, t]\}$ and $* = \oplus$ if ρ is a disjunctive p-strategy and $* = \otimes$ if ρ is a conjunctive p-strategy.

But by the definition of a p-strategy based compaction of annotated relations $\kappa_\chi^A(ANN(r))$ is equal to the same expression.

Therefore, $ANN(\kappa_\rho^T(r)) = \kappa_\rho^A(ANN(r))$. □

11.3 Intersection of two TP-relations

In this section, we show how we can correctly implement the intersection of two TP-relations. Intersection consists of of two suboperations — multiset intersection and combination function based compaction.

Definition 11.3.1 (multiset intersection of two TP-relations) *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.* □

For example, $r'' = r_1 \cap r_2$ will be

Data	H	C	D	L	U	δ
D1		$(2/8/1997 \sim 3/8/1997)$	$(1/8/1997 \sim 3/8/1997)$	0.64	0.88	g
		$(6/8/1997 \sim 8/8/1997)$	$(5/8/1997 \sim 8/8/1997)$	0.40	0.80	u
D1		$(2/8/1997 \sim 3/8/1997)$	$(2/8/1997 \sim 3/8/1997)$	0.20	0.50	g
		$(6/8/1997 \sim 8/8/1997)$	$(6/8/1997 \sim 9/8/1997)$	0.40	0.80	u

As in the case of the TATA, we apply a κ_χ compaction operator to the result of a multiset intersection.

Definition 11.3.2 (intersection of two TP-relations) *The* intersection of TP-relations r and r' under the χ combination function*, denoted* $r \cap_\chi r'$*, is defined as* $\kappa_\chi(r \cap r')$. $\square$

In order to keep the size of $r \cap_\chi r'$ manageable, we usually perform a TP-compression on the result of a compaction.

For example, $\Xi^{hy}(r \cap_{eq} r') = \Xi^{hy}(\kappa_{eq}(r''))$ will be

Data	H	C	D	L	U	δ
D1		$(\#)$	$(2/8/1997 \sim 2/8/1997)$	0.16	0.22	u
		$(\#)$	$(3/8/1997 \sim 3/8/1997)$	0.08	0.11	u
		$(\#)$	$(6/8/1997 \sim 8/8/1997)$	0.30	0.60	u

The following theorem shows that our definition of intersection correctly implements the TATA semantics. This lets us completely avoid the construction of the (huge) annotated expansion while preserving the same semantics.

Theorem 34 (Correctness of intersection) $ANN(r \cap_\chi r') = ANN(r) \cap_\chi ANN(r')$.

Proof.

By definition $r \cap_\chi r' = \kappa_\chi(r \cap r')$. Our proof consists of three parts. We show that:

1. $ANN(r \cap r') \subseteq ANN(r) \cap ANN(r')$.

   Let $at'' = (d, t, L, U) \in ANN(r \cap r')$. We show that $at'' \in ANN(r) \cap ANN(r')$. By definition of ANN, $at'' \in ANN(r \cap r')$ implies that there exists a tp-tuple $tp'' \in r \cap r'$ such that:

   $tp'' = (d, \gamma''); \gamma'' = \gamma''_1, \ldots \gamma''_k$
   $(\forall 1 \leq i \leq k)\, \gamma''_i = \langle C''_i, D''_i, L''_i, U''_i, \delta''_i \rangle$ **and**
   $(\exists i \in \{1, \ldots k\})(t \in sol(C''_i) \wedge [L, U] = [\delta''_i(D''_i, t) \cdot L''_i, \delta''_i(D''_i, t) \cdot U''_i])$.

   As $tp'' \in r \cap r'$, by definition of intersection of two tp-relations, there exist tp-tuples $tp = (d, \gamma) \in r)$ and $tp' = (d, \gamma') \in r')$ where:

$$\begin{aligned}
\gamma &= \gamma_1, \ldots, \gamma_n \\
\gamma' &= \gamma'_1, \ldots, \gamma_{n'} \\
\gamma_i &= \langle C_i, D_i, L_i, U_i, \delta_i \rangle \\
\gamma'_i &= \langle C'_i, D'_i, L'_i, U'_i, \delta'_i \rangle
\end{aligned}$$

   In addition, there exist integers $i_1 \in \{1, \ldots, n\}$ and $i_2 \in \{1, \ldots, n'\}$ such that either

$$\begin{aligned}
C''_i &= (C_{i_1} \wedge C'_{i_2}) \text{ and} \\
D''_i &= D_{i_1} \text{ and} \\
\delta''_i &= \delta_{i_1} \text{ and} \\
[L''_i, U''_i] &= [L_{i_1}, U_{i_1}] \,.
\end{aligned}$$

   or

$$C''_i = (C_{i_1} \wedge C'_{i_2}) \text{ and}$$

$$
\begin{aligned}
D_i'' &= D_{i_2} \text{ and} \\
\delta_i'' &= \delta_{i_2} \text{ and} \\
[L_i'', U_i''] &= [L_{i_2}, U_{i_2}] \,.
\end{aligned}
$$

Without loss of generality, assume that the first case holds (the reasoning for the other case is symmetric). As $t \in sol(C_{i_1})$ and $t \in sol(C'_{i_2})$, by definition of ANN, we may say that $ANN(tp) \ni at = (d, t, L_1, U_1)$ and $ANN(tp) \ni at' = (d, t, L_2, U_2)$ where $[L_1, U_1] = [\delta(D_{i_1}, t) \cdot L_{i_1}, \delta(D_{i_1}, t) \cdot U_{i_1}]$ and $[L_2, U_2] = [\delta(_{i_2}' D_{i_1}, t) \cdot L_{i_1}, \delta'_{i_2}(D_{i_2}, t) \cdot U_{i_2}]$ Therefore, by definition of intersection of two annotated relations, we conclude that $\{at, at'\} \subseteq ANN(r) \cap ANN(r')$. It follows that $at'' = (d, t, L, U) = at = (d, t, L_{i_1}, U_{i_1})$, because $[L, U] = [L_{i_1}, U_{i_1}]$.

2. $ANN(r \cap r') \supseteq ANN(r) \cap ANN(r')$.

   Suppose $at = (d, t, L, U) \in ANN(r) \cap ANN(r')$. Hence, by definition of intersection there exists an annotated tuple $at' = (d, t, L', U') \in ANN(r) \cap ANN(r')$, such that

   - either $at \in ANN(r)$; $at' \in ANN(r')$
   - or $at' \in ANN(r)$; $at \in ANN(r')$

   Without loss of generality we will assume that $at \in ANN(r)$; $at' \in ANN(r')$ (the other case is symmetric).

   As $at \in ANN(r)$ we conclude by definition of ANN that there exists a tp-tuple $tp = (d, \gamma) \in r$ such that $\gamma = \gamma_1, \ldots, \gamma_n$
   $(\forall 1 \leq i \leq n)(\gamma_i = \langle C_i, D_i, L_i, U_i, \delta_i \rangle)$
   and $(\exists i \in \{1, \ldots, n\})(t \in sol(C_i) \wedge [L, U] = [\delta_i(D_i, t)L_i, \delta_i(D_i, t)U_i])$.

Now from $at' \in ANN(r')$, we know that there exists a tp-tuple $tp' = (d, \gamma') \in r'$ such that $\langle tp, tp' \rangle \in m$ and

$\gamma' = \gamma'_1, \ldots, \gamma'_n$

$(\forall 1 \leq i \leq n')(\gamma'_i = \langle C'_i, D'_i, L'_i, U'_i, \delta'_i \rangle)$

and $(\exists j \in \{1, \ldots, n'\})(t \in sol(C'_j) \wedge [L', U'] = [\delta'_j(D'_j, t)L'_j, \delta'_j(D'_j, t)U'_j])$.

As tp and tp' are data identical, there exists such a tp-tuple $tp'' = (d, \gamma'') \in r \cap r'$ that

$\gamma'' = \gamma''_1, \ldots, \gamma''_{n''}$

and $(\exists 1 \leq k \leq n'')(\gamma''_k = \langle C''_k, D''_k, L''_k, U''_k, \delta''_k \rangle \wedge C''_k = (C_i \wedge C'_j) \wedge D''_k = D_i \wedge [L''_k, U''_k] = [L_i, U_i] \wedge \delta''_k = \delta_i)$.

As $t \in sol(C_i)$ and $t \in sol(C'_j)$, $t \in sol(C_i \wedge C'_j) = C''_k$. Therefore, $ANN(t'')$ will contain a tuple $at'' = (d, t, L'', U'')$ where

$[L'', U''] = [\delta''_k(D''_k, t) \cdot L''_k, \delta''_k(D''_k, t) \cdot U''_k] = [\delta_i(D_i, t) \cdot L_i, \delta_i(D_i, t) \cdot U_i]$.

We can see that $[L'', U''] = [L, U]$ and therefore, $at'' = at$ and therefore $at \in ANN(tp'')$, from which it follows that $at \in ANN(r \cap_m r')$.

3. $ANN(r \cap_\chi r') = ANN(r) \cap_\chi ANN(r')$.

   By definition of the intersection of two TP relations $ANN(r \cap_\chi r') = ANN(\kappa_\chi(r \cap r'))$. As $\langle \kappa_\chi(.), \kappa_\chi(\mathrm{ANN}(.)) \rangle$ is a compatible pair,we have $ANN(\kappa_\chi(r \cap r')) = \kappa_\chi(ANN(r \cap r')) = \kappa_\chi(ANN(r) \cap ANN(r')) = ANN(r)_\chi ANN(r')$. $\square$

11.4 Union of two TP-relations

In this section, we show how we can correctly implement the union of two TP-relations. Union consists of two suboperations – multiset union and combination function based compaction.

Definition 11.4.1 (multiset union of two TP-relations) *The* multiset union of TP-relations r *and* r'*, denoted* $r \cup r'$*, is defined as* $r'' = r \uplus r'$. □

Intuitively, r'' will contain all $tp \in r$ and all $tp' \in r'$. For example, $r'' = r_1 \cup r_2$ will be

Data	H	C	D	L	U	δ
D1		$(\#)$	$(1/8/1997 \sim 3/8/1997)$	0.64	0.88	g
		$(\#)$	$(5/8/1997 \sim 8/8/1997)$	0.40	0.80	u
D1		$(\#)$	$(2/8/1997 \sim 3/8/1997)$	0.20	0.50	g
		$(\#)$	$(6/8/1997 \sim 9/8/1997)$	0.40	0.80	u

As in the case of the TATA, we apply a κ_χ compaction operator to the result of a multiset union.

Definition 11.4.2 (union of two TP-relations) *The* union of TP-relations r and r' *under the* χ *combination function, denoted* $r \cup_\chi r'$*, is defined as* $\kappa_\chi(r \cup r')$. □

For example, $\Xi^{hy}(r \cup_{eq} r') = \Xi^{hy}(\kappa_{eq}(r''))$ will be

Data	H	C	D	L	U	δ
D1		$(1/8/1997)$	$(1/8/1997 \sim 3/8/1997)$	0.64	0.88	g
		$(\#)$	$(2/8/1997 \sim 2/8/1997)$	0.16	0.22	u
		$(\#)$	$(3/8/1997 \sim 3/8/1997)$	0.08	0.11	u
		$(\#)$	$(5/8/1997 \sim 9/8/1997)$	0.50	1.00	u

The following shows that our definition of union correctly implements the TATA semantics.

Theorem 35 (Correctness of union) $\mathrm{ANN}(r \cup_\chi r') = \mathrm{ANN}(r) \cup_\chi \mathrm{ANN}(r')$.

Proof.

Let r and r' be two tp-relations and χ be a combination function. By definition of union $r \cup_\chi r' = \kappa_\chi(r \cup r')$ and $ANN(r) \cup_\chi ANN(r') = \kappa_\chi(ANN(r) \cup ANN(r'))$. Since $\kappa_c hi(r)$ and $\kappa_\chi(ANN(r))$ is a compatible pair of compaction operations, it is sufficient to prove that $ANN(r \cup r') = ANN(r) \cup ANN(r')$.

- $ANN(r \cup r') \subseteq ANN(r) \cup ANN(r')$.

  Let $at \in ANN(r \cup r')$. Then there exists a tp-tuple $tp \in r \cup r'$, such that $at \in ANN(tp)$. By definition of multiset union of two tp-relations, either $tp \in r$ or $tp \in r'$. If the former holds, $at \in ANN(r)$ and if the latter holds, $at \in ANN(r')$. In either case, $at \in ANN(r) \cup ANN(r')$ as $ANN(r) \cup ANN(r')$ consists of all annotated tuples in $ANN(r)$ and all annotated tuples in $ANN(r')$.

- $ANN(r \cup r') \supseteq ANN(r) \cup ANN(r')$.

  Let $at \in ANN(r) \cup ANN(r')$. Then, either $at \in ANN(r)$ or $at \in ANN(r)$. Assume the former is true (the other case is symmetric). Then, by def. of annotation operation, there exists a tp-tuple $tp \in r$, such that $at \in ANN(tp)$. But, since by definition of multiset union of two tp-relations $tp \in r \cup r'$, we also get $at \in ANN(r \cup r')$. $\square$

11.5 Selection on a TP-relation

In this section, we show how we can correctly implement selection on a TP-relation. The TP-filter operator defined below will help us handle selections of probabilistic conditions (Section 10.4) on TP-relations.

Definition 11.5.1 (TP-filter) *Let $\gamma_i = \langle C_i, D_i, L_i, U_i, \delta_i \rangle$ be a TP-case, let $\mathcal{C}$ = (F op v) 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. *$sol(C_i'') \subseteq sol(C_i)$*

2. *For each time point $t \in sol(C_i'')$, (x_t op v) must be true when $x_t = \delta_i(D_i, t) \cdot x$*

3. *There is no temporal constraint C_i' where $(sol(C_i') \supset sol(C_i''))$ and C_i' satisfies the previous cases.*

Intuitively, a TP-filter returns a temporal constraint whose solution set consists of all time points $t \in \mathrm{sol}(C_i)$ 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 $t \in sol(C_i)$ satisfies this condition, TP-filter$(\gamma_i, \mathcal{C})$ returns an inconsistent temporal constraint. □

For example if $\gamma_i = \langle (\#), (5/8/1997 \sim 8/8/1997), 0.4, 0.8, g \rangle$ and $\mathcal{C} = (U > 0.15)$, TP-filter$(\gamma_i, \mathcal{C})$ will be $C_i'' = (5/8/1997 \sim 6/8/1997)$ since $(0.4 > 0.15)$ for $5/8/1997$ and $(0.2 > 0.15)$ for $6/8/1997$.

In general, $n = |sol(C_i)|$ may be a large number. With arbitrary distribution functions, this can be problematic since the TP-filter function may have to test all n time points. Fortunately, this problem can be alleviated by exploiting regularities in our distribution functions. For instance if δ_i = "u", then we only need to test one time point $t \in sol(C_i)$; if t should be in $sol(C_i'')$, then $C_i'' = C_i$ or $C_i'' = \emptyset$ otherwise. This "all or none" behavior occurs since each $t \in sol(C_i)$ will have the same probability value after distributing uniformly.

Implementations of TP-filters can also exploit regularities in the geometric PDF by searching $sol(C_i)$ in chronological (or reverse chronological) order and then ending the search after finding the first t which should not be in $sol(C_i'')$. The exact search method to use will, of course, depend on which op is present in $\mathcal{C}$. For instance if op = ($\neq$), it may be cheaper to let C_i' = TP-filter($\gamma_i, \neg\mathcal{C}$) and then return $C_i'' = (C_i \wedge \neg C_i')$.

We are now ready to define selection using atomic selection conditions.

Definition 11.5.2 (Selection on a TP-tuple; atomic condition) *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'' *= 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 11.5.3 (Selection on a TP-relation; atomic condition) *The* selection of atomic condition $\mathcal{C}$ on TP-relation $r = \{tp_1, \ldots, tp_n\}$*, denoted* $\sigma_{\mathcal{C}}(r)$*, is defined as* $(\sigma_{\mathcal{C}}(tp_1) \uplus \ldots \uplus \sigma_{\mathcal{C}}(tp_n))$. □

Note that for all $tp_i, tp_j \in r$, $\sigma_{\mathcal{C}}(tp_i)$ does not affect the results of $\sigma_{\mathcal{C}}(tp_j)$ when computing $\sigma_{\mathcal{C}}(r)$.

For example if $\mathcal{C} = (2/8/1997 \sim 7/8/1997)$, $\sigma_{\mathcal{C}}(r_1)$ will be

Data	H	C	D	L	U	δ
D1		$(2/8/1997 \sim 3/8/1997)$	$(1/8/1997 \sim 3/8/1997)$	0.64	0.88	g
		$(5/8/1997 \sim 7/8/1997)$	$(5/8/1997 \sim 8/8/1997)$	0.40	0.80	u

and $\sigma_{\mathcal{C}}(r_2)$ will be

Data	H	C	D	L	U	δ
D1		$(2/8/1997 \sim 3/8/1997)$	$(2/8/1997 \sim 3/8/1997)$	0.20	0.50	g
		$(6/8/1997 \sim 7/8/1997)$	$(6/8/1997 \sim 9/8/1997)$	0.40	0.80	u

but if $\mathcal{C} = (L \neq 0.10)$, $\sigma_{\mathcal{C}}(r_1)$ will be

Data	H	C	D	L	U	δ
D1		$(\#)$	$(1/8/1997 \sim 3/8/1997)$	0.64	0.88	g

and $\sigma_{\mathcal{C}}(r_2)$ will be

Data	H	C	D	L	U	δ
D1		$(3/8/1997)$	$(2/8/1997 \sim 3/8/1997)$	0.20	0.50	g

We can extend selection to handle non-atomic selection conditions by using the following definition.

Definition 11.5.4 (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 $\mathcal{C}_1$.*
- *Otherwise, $\mathcal{C}_1$ is an inapplicable condition and so r'' is not defined.*

To perform selections with non-atomic conditions on annotated relations, use the above definition except replace all instances of r and r'' with $ANN(r)$ and $ANN(r'')$ respectively. $\square$

For example if $\mathcal{C}_1 = (2/8/1997 \sim 7/8/1997)$, $\mathcal{C}_2 = (L \neq 0.10)$, and $\mathcal{C} = (\mathcal{C}_1 \wedge \mathcal{C}_2)$, then $\sigma_{\mathcal{C}}(r_1)$ will be

Data	H	C	D	L	U	δ
D1		$(2/8/1997 \sim 3/8/1997)$	$(1/8/1997 \sim 3/8/1997)$	0.64	0.88	g

and $\sigma_{\mathcal{C}}(r_2)$ will be

Data	H	C	D	L	U	δ
D1		$(3/8/1997)$	$(2/8/1997 \sim 3/8/1997)$	0.20	0.50	g

but if $\mathcal{C} = (\mathcal{C}_1 \vee \mathcal{C}_2)$, $\Xi^{hy}(\sigma_{\mathcal{C}}(r_1))$ will be

Data	H	C	D	L	U	δ
D1		$(1/8/1997 \sim 3/8/1997)$	$(1/8/1997 \sim 3/8/1997)$	0.64	0.88	g
		$(5/8/1997 \sim 7/8/1997)$	$(5/8/1997 \sim 8/8/1997)$	0.40	0.80	u

and $\Xi^{hy}(\sigma_{\mathcal{C}}(r_2))$ will be

Data	H	C	D	L	U	δ
D1		$(2/8/1997 \sim 3/8/1997)$	$(2/8/1997 \sim 3/8/1997)$	0.20	0.50	g
		$(6/8/1997 \sim 7/8/1997)$	$(6/8/1997 \sim 9/8/1997)$	0.40	0.80	u

The following theorem states that our definition of selection preserves commutativity.

Theorem 36 $\sigma_{\mathcal{C}_1}(\sigma_{\mathcal{C}_2}(r)) = \sigma_{\mathcal{C}_2}(\sigma_{\mathcal{C}_1}(r))$.

Proof.

Let $\mathcal{C}_1$ and $\mathcal{C}_2$ be two atomic selection constraints. We will consider a number of cases, each for each pair of constraint types.

- If both $\mathcal{C}_1$ and $\mathcal{C}_2$ are *data* constraints, then the statement of the theorem follows from the similar result in relational algebra.

- If both $\mathcal{C}_1$ and $\mathcal{C}_2$ are *temporal* constraints, the statement of the theorem will be true because of commutativity of the conjunction of boolean temporal constraints.

- If both $\mathcal{C}_1$ and $\mathcal{C}_2$ are *probabilistic* constraints, the statement of the theorem will be true because of commutativity of the conjunction of boolean probabilistic constraints.

- Let $\mathcal{C}_1$ be a *data* constraint and $\mathcal{C}_2$ be a *temporal* constraint.

  First we prove $\sigma_{\mathcal{C}_1}(\sigma_{\mathcal{C}_2}(r)) \subseteq \sigma_{\mathcal{C}_2}(\sigma_{\mathcal{C}_1}(r))$. Let $tp \in \sigma_{\mathcal{C}_1}(\sigma_{\mathcal{C}_2}(r))$. Then, (a) tp satisfies the data constraint $\mathcal{C}_1$ and (b) $tp \in \sigma_{\mathcal{C}_2}(r)$. By definition of selection on atomic temporal constraint, there exists a tp-tuple $tp' \in r$ such that
  $tp = (d, \gamma), \gamma = \gamma_1, \ldots, \gamma_k, \gamma_i = \langle C_i, D_i, L_i, U_i, \delta_i \rangle, 1 \le i \le k$;
  $tp' = (d, \gamma'), \gamma' = \gamma'_1, \ldots, \gamma'_m, \gamma'_i = \langle C'_i, D_i, L_i, U_i, \delta_i \rangle, 1 \le i \le m$ and

  1. $m \ge k$;
  2. and there exists a mapping $f : \{1, \ldots, k\} \to \{1, \ldots, m\}$ such that $f(i) = f(j)$ iff $i = j$ and $C_i = C'_{f(i)} \wedge \mathcal{C}_2$.

3. If for some $1 \leq l \leq m$, $C'_l \wedge \mathcal{C}_2$ is consistent, then there exists such number $1 \leq j \leq k$, that $f(j) = l$.

Since tp and tp' are data-identical, $tp' \in \sigma_{\mathcal{C}_1}(r)$, since tp' must also satisfy $\mathcal{C}_1$. But then, by the definition of selection on temporal constraint $\sigma_{\mathcal{C}_2}(\sigma_{\mathcal{C}_1}(r))$ will contain the tuple tp'' defined as follows:

$tp'' = (d, \gamma'')$, $\gamma'' = \gamma''_1, \ldots \gamma''_n$, $\gamma''_i = \langle C''_i, D_i, L_i, U_i, \delta_i \rangle$, $1 \leq i \leq n$ and for each $1 \leq j \leq m$ such that $C'_j \wedge \mathcal{C}_2$ is consistent, there exists a unique $1 \leq i \leq n$ such that $C''_i = C'_j \wedge \mathcal{C}_2$.

But the latter description is equivalent to the description of tp, i.e. $tp'' = tp$, i.e. $tp \in \sigma_{\mathcal{C}_2}(\sigma_{\mathcal{C}_1}(r))$.

Now we prove that $\sigma_{\mathcal{C}_1}(\sigma_{\mathcal{C}_2}(r)) \supseteq \sigma_{\mathcal{C}_2}(\sigma_{\mathcal{C}_1}(r))$. Let $tp \in \sigma_{\mathcal{C}_2}(\sigma_{\mathcal{C}_1}(r))$. This means that $(\sigma_{\mathcal{C}_1}(r))$ contains a tuple tp' such that $tp = (d, \gamma), \gamma = \gamma_1, \ldots, \gamma_k$, $\gamma_i = \langle C_i, D_i, L_i, U_i, \delta_i \rangle$, $1 \leq i \leq k$;

$tp' = (d, \gamma'), \gamma' = \gamma'_1, \ldots, \gamma'_m$, $\gamma'_i = \langle C'_i, D_i, L_i, U_i, \delta_i \rangle$, $1 \leq i \leq m$ and

1. $m \geq k$;
2. and there exists a mapping $f : \{1, \ldots, k\} \rightarrow \{1, \ldots, m\}$ such that $f(i) = f(j)$ iff $i = j$ and $C_i = C'_{f(i)} \wedge \mathcal{C}_2$.
3. If for some $1 \leq l \leq m$, $C'_l \wedge \mathcal{C}_2$ is consistent, then there exists such number $1 \leq j \leq k$, that $f(j) = l$.

Since $tp' \in \sigma_{\mathcal{C}_1}(r)$ and $\mathcal{C}_1(r)$, $tp' \in r$. Then $\sigma_{\mathcal{C}_2}(r)$ will contain a tuple tp'' such that $tp'' = (d, \gamma'')$, $\gamma'' = \gamma''_1, \ldots \gamma''_n$, $\gamma''_i = \langle C''_i, D_i, L_i, U_i, \delta_i \rangle$, $1 \leq i \leq n$ and for each $1 \leq j \leq m$ such that $C'_j \wedge \mathcal{C}_2$ is consistent, there exists a unique $1 \leq i \leq n$ such that $C''_i = C'_j \wedge \mathcal{C}_2$. Clearly, $tp = tp''$. Since tp'' and tp' are data-identical, $tp'' \in \sigma_{\mathcal{C}_1}(\sigma_{\mathcal{C}_2}(r))$, i.e., $tp \in \sigma_{\mathcal{C}_1}(\sigma_{\mathcal{C}_2}(r))$.

The proof for the case when $\mathcal{C}_1$ is a temporal constraint and $\sigma_{\mathcal{C}_2}$ is a data constraint is symmetric.

- $\mathcal{C}_1$ is a *data* constraint and $\sigma_{\mathcal{C}_2}$ is a *probabilistic* constraint.

  First we prove $\sigma_{\mathcal{C}_1}(\sigma_{\mathcal{C}_2}(r)) \subseteq \sigma_{\mathcal{C}_2}(\sigma_{\mathcal{C}_1}(r))$. Let $tp \in \sigma_{\mathcal{C}_1}(\sigma_{\mathcal{C}_2}(r))$. Then, (a) tp satisfies the data constraint $\mathcal{C}_1$ and (b) $tp \in \sigma_{\mathcal{C}_2}(r)$. By definition of selection on atomic probabilistic constraint, there exists a tp-tuple $tp' \in r$ such that

  $tp = (d, \gamma), \gamma = \gamma_1, \ldots, \gamma_k, \gamma_i = \langle C_i, D_i, L_i, U_i, \delta_i \rangle, 1 \leq i \leq k$;

  $tp' = (d, \gamma'), \gamma' = \gamma'_1, \ldots, \gamma'_m, \gamma'_i = \langle C'_i, D_i, L_i, U_i, \delta_i \rangle, 1 \leq i \leq m$ and

  1. $m \geq k$;
  2. and there exists a mapping $f : \{1, \ldots, k\} \to \{1, \ldots, m\}$ such that $f(i) = f(j)$ iff $i = j$ and $C_i = TP - filter(\gamma_{f(i)}, \mathcal{C}_2)$.
  3. If for some $1 \leq l \leq m$, $sol(TP - filter(\gamma_{f(i)}, \mathcal{C}_2)) \neq \emptyset$ then there exists such number $1 \leq j \leq k$, that $f(j) = l$.

  Then by definition of selection on atomic *data* constraint, $\sigma_{\mathcal{C}_1}(r)$ will contain tp' as tp and tp' are data-identical and tp satisfies $\mathcal{C}_1$. Therefore, $\sigma_{\mathcal{C}_2}(\sigma_{\mathcal{C}_1}(r))$ will contain a tp-tuple $tp'' = (d, \gamma'')$ constructed as follows: for each $\gamma'_i \in \gamma'$ such that sol$(TP-filter(\gamma'_i, \mathcal{C}_2)) \neq \emptyset$, there will be a case $\langle TP - filter(\gamma'_i, \mathcal{C}_2), D_i, L_i, U_i, \delta_i \rangle$ in γ'' and there will be no other cases in γ''. But it is clear that in this case $tp'' = tp$ and therefore $tp \in \sigma_{\mathcal{C}_2}(\sigma_{\mathcal{C}_1}(r))$.

  To prove that $\sigma_{\mathcal{C}_1}(\sigma_{\mathcal{C}_2}(r)) \supseteq \sigma_{\mathcal{C}_2}(\sigma_{\mathcal{C}_1}(r))$ now let $tp \in \sigma_{\mathcal{C}_2}(\sigma_{\mathcal{C}_1}(r))$.

  In this case, $\sigma_{\mathcal{C}_1}(r)$ will contain a tp-tuple tp' such that

  $tp = (d, \gamma), \gamma = \gamma_1, \ldots, \gamma_k, \gamma_i = \langle C_i, D_i, L_i, U_i, \delta_i \rangle, 1 \leq i \leq k$;

  $tp' = (d, \gamma'), \gamma' = \gamma'_1, \ldots, \gamma'_m, \gamma'_i = \langle C'_i, D_i, L_i, U_i, \delta_i \rangle, 1 \leq i \leq m$ and

1. $m \geq k$;

2. and there exists a mapping $f : \{1, \ldots, k\} \rightarrow \{1, \ldots, m\}$ such that $f(i) = f(j)$ iff $i = j$ and $C_i = TP - filter(\gamma_{f(i)}, \mathcal{C}_2)$.

3. If for some $1 \leq l \leq m$, $sol(TP - filter(\gamma'_{f(i)}, \mathcal{C}_2)) \neq \emptyset$ then there exists such number $1 \leq j \leq k$, that $f(j) = l$.

Then by definition of selection on atomic *data* constraint, tp' must satisfy $\mathcal{C}_1(r)$ and therefore $tp' \in r$. But then $\mathcal{C}_2(r)$ will contain tp (see the construction of tp'' above to see why this is true). And since tp and tp' are data-identical, tp will satisfy $\mathcal{C}_1(r)$ and therefore $tp \in \sigma_{\mathcal{C}_1}(\sigma_{\mathcal{C}_2}(r))$.

The proof for the case when $\mathcal{C}_1$ is a probabilistic constraint and $\mathcal{C}_2$ is a data constraint is symmetric.

- $\mathcal{C}_1$ is a *temporal* constraint and $\mathcal{C}_2$ is a *probabilistic* constraint.

First we prove $\sigma_{\mathcal{C}_1}(\sigma_{\mathcal{C}_2}(r)) \subseteq \sigma_{\mathcal{C}_2}(\sigma_{\mathcal{C}_1}(r))$. Let $tp \in \sigma_{\mathcal{C}_1}(\sigma_{\mathcal{C}_2}(r))$. Then, by definition of a selection on an atomic temporal condition, $\sigma_{\mathcal{C}_2}(r)$ will contain a tuple tp' such that

$tp = (d, \gamma), \gamma = \gamma_1, \ldots, \gamma_k, \gamma_i = \langle C_i, D_i, L_i, U_i, \delta_i \rangle, 1 \leq i \leq k$;
$tp' = (d, \gamma'), \gamma' = \gamma'_1, \ldots, \gamma'_m, \gamma'_i = \langle C'_i, D_i, L_i, U_i, \delta_i \rangle, 1 \leq i \leq m$ and

1. $m \geq k$;

2. and there exists a mapping $f : \{1, \ldots, k\} \rightarrow \{1, \ldots, m\}$ such that $f(i) = f(j)$ **iff** $i = j$ and $C_i = C'_{f(i)} \wedge \mathcal{C}_2$.

3. If for some $1 \leq l \leq m$, $C'_l \wedge \mathcal{C}_2$ is consistent, then there exists such number $1 \leq j \leq k$, that $f(j) = l$.

Since $tp' \in \sigma_{\mathcal{C}_2}(r)$, by definition of selection on atomic *probabilistic* condition there exists a tuple $tp'' \in r$ such that

$tp'' = (d, \gamma''), \gamma'' = \gamma''_1, \ldots, \gamma''_n, \gamma''_i = \langle C''_i, D_i, L_i, U_i, \delta_i \rangle, 1 \leq i \leq n$ and

1. $n \geq m$;
2. and there exists a mapping $g : \{1, \ldots, m\} \to \{1, \ldots, n\}$ such that $g(i) = g(j)$ **iff** $i = j$ and $C'_i = TP - filter(\gamma''_{g(i)}, \mathcal{C}_2)$.
3. If for some $1 \leq l \leq n$, $sol(TP - filter(\gamma_{g(i)}, \mathcal{C}_2)) \neq \emptyset$ then there exists such number $1 \leq j \leq k$, that $g(j) = l$.

From the above we obtain that $C_i = TP - filter(\gamma(g(f(i)), \mathcal{C}_2) \wedge \mathcal{C}_1$.

Since $tp'' \in r$, we know that $\sigma_{\mathcal{C}_1}(r)$ will contain a tp-tuple $tp''' = (d, \gamma''')$ such that $\gamma''' = \gamma'''_1, \ldots \gamma'''_r, \gamma'''_i = \langle C'''_i, D_i, L_i, U_i, \delta_i \rangle$, $1 \leq i \leq r$ and for each $1 \leq j \leq n$ such that $C''_j \wedge \mathcal{C}_1$ is consistent, there exists a unique $1 \leq i \leq r$ such that $C'''_i = C''_j \wedge \mathcal{C}_2$.

Finally, $\sigma_{\mathcal{C}_2}(\sigma_{\mathcal{C}_1}(r))$ will contain a tp-tuple $tp^* = (d, \gamma^*)$ such that for each $\gamma'''_i \in \gamma'''$ such that sol$(TP - filter(\gamma'''_i, \mathcal{C}_2)) \neq \emptyset$, there will be a case $\langle TP - filter(\gamma'''_i, \mathcal{C}_2), D_i, L_i, U_i, \delta_i \rangle$ in γ^* and there will be no other cases in γ^*.

Now we will show that tp indeed is equal to tp^*. As we noticed, every case in tp had its C constraint have a form $C_i = TP - filter(\gamma''_g(f(i)), \mathcal{C}_2) \wedge \mathcal{C}_1$.

Now, it is easy to notice that since every constraint in tp''' had a form $C'''_j = C''_l \wedge \mathcal{C}_1$, every constraint is tp^* will be of the form $C^*_i = TP - filter(\gamma'''_j, \mathcal{C}_2) = TP - filter(\langle C''_l \wedge \mathcal{C}_1, D''_l, L''_l, U''_l, \delta''_l \rangle, \mathcal{C}_2)$.

Clearly, if for some $1 \leq j \leq n$, there was such $1 \leq i \leq k$ that $C_i = TP - filter(\gamma''_j, \mathcal{C}_2) \wedge \mathcal{C}_1$ then $TP - filter(\langle C''_j \wedge \mathcal{C}_1, D''_j, L''_j, U''_j, \delta''_j \rangle, \mathcal{C}_2)$ will be the value of some C^*_l from γ^*.

We now show that

$$TP - filter(\gamma_j'', \mathcal{C}_2) \wedge \mathcal{C}_1 = TP - filter(\langle C_j'' \wedge \mathcal{C}_1, D_j'', L_j'', U_j'', \delta_j'' \rangle, \mathcal{C}_2)).$$

Let $t \in sol(TP - filter(\gamma_j'', \mathcal{C}_2) \wedge \mathcal{C}_1)$. In this case, $t \in sol(C_j'')$ and $\delta_j''(D_j'', t)$ satisfies $\mathcal{C}_2$. Also, t satisfies $\mathcal{C}_1$. But then, t satisfies $C_j'' \wedge \mathcal{C}_1$. Since this does not affect the probability estimate for t, we obtain that $t \in sol(TP - filter(\langle C_j'' \wedge \mathcal{C}_1, D_j'', L_j'', U_j'', \delta_j'' \rangle, \mathcal{C}_2))$.

Conversely, if $t \in sol(TP - filter(\langle C_j'' \wedge \mathcal{C}_1, D_j'', L_j'', U_j'', \delta_j'' \rangle, \mathcal{C}_2))$ then (a) $\delta_j''(D_j'', t)$ satisfies $\mathcal{C}_2$ and (b) t satisfies $C_j'' \wedge \mathcal{C}_1$. Therefore t satisfies C_j'' and t satisfies $\mathcal{C}_1$. Then clearly, $t \in sol(TP - filter(\langle C_j'', D_j'', L_j'', U_j'', \delta_j'' \rangle \mathcal{C}_2))$. Therefore, $t \in sol(TP - filter(\langle C_j'', D_j'', L_j'', U_j'', \delta_j'' \rangle \mathcal{C}_2)) \cap sol(\mathcal{C}_1)$, i.e.
$t \in sol(TP - filter(\gamma_j'', \mathcal{C}_2) \wedge \mathcal{C}_1)$.

This proves the desired inclusion.

The proof that $\sigma_{\mathcal{C}_1}(\sigma_{\mathcal{C}_2}(r)) \supseteq \sigma_{\mathcal{C}_2}(\sigma_{\mathcal{C}_1}(r))$ is symmetric to the proof above. Similarly, the proof of the statement of the theorem for the case when $\mathcal{C}_1$ is a probabilistic condition and $\mathcal{C}_2$ is a temporal condition is symmetric to the proof above.

The list enumerates all possible pairs of types of constraints, therefore the theorem is proven. □

The following table shows how one may generate queries on TP-relation r which correspond to seven of J. F. Allen's thirteen possible temporal relationships [3]. The six remaining possibilities correspond to the inverses of these original seven (the inverse of "equal" is identical to "equal" so it is not counted). Here, we ensure that r uses two TP-tuples for each continuous duration event e. Specifically, suppose e can be described by data tuple d with relational schema A. Then r is a TP-relation over relational schema

(A,Kind) where $dom(Kind) = \{S, E\}$. r contains a TP-tuple (d,"S",γ_S) for instantaneous event $st(e)$, and a TP-tuple (d,"E",γ_E) for instantaneous event $end(e)$.

Description	Specification	Query	Conditions
e before e_q	$end(e) \leq st(e_q)$	$\sigma_{\mathcal{C}}(r)$	$\mathcal{C} = ((t_S \sim t_1) \wedge (\text{Kind} = E))$
e equal e_q	$st(e) = st(e_q) \wedge$ $end(e) = end(e_q)$	$\sigma_{\mathcal{C}_1}(r) \cap_{eq} \sigma_{\mathcal{C}_2}(r)$	$\mathcal{C}_1 = ((t_1 \sim t_1) \wedge (\text{Kind} = S))$ $\mathcal{C}_2 = ((t_2 \sim t_2) \wedge (\text{Kind} = E))$
e meets e_q	$end(e) = st(e_q)$	$\sigma_{\mathcal{C}}(r)$	$\mathcal{C} = ((t_1 \sim t_1) \wedge (\text{Kind} = E))$
e overlaps e_q	$st(e) \leq st(e_q) \wedge$ $st(e_q) \leq end(e)$	$\sigma_{\mathcal{C}_1}(r) \cap_{eq} \sigma_{\mathcal{C}_2}(r)$	$\mathcal{C}_1 = ((t_S \sim t_1) \wedge (\text{Kind} = S))$ $\mathcal{C}_2 = ((t_1 \sim t_E) \wedge (\text{Kind} = E))$
e during e_q	$st(e_q) \leq st(e) \wedge$ $end(e) \leq end(e_q)$	$\sigma_{\mathcal{C}_1}(r) \cap_{eq} \sigma_{\mathcal{C}_2}(r)$	$\mathcal{C}_1 = ((t_1 \sim t_E) \wedge (\text{Kind} = S))$ $\mathcal{C}_2 = ((t_S \sim t_2) \wedge (\text{Kind} = E))$
e starts e_q	$st(e) = st(e_q) \wedge$ $end(e) \leq end(e_q)$	$\sigma_{\mathcal{C}_1}(r) \cap_{eq} \sigma_{\mathcal{C}_2}(r)$	$\mathcal{C}_1 = ((t_1 \sim t_1) \wedge (\text{Kind} = S))$ $\mathcal{C}_2 = ((t_S \sim t_2) \wedge (\text{Kind} = E))$
e finishes e_q	$st(e_q) \leq st(e)$ $end(e) = end(e_q)$	$\sigma_{\mathcal{C}_1}(r) \cap_{eq} \sigma_{\mathcal{C}_2}(r)$	$\mathcal{C}_1 = ((t_1 \sim t_E) \wedge (\text{Kind} = S))$ $\mathcal{C}_2 = ((t_2 \sim t_2) \wedge (\text{Kind} = E))$

When executing these temporal queries, use the following rule: If $st(e)$ or $end(e)$ satisfies the selection condition, then the TP-tuples for both $st(e)$ and $end(e)$ should be returned. Thus, our queries will return TP-tuples for every event e which satisfies some relationship w.r.t. query event e_q where $st(e_q) = t_1$ and $end(e_q) = t_2$. Recall that t_S (t_E) denotes the earliest (latest) time point of a calendar.

The following shows that our definition of selection correctly implements the TATA semantics.

Theorem 37 (Correctness of selection) $\text{ANN}(\sigma_{\mathcal{C}}(r)) = \sigma_{\mathcal{C}}(\text{ANN}(r))$.

Proof.

We will prove this theorem for the case when $\mathcal{C}$ is an atomic predicate (constraint). Then, by theorem 36 this theorem will be true for arbitrary constraints as well.

So, assume $\mathcal{C}$ is atomic predicate. We have to consider three cases:

1. $\mathcal{C}$ is a predicate on the data part of the relational schema of r.

   In this case we notice that $\sigma_{\mathcal{C}}(r) \subseteq r$, i.e. only tuples from r can be found in $\sigma_{\mathcal{C}}(r)$.

   Let $at = (d, t, L, U) \in ANN(\sigma_{\mathcal{C}}(r))$. $\mathcal{C}$. By definition of annotation operation, there must be a tuple $tp = (d, \gamma) \in \sigma_{\mathcal{C}}(r)$, such that $at \in ANN(tp)$. Since $\mathcal{C}$

   is a predicate on the data part, it must be the case that d satisfies $\mathcal{C}$.

   But since $\sigma_{\mathcal{C}}(r) \subseteq r$, we know that $tp \in r$, therefore, since $at \in ANN(tp)$, $at \in ANN(r)$. Finally, as d satisfies $\mathcal{C}$ we get $at \in \sigma_{\mathcal{C}}(ANN(r))$.

   To prove the inclusion the other way around, let us consider the tuple
   $at = (d, t, L, U \in \sigma_{\mathcal{C}}(ANN(r))$. Since $\mathcal{C}$ is a predicate on the data part d satisfies $\mathcal{C}$.

   By definition of selection on annotated tuples, $at \in ANN(r)$. But then, there must be a tp-tuple $tp = (d, \gamma) \in r$ such that $at \in ANN(tp)$. since d satisfies $\mathcal{C}$, $tp \in \sigma_{\mathcal{C}}(r)$, and therefore, $at \in ANN(\sigma_{\mathcal{C}}(r))$.

2. $\mathcal{C}$ is a temporal predicate.

   - $ANN(\sigma_{\mathcal{C}}(r)) \subseteq \sigma_{\mathcal{C}}(ANN(r))$

     Let $at = (d, t, L, U) \in ANN(\sigma_{\mathcal{C}}(r))$. We show that $at \in \sigma_{\mathcal{C}}(ANN(r))$.

     $at \in ANN(\sigma_{\mathcal{C}}(r))$ implies that there exists a tp-tuple $tp = (d, \gamma) \in \sigma_{\mathcal{C}}(r)$, such that

     $\gamma = \gamma_1, \ldots, \gamma_n$;

     $at \in ANN(tp)$ and

$(\exists i \in \{1, \ldots, n\})(\gamma_i = \langle C_i, D_i, L_i, U_i, \delta_i \rangle \wedge t \in sol(C_i) \wedge [L, U] = [\delta_i(D_i, t)L_i, \delta_i(D_i, t)U_i]$.

Since $tp \in \sigma_{\mathcal{C}}(r)$, by definition of selection on a TP-relation, there must exist a tp-tuple $tp' \in r$, such that:

$tp' = (d, \gamma'); \gamma' = \gamma'_1, \ldots, \gamma'_k$ and

$(\exists j \in \{1, \ldots, k\})(\gamma'_j = \langle C'_j, D'_j, L'_j, \delta'_j \rangle)$ such that

$\langle D'_j, L'_j, U'_j, \delta'_j \rangle = \langle D_i, L_i, U_i, \delta_i \rangle$ and

$C_i = C'_j \wedge \mathcal{C}$.

Since $t \in sol(C_i)$, $t \in sol(C'_j)$ and $t \in sol(\mathcal{C})$. The former means that $at \in ANN(tp')$, i.e., $at \in ANN(r)$. The latter means that by definition of selection on annotated relation, $at \in \sigma_{\mathcal{C}}(ANN(r))$.

- $ANN(\sigma_{\mathcal{C}}(r)) \supseteq \sigma_{\mathcal{C}}(ANN(r))$

  Let $at = (d, t, L, U) \in \sigma_{\mathcal{C}}(ANN(r))$. We show that $at \in ANN(\sigma_{\mathcal{C}}(r))$.

  Since $at \in \sigma_{\mathcal{C}}(ANN(r))$, by definition of selection on annotated relations

  (i) $at \in ANN(r)$

  (ii) $t \in sol(\mathcal{C})$

  Since $at \in ANN(r)$, there exists a tp-tuple $tp = (d, \gamma) \in r$, such that $\gamma = \gamma_1, \ldots, \gamma_n$;

  $at \in ANN(tp)$ and

  $(\exists i \in \{1, \ldots, n\})(\gamma_i = \langle C_i, D_i, L_i, U_i, \delta_i \rangle \wedge t \in sol(C_i) \wedge [L, U] = [\delta_i(D_i, t)L_i, \delta_i(D_i, t)U_i]$.

  We know that $t \in sol(C_i)$ and $t \in sol(\mathcal{C})$, therefore $t \in sol(C_i \wedge \mathcal{C})$. This means that there exists a tp-tuple $tp' \in \sigma_{\mathcal{C}}(r)$ such that,

  $tp' = (d, \gamma'); \gamma' = \gamma'_1, \ldots, \gamma'_k$ and

  $(\exists j \in \{1, \ldots, k\})(\gamma'_j = \langle C'_j, D'_j, L'_j, \delta'_j \rangle)$ such that

$\langle D'_j, L'_j, U'_j, \delta'_j \rangle = \langle D_i, L_i, U_i, \delta_i \rangle$ and

$C'_j = C_i \wedge \mathcal{C}$.

$t \in sol(C'_j)$ and therefore, $at \in ANN(tp')$, i.e., $at \in ANN(\sigma_{\mathcal{C}}(r))$.

3. $\mathcal{C}$ is a probabilistic predicate.

   - $ANN(\sigma_{\mathcal{C}}(r)) \subseteq \sigma_{\mathcal{C}}(ANN(r))$

     Let $at = (d, t, L, U) \in ANN(\sigma_{\mathcal{C}}(r))$. We show that $at \in \sigma_{\mathcal{C}}(ANN(r))$.

     $at \in ANN(\sigma_{\mathcal{C}}(r))$ implies that there exists a tp-tuple $tp = (d, \gamma) \in \sigma_{\mathcal{C}}(r)$, such that

     $\gamma = \gamma_1, \ldots, \gamma_n$;

     $at \in ANN(tp)$ and

     $(\exists i \in \{1, \ldots, n\})(\gamma_i = \langle C_i, D_i, L_i, U_i, \delta_i \rangle \wedge t \in sol(C_i) \wedge [L, U] = [\delta_i(D_i, t)L_i, \delta_i(D_i, t)U_i])$.

     Since $tp \in \sigma_{\mathcal{C}}(r)$, by definition of selection on a TP-relation, there must exist a tp-tuple $tp' \in r$, such that:

     $tp' = (d, \gamma'); \gamma' = \gamma'_1, \ldots, \gamma'_k$ and

     $(\exists j \in \{1, \ldots, k\})(\gamma'_j = \langle C'_j, D'_j, L'_j, \delta'_j \rangle)$ such that

     $\langle D'_j, L'_j, U'_j, \delta'_j \rangle = \langle D_i, L_i, U_i, \delta_i \rangle$ and

     $C_i =$ TP-filter$(\gamma'_j, \mathcal{C})$

     Since $t \in sol(C_i)$, i.e, $t \in sol($TP-filter$(\gamma'_j, \mathcal{C}))$, we have, $[L, U] \in sol(\mathcal{C})$ and also, $t \in sol(C_j)$ The latter means that $at \in ANN(tp')$, i.e., $at \in ANN(r)$. The former means that by definition of selection on annotated relation, $at \in \sigma_{\mathcal{C}}(ANN(r))$.

   - $ANN(\sigma_{\mathcal{C}}(r)) \supseteq \sigma_{\mathcal{C}}(ANN(r))$

     Let $at = (d, t, L, U) \in \sigma_{\mathcal{C}}(ANN(r))$. We show that $at \in ANN(\sigma_{\mathcal{C}}(r))$.

Since $at \in \sigma_{\mathcal{C}}(ANN(r))$, by definition of selection on annotated relations

(i) $at \in ANN(r)$

(ii) $[L, U] \in sol(\mathcal{C})$.

Since $at \in ANN(r)$, there exists a tp-tuple $tp = (d, \gamma) \in r$, such that $\gamma = \gamma_1, \ldots, \gamma_n$;

$at \in ANN(tp)$ and

$(\exists i \in \{1, \ldots, n\})(\gamma_i = \langle C_i, D_i, L_i, U_i, \delta_i \rangle \wedge t \in sol(C_i) \wedge [L, U] = [\delta_i(D_i, t)L_i, \delta_i(D_i, t)U_i])$.

Let us consider tuple $tp' = \sigma_{\mathcal{C}}(tp)$ i.e.,

$tp' = (d, \gamma'); \gamma' = \gamma'_1, \ldots, \gamma'_k$ and

$(\exists j \in \{1, \ldots, k\})(\gamma'_j = \langle C'_j, D'_j, L'_j, \delta'_j \rangle)$ such that

$\langle D'_j, L'_j, U'_j, \delta'_j \rangle = \langle D_i, L_i, U_i, \delta_i \rangle$ and

$C'_j =$ TP-filter$(\gamma_i, \mathcal{C})$

Since $[L, U] \in sol(\mathcal{C})$, and $t \in sol(C_i)$, we know that

$t \in sol($TP-filter$(\gamma_i, \mathcal{C}))$ and therefore, $at \in ANN(tp')$, i.e.,

$at \in ANN(\sigma_{\mathcal{C}}(r))$. □

11.6 Difference of two TP-relations

In this section, we show how the difference of two TP-relations can be correctly implemented.

Definition 11.6.1 (difference of two TP-relations) *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''. □

For example $r_1 - r_2$ will be

Data	H	C	D	L	U	δ
D1		$(1/8/1997)$	$(1/8/1997 \sim 3/8/1997)$	0.64	0.88	g
		$(5/8/1997)$	$(5/8/1997 \sim 8/8/1997)$	0.40	0.80	u

and $r_2 - r_1$ will be

Data	H	C	D	L	U	δ
D1		$(9/8/1997)$	$(6/8/1997 \sim 9/8/1997)$	0.40	0.80	u

The following theorem shows that our definition of difference correctly implements the TATA semantics.

Theorem 38 (Correctness of difference) $\mathrm{ANN}(r - r') = \mathrm{ANN}(r) - \mathrm{ANN}(r')$.

Proof.

We break the proof into two parts:

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

   Let $at = (d, t, L, U) \in ANN(r - r')$. We will show that $at \in \mathrm{ANN}(r) - \mathrm{ANN}(r')$. As $at \in ANN(r - r')$, by definition of ANN, there exists such a tuple $tp'' = (d, \gamma'') \in r - r'$ such that $at \in ANN(tp)$. Two cases have to be considered here:

(a) tp'' is in $(r - r')$ because $tp'' \in r$ and there is no tp-tuple in r' which is data identical to it. In this case, there is no annotated tuple in $ANN(r')$ which is data identical to $ANN(tp^p p)$ and hence, $at \in ANN(r) - ANN(r')$.

(b) Otherwise, tp'' is in $(r - r')$ because there is a tp-tuples $tp = (d, \gamma) \in r$ and a $tp' = (d, \gamma')$ in r' and tp'' is constructed from these two tp-tuples using the construction shown in Definition 11.6.1. Let (tp, tp') be any such pair of tp-tuples.

As $at = (d, t, L, U) \in ANN(r - r')$, it follows that there is a unique integer i such that the constraint of the form $C_i \wedge \neg C'$ in Step (2) of Definition 11.6.1 is satisfied. This means that the time point t is a solution of one of the C-constraints of tp and none of the C-constraints of tp'. This holds for all tp-tuples in r' that have an annotated tuple of the form $(d, t, -, -)$ in their annotated expansion. Hence, independently of how we choose a tuple tp° from r', if tp and tp° are data-identical, then no tuple of the form $(d, t, -, -)$ can be in $ANN(tp) - ANN(tp^\circ)$. It follows that there can be no annotated tuple of the form $(d, t, -, -)$ can be in $ANN(r')$. If the C-constraint of tp alluded to above is $\langle C_i, D_i, L_i, U_i, \delta_i \rangle$, then we know that $(d, t, L, U) \in ANN(r)$ where $L = \delta_i(D_i, t) \cdot L_i$ and $U = \delta_i(D_i, t) \cdot U_i$. Hence, $at \in \mathrm{ANN}(r) - \mathrm{ANN}(r')$.

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

Let $at = (d, t, L, U) \in \mathrm{ANN}(r) - \mathrm{ANN}(r')$. By definition, $at \in ANN(r)$ and there is no $at' = (d', t', L', U') \in ANN(r')$ such that $d = d'$ and $t = t'$. It follows by definition of ANN that there exists $tp \in r$ such that $at \in ANN(tp)$ and there is no $tp' \in r'$ such that $ANN(tp')$ is of the form $(d, t, -, -)$. It follows immediately from the construction of $r - r'$ (Definition 11.6.1) that $at \in ANN(r - r')$. $\square$

11.7 Cartesian product of two TP-relations

In this section, we show how we can correctly implement the cartesian product of two TP-relations. Recall, as in the annotated case, that when taking the cartesian product of two relations, we must know the relationship, if any, between the events denoted by the tuples in the two relations because the probability of a concatenated tuple being present in the result of the Cartesian Product is the probability of a conjunctive event. Thus, conjunction strategies parameterize the Cartesian Product operation.

Definition 11.7.1 (cartesian product of two TP-relations) *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 $t \in \mathrm{sol}(C_i)$ for some $\gamma_i \in \gamma$ and $t \in \mathrm{sol}(C'_j)$ 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 $h'' = (d.\mathrm{H} \parallel d'.\mathrm{H})$.* □

For example, $\Xi^{hy}(r_1 \times_{ig} r_2)$ will be

r_1.Data	r_2.Data	H	C	D	L	U	δ
D1	D1		$(\#)$	$(2/8/1997 \sim 2/8/1997)$	0.00	0.22	u
			$(\#)$	$(3/8/1997 \sim 3/8/1997)$	0.00	0.11	u
			$(\#)$	$(6/8/1997 \sim 8/8/1997)$	0.00	0.60	u

but $\Xi^{hy}(r_1 \times_{pc} r_2)$ will be

r_1.Data	r_2.Data	H	C	D	L	U	δ
D1	D1		$(\#)$	$(2/8/1997 \sim 2/8/1997)$	0.10	0.22	u
			$(\#)$	$(3/8/1997 \sim 3/8/1997)$	0.05	0.11	u
			$(\#)$	$(6/8/1997 \sim 8/8/1997)$	0.30	0.60	u

The use of a TP-compression operation when executing a Cartesian product operation is important because sometimes, Cartesian product can produce a large number of TP-cases when an existing tp-case gets broken into "pieces." TP-compressions prevent this from happening. The following result shows that our definition of cartesian product correctly implements the TATA semantics.

Theorem 39 (Correctness of cartesian product)

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

Proof.

The proof will consist of two parts

1. $ANN(r \times_\alpha r') \subseteq ANN(r) \times_\alpha ANN(r')$.

   Let $at'' = (d'', t'', L'', U'') \in ANN(r \times_\alpha r')$. We will show that $at'' \in ANN(r) \times_\alpha ANN(r')$. By the definition of an annotated relation, there exists a tp-tuple $tp'' \in r \times_\alpha r'$ such that $at'' \in ANN(tp'')$. This means that

$$tp'' = (d'', \gamma''), \gamma'' = \gamma''_1, \ldots, \gamma''_n, \gamma''_i = \langle C''_i, D''_i, L''_i, U''_i, \delta''_i \rangle, 1 \leq i \leq n$$

and there exists a number $1 \leq j \leq n$ such that $sol(C''_j) = \{t''\}$ and $[L'', U''] = [L''_j, U''_j]$.

Since $tp'' \in r \times_\alpha r'$, by the definition of cartesian product of two tp-relations, there exists such a tp-tuple $tp \in r$ and a tp-tuple $tp' \in r'$ that:

$$tp = (d, \gamma) \qquad\qquad tp' = (d', \gamma')$$

$$d'' = d, d'$$

$$\gamma = \gamma_1, \ldots \gamma_k \qquad\qquad \gamma' = \gamma'_1, \ldots \gamma'_m$$

$$\gamma_i = \langle C_i, D_i, L_i, U_i, \delta_i \rangle, \qquad\qquad \gamma'_i = \langle C'_i, D'_i, L'_i, U'_i, \delta'_i \rangle,$$

$$1 \leq i \leq k \qquad\qquad 1 \leq i \leq m$$

$$(\exists h \in \{1, \ldots k\})(t'' \in sol(C_h)) \qquad\qquad (\exists l \in \{1, \ldots m\})(t'' \in sol(C'_l))$$

and $[L''_j, U''_j] = [L'', U''] = [\delta_h(D_h, t'') \cdot L_h, \delta_h(D_h, t'') \cdot U_h] \otimes_\alpha [\delta'_l(D'_l, t'') \cdot L'_l, \delta'_l(D'_l, t'') \cdot U'_l]$.

But in this case, by the definition of annotated relation, there will be an annotated tuple $at = (d, t'', \delta_h(D_h, t'') \cdot L_h, \delta_h(D_h, t'') \cdot U_h) \in ANN(tp) \subseteq ANN(r)$ and an annotated tuple $at' = (d', t'', \delta'_l(D'_l, t'') \cdot L'_l, \delta'_l(D'_l, t'') \cdot U'_l) \in ANN(tp') \subseteq ANN(r')$. Then by the definition of a cartesian product of annotated relations and since $[L'', U''] = [\delta_h(D_h, t'') \cdot L_h, \delta_h(D_h, t'') \cdot U_h] \otimes_\alpha [\delta'_l(D'_l, t'') \cdot L'_l, \delta'_l(D'_l, t'') \cdot U'_l]$, $at'' = (d, d', t'', L'', U'') \in ANN(r) \times_\alpha ANN(r')$.

2. $ANN(r) \times_\alpha ANN(r') \subseteq ANN(r \times_\alpha r')$.

   Let $at'' = (d'', t'', L'', U'') \in ANN(r) \times_\alpha ANN(r')$. We will show that $at'' \in ANN(r \times_\alpha r')$. By the definition of cartesian product of two annotated relations there exists an annotated tuple $at = (d, t'', L, U) \in ANN(r)$ and an annotated tuple $at' = (d', t'', L', U') \in ANN(r')$ such that $d'' = d, d'$ and $[L'', U''] = [L, U] \otimes_\alpha [L', U']$.

Since $at \in ANN(r)$, there exists a tp-tuple $tp \in r$ such that $at \in ANN(tp)$, i.e., $tp = (d, \gamma)$, $\gamma = \gamma_1, \ldots \gamma_k$, $\gamma_i = \langle C_i, D_i, L_i, U_i, \delta_i \rangle$, $1 \leq i \leq k$ and $(\exists h \in \{1, \ldots k\})$
$(t'' \in sol(C_h))$ such that $[L, U] = [\delta_h(D_h, t'') \cdot L_h, \delta_h(D_h, t'') \cdot U_h]$.

Similarly, there exists a tp-tuple $tp' \in r'$ such that $at' \in ANN(tp')$, i.e., $tp' = (d'\gamma')$, $\gamma' = \gamma'_1, \ldots \gamma'_m$, $\gamma'_i =$
$\langle C'_i, D'_i, L'_i, U'_i, \delta'_i \rangle$, $1 \leq i \leq m$ and $(\exists l \in \{1, \ldots m\})(t'' \in sol(C'_l))$ and $[L', U'] = [\delta'_l(D'_l, t'') \cdot L'_l, \delta'_l(D'_l, t'') \cdot U'_l]$.

Since $t'' \in C_h$ and $t'' \in C'_l$, by definition of cartesian product of two tp-relations, $r \times_\alpha r'$ will contain a tp-tuple

$$tp'' = (d, d', \gamma''), \gamma'' = \gamma''_1, \ldots, \gamma''_n, \gamma''_i = \langle C''_i, D''_i, L''_i, U''_i, \delta''_i \rangle, 1 \leq i \leq n$$

such that there exists a number $1 \leq j \leq n$ such that $sol(C''_j) = \{t''\}$ and $[L''_j, U''_j] = [L, U] \otimes_\alpha [L', U'] = [L'', U'']$. But then $ANN(r \times_\alpha r')$ will contain the tuple $at'' = ((d, d'), t'', L'', U'')$. $\square$

11.8 Projection on a TP-relation

In this section, we show how we can correctly implement projection on a TP-relation.

Definition 11.8.1 (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 $h'' = (d.\mathrm{H} \parallel$ "A_1:$d.\mathrm{A}_1$,..., A_n:$d.\mathrm{A}_n$").* $\square$

Recall that the $\pi_{\mathcal{F}}(\mathcal{P}(d))$ operator was defined in section 10.7.

For example if $\mathcal{F}$ = "Data1" and our primary key for r_3 was "Data1,Data2", $r'' = \pi_{\mathcal{F}}(r_3)$ will be

Data1	H	C	D	L	U	δ
D1	Data2:D2	$(\#)$	$(2/8/1997 \sim 2/8/1997)$	0.20	0.40	u
D1	Data2:D3	$(\#)$	$(2/8/1997 \sim 3/8/1997)$	0.60	0.80	g

The following theorem shows that our definition of projection correctly implements the TATA semantics.

Theorem 40 (Correctness of projection) $\mathrm{ANN}(\pi_{\mathcal{F}}(r)) = \pi_{\mathcal{F}}(\mathrm{ANN}(r))$.

Proof.

As usual the proof has two parts.

1. $ANN(\pi_{\mathcal{F},\rho}(r)) \subseteq \pi_{\mathcal{F},\rho}(\mathrm{ANN}(r))$.

   Suppose $at = (d, t, L, U) \in ANN(\pi_{\mathcal{F},\rho}(r))$. By definition, there is a tp-tuple in $\pi_{\mathcal{F},\rho}(r)$ of the form $tp = (d, \gamma)$ in $\pi_{\mathcal{F},\rho}(r)$ such that $at \in ANN(tp)$. Let $\gamma = \gamma_1, \ldots, \gamma_n$ and let $\gamma_i = \langle C_i, D_i, L_i, U_i, \delta_i \rangle$ — hence, there is a unique integer $1 \leq j \leq n$ such that $t \in$ sol(D_j), and $L = \delta_j(D_j, t) \cdot L_j$ and $U = \delta_j(D_j, t) \cdot U_j$. As TP-projection is a multiset operation, there is a unique tuple, $tp^\star$ in r such that $tp.H = tp^\star.H \parallel$ "$A_1 : d.\mathrm{A}_1, ..., A_n : d.\mathrm{A}_\mathrm{n}$" and for all attributes in F, $tp^\star$'s attribute values and those of tp coincide. Hence, $at \in \pi_{\mathcal{F},\rho}(\mathrm{ANN}(tp^\star))$.

2. $\pi_{\mathcal{F},\rho}(\mathrm{ANN}(r)) \subseteq ANN(\pi_{\mathcal{F},\rho}(r))$.

   Suppose $at = (d, t, L, U) \in \pi_{\mathcal{F},\rho}(\mathrm{ANN}(r))$. Then there is an annotated tuple, $at^\star \in ANN(r)$ such that $at = \pi_{\mathcal{F},\rho}(\mathrm{ANN}(at^\star))$. But then there is a TP-tuple $tp^\star$ in r such that $at^\star \in tp^\star$. Hence, $at \in \pi_{\mathcal{F},\rho}(\mathrm{ANN}(tp^\star))$. □

11.9 Join of two TP-relations

In this section, we show how we can correctly implement the join of two TP-relations.

Definition 11.9.1 (join of two TP-relations) *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.* $\square$

For example, $r_3 \bowtie_{pc} r_4$ will be

Data1	Data2	H	C	D	L	U	δ
D1	D2		$(\#)$	$(2/8/1997)$	0.10	0.40	u
D1	D3		$(\#)$	$(3/8/1997)$	0.15	0.20	u

The following shows that our definition of join correctly implements the TATA semantics.

Theorem 41 (Correctness of join) $r \bowtie_\alpha r' = \mathrm{ANN}(r) \bowtie_\alpha \mathrm{ANN}(r')$.

Proof.
As join is a derived operation defined in terms of selection, projection and cartesian product operations, the result follows immediately from Theorems 37, 40 and 39. $\square$

11.10 Tightening of a TP-relation

Recall that TP-tuples can be "loose" in the sense that a TP-tuple may, for example, have two TP-cases, γ_1, γ_2 with $[L_1, U_1] = [0.3, 0.6]$ and $[L_2, U_2] = [0.5, 0.8]$. In this case, it is easy to see that we can tighten the upper bounds of these ranges, and reset them to $[0.3, 0.5]$ and $[0.5, 0.7]$ respectively. The reason is that the two lower bounds sum up to 0.8, thus allowing us an upper bound in each case that is no more than 0.2 more than the lower bound. This is a simple example of tightening. Recall that tightening played a role in establishing Theorem 27, which showed that the annotated expansion of a TP-relation faithfully represents the TP-relation as long as the TP-relation is compact and tight.

In this section, we describe a procedure for tightening a compact TP-relation r. Tight TP-relation r'' can be constructed from r in the following way: Initially, let $r'' = \emptyset$ and let $r = \kappa_\chi(r)$ for some combination function χ. Then for each $tp \in r$, add Tighten-TP-Tuple(tp) to r''. An algorithm for Tighten-TP-Tuple is presented below.

Note that for each $tp_1, tp_2 \in r$, Tighten-TP-Tuple(tp_1) and Tighten-TP-Tuple(tp_2) do not affect each other since r is compact and hence tp_1 and tp_2 must refer to different events. Also note that Tighten-TP-Tuple works much faster when the distribution functions are uniform since here, all upper bounds for a single TP-case will be tightened by the same amount.

Algorithm 8 Tighten-TP-Tuple(tp)

Input: TP-tuple $tp = (d, \gamma)$ where $\gamma = \{\gamma_1, \ldots, \gamma_n\}$ and for all $1 \leq i \leq n$, $\gamma_i = \langle C_i, D_i, L_i, U_i, \delta_i \rangle$

Output: Tight TP-tuple tp'' which is a tightening of tp

Note: In this algorithm, let $\delta(D, C)$ be a "shortcut" for $\Sigma_{t \in \mathrm{sol}(C)} \delta(D, t)$

$L := 0;\ U := 0;$ `// ` $[L, U]$ ` will hold the sum of the lower and upper bounds`

for $i := 1$ **to** n **do** $\{$

$\quad L'_i := \delta_i(D_i, C_i) \cdot L_i;\ \ L := L + L'_i;$

$\quad U'_i := \delta_i(D_i, C_i) \cdot U_i;\ \ U := U + U'_i; \}$

if $U \leq 1.0$ **then** **return** $tp'' := tp;$ `// If ` $U \leq 1.0$`, then ` tp ` was already tight`

$\gamma'' = \emptyset;$

for $i := 1$ **to** n **do** $\{$

$\quad$ **if** $\delta_i \neq u$ **then** $\{$

$\quad\quad$ **foreach** $t \in \mathrm{sol}(C_i)$ $\{$

$\quad\quad\quad L_t := \delta_i(D_i, t) \cdot L_i;\ \ U_t := \delta_i(D_i, t) \cdot U_i;\ \ U' := 1 - (L - L_t);$

<u>if</u> $U' < U_t$ **<u>then</u>** $U_t := U'$;
Add TP-case $\langle(\#), (t), L_t, U_t, u\rangle$ to γ''; } }
<u>else</u> {
$m = |\mathrm{sol}(D_i)|$; $L_t = \frac{L_i}{m}$; $U_t = \frac{U_i}{m}$;
$U' := 1 - (L - L_t)$; $U_i'' := m \cdot \min(U_t, U')$;
Add TP-case $\langle C_i, D_i, L_i, U_i'', \delta_i\rangle$ to γ''; } }
<u>return</u> $tp'' := (d, \gamma'')$;
End-Algorithm

Chapter 12

Query Equivalences in TPA and TATA

This chapter describes our first step towards developing the techniques for query optimizations for TPA and TATA – the query equivalence results. In the near future, we plan to develop cost models for TP-Databases and design enumeration techniques for query execution plans and efficient algorithms for the exploration of the search space of query execution plans.

All the equivalence results below are stated for TP-relations and TPA. However, since we know that annotated relations and TATA is correctly implemented by TPA, we shall prove the similar theorems in TATA.

Theorem 42 *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*

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

Proof. We prove this theorem for atomic constraints C. Then by Theorem 36 this will also hold for non-atomic constraints. For an atomic constraint C three cases are possible.

- C is a data constraint.

  In this case the statement of the theorem is true since a similar statement is true in "standard" relational algebra [105]. The TP-case field of r is not "touched" by either $\mathcal{F}$ or C, and hence both selection and projection on r will behave exactly as they would in the absence of the TP-case.

- C is a temporal constraint.

  We know that $\mathcal{F}$ contains only data fields in it. We also know that C does not refer to any data field. It is easy to see from this that the statement of the theorem holds.

  Indeed, let $at = (d, t, l, u)$ be an annotated tuple from $ANN(\sigma_C(\pi_{\mathcal{F}}(r)))$. Since C is a temporal constraint, we know that t satisfies C. Since $ANN(\sigma_C(r)) \subseteq ANN(r)$, we get: $at \in ANN(\pi_{\mathcal{F}}(r))$. Since $ANN(\pi_{\mathcal{F}}(r)) = \pi_{\mathcal{F}}(ANN(r))$, there exists an annotated tuple $at' = (d', t, l, u) \in ANN(r)$ such that $d = \mathcal{P}(d')$.

  But then, since t satisfies C, $at' \in \sigma_C(ANN(r))$ and $at \in \pi_{\mathcal{F}}(\sigma_C(ANN(r)))$.

  Going the other way, let $at = (d, t, l, u) \in \pi_{\mathcal{F}}(\sigma_C(ANN(r)))$.
  Then, $\sigma_C(ANN(r))$ contains an annotated tuple $at' = (d', t, l, u)$ where $d = \mathcal{P}(d')$. As $(\sigma_C(ANN(r))) \subseteq (ANN(r))$, $at' \in ANN(r)$, we get $at \in \pi_{\mathcal{F}}(ANN(r))$. But $at' \in \sigma_C(ANN(r))$ implies that t satisfies C, and therefore it must be the case that $at \in \sigma_C(\pi_{\mathcal{F}}(ANN(r)))$.

- C is a probabilistic constraint.

  As in the case above, $\mathcal{F}$ contains only data fields while C refers only to the contents of the TP-case field. Applying reasoning similar to the case of temporal constraints we can establish that the statement of the theorem is true in this case as well. $\square$

Theorem 43 *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*

$$\sigma_C(r \times_\alpha r') \equiv \sigma_C(r) \times_\alpha \sigma_C(r') = r \times_\alpha \sigma_C(r') = \sigma_C(r) \times_\alpha r'$$

Proof.

1. $\sigma_{C_1}(r \times_\alpha r') \equiv \sigma_{C_1}(r) \times_\alpha r'$. We break the proof of this fact into two parts.

   - $\sigma_{C_1}(ANN(r \times_\alpha r')) \subseteq \sigma_{C_1}(ANN((r)) \times_\alpha ANN(r')$.

     Let $at'' = (d'', t, l, u) \in \sigma_{C_1}(ANN(r \times_\alpha r'))$. Clearly, $at'' \in ANN(r \times_\alpha r')$. As we know ([25], Theorem 16), $ANN(r \times_\alpha r') = ANN(r) \times_\alpha ANN(r')$, hence $at'' = (d'', t, l, u) \in ANN(r) \times_\alpha ANN(r')$. Then, by the definition of cartesian product on annotated relations, there exist two annotated tuples $at = (d, t, l_1, u_1) \in ANN(r)$ and $at' = (d', t, l_2, u_2) \in ANN(r')$ such that $d = d, d'$ and $[l, u] = [l_1, u_1] \otimes_\alpha [l_2, u_2]$. As $at'' \in \sigma_{C_1}(ANN(r \times_\alpha r'))$ we know that d'' satisfies C_1. But since the only fields mentioned in C_1 are those from the relational schema of r (and $ANN(r)$), it must be the case that d satisfies C_1 as d is the part of d''. But then, $at \in \sigma_{C_1}(r)$, and therefore $at'' \in \sigma_{C_1}(ANN(r)) \times_\alpha ANN(r')$.

   - $\sigma_{C_1}(ANN(r \times_\alpha r')) \supseteq \sigma_{C_1}(ANN((r)) \times_\alpha ANN(r')$.

     Going the other way, we assume that $at'' = (d'', t, l, u) \in \sigma_{C_1}(ANN(r)) \times_\alpha ANN(r')$. Clearly, then there exist two tuples $at = (d, t, l_1, u_1) \in ANN(r)$

and $at' = (d', t, l_2, u_2)\sigma_{C_1}(ANN(r'))$ such that $d = d, d'$ and $[l, u] = [l_1, u_1] \otimes_\alpha [l_2, u_2]$. As $at \in \sigma_{C_1}(ANN(r'))$ we also know that $at \in ANN(r)$. But then $at'' \in ANN(r) \times_\alpha ANN(r') = ANN(r \times_\alpha r')$. Also, since $at \in \sigma_{C_1}(ANN(r'))$ we know that d satisfies C_1. Therefore, since C_1 contains only references to the fields from the relational schema of r ($ANN(r)$), d'' also satisfies C_1 and therefore $at'' \in \sigma_{C_1}(ANN(r \times_\alpha r'))$.

2. $\sigma_{C_2}(r \times_\alpha r') \equiv r \times_\alpha \sigma_{C_2}(r')$.

   The proof of this part of the theorem is symmetric to the proof of part 1.

3. $\sigma_C(r \times_\alpha r') \equiv \sigma_C(r) \times \sigma_C(r')$.

   We prove this statement similarly to the proof of the statement in part 1.

Note: While it is possible to push the data and temporal selections into a cartesian product, the same does not hold for the probabilistic selections. The reason for this 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 when a conjunctive p-strategy is applied. 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 relations.

As it turns out, it is much harder to work with probabilistic select conditions than it is to work with temporal and data select conditions. The full extent of the difference between the temporal/data select conditions and probabilistic select conditions is demonstrated in the following results.

Theorem 44 *Let C be a data or temporal select condition, and χ be a combination function. Then for any tp-relation r*

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

Proof.

- $\sigma_C(\kappa_\chi(r) \subseteq \kappa_\chi(\sigma_C(r))$.

  Let $at = (d, t, l, u) \in ANN(\sigma_C(\kappa_\chi(r))$. We know that at satisfies C, and as C is either a data or temporal constraint, we know that any annotated tuples $at'' = (d, t, l'', u'')$ will also satisfy C.

  As at satisfies C and since $ANN(\sigma_C(r')) = \sigma_C(ANN(r'))$ for any tp-relation r', we know that $at \in ANN(\kappa_\chi(r))$. By the property of Conservativeness of κ_χ there exists at least one annotated tuple $at' = (d, t, l', u') \in ANN(r)$. Now, let $ANN(r)[d, t] = \{at_1, \ldots, at_n\}$, $(\forall 1 \leq i \leq n)(at_i = (d, t, l_i, u_i))$. We know that $[l, u] = \chi(\{[l_1, u_1], \ldots, [l_n, u_n]\})$. As it was noticed above all $at' \in ANN(r)[d, t]$ satisfy C, and therefore, $ANN(r)[d, t] \subseteq ANN(\sigma_C(r))$. But as $ANN(\sigma_C(r)) \subseteq ANN(r)$, $ANN(r)[d, t] = ANN(\sigma_C(r))[d, t]$. Therefore, $at = (d, t, l, u)$ will be in $ANN(\kappa_\chi(\sigma_C(r)))$ which means that $\sigma_C(\kappa_\chi(r) \subseteq \kappa_\chi(\sigma_C(r))$.

- $\sigma_C(\kappa_\chi(r) \supseteq \kappa_\chi(\sigma_C(r))$.

  Let $at = (d, t, l, u) \in ANN(\kappa_\chi(\sigma_C(r)))$. By the Conservativeness property of κ_χ operation we know that there exists at least one annotated tuple $tp' = (d, t, l', u')$ $\in ANN(\sigma_C(r))$. Let now $ANN(\sigma_C(r))[d, t] = \{at_1, \ldots, at_n\}$, where for all $1 \leq i \leq n$ $at_i = (d, t, l_i, u_i)$ and $[l, u] = \chi(\{[l_1, u_1], \ldots, [l_n, u_n]\})$. We know that all these annotated tuples satisfy C. Also, since for any tp-relation r and selection condition C' $ANN(\sigma_{C'}(r)) \subseteq ANN(r)$, $ANN(\sigma_C(r))[d,t] \subseteq ANN(r)$. But since C is a data or temporal constraint, we know that all annotated tuples in the set $ANN(r)[d,t]$ must satisfy it. Therefore $ANN(r)[d,t] = ANN(\sigma_C(r))[d,t]$.

From the latter equality and the fact that $[l,u] = \chi(\{[l_1,u_1],\ldots,[l_n,u_n]\})$ we get that $at = (d,t,l,u) \in ANN(\kappa_\chi(r))$. But we also know that at satisfies C as would any annotated tuple with data part d and temporal part t. Therefore $at \in ANN(\sigma_C(\kappa_\chi(r)))$ which means that $\sigma_C(\kappa_\chi(r) \supseteq \kappa_\chi(\sigma_C(r))$.

As you may notice, the key step in the above proof was the fact that in a tp-relation r, for any data part d and timepoint t it was true that $ANN(\sigma_C(r))[d,t] = ANN(r)[d,t]$ for temporal or data select conditions C. This may not be true in the case when select condition is probabilistic. Therefore the statement of the theorem above is not true for probabilistic select conditions. $\square$

Theorem 45 *Let r and r' be tp-relations, and C be a data or temporal select condition. Then*

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

Proof. We prove that $\sigma_C(r \cap r') \equiv \sigma_C(r) \cap \sigma_C(r')$. Other equalities can be proven similarly.

- $\sigma_C(r \cap r') \subseteq \sigma_C(r) \cap \sigma_C(r')$.

  Let $at = (d,t,l,u) \in ANN(\sigma_C(r \cap r'))$. Then at satisfies C and as C is a data or temporal condition, so would any annotated tuple at'' with data part d and temporal part t. As $ANN(\sigma_C(r \cap r')) \subseteq ANN(r \cap r')$, $at \in r \cap r'$. Two cases are possible: (i) $at \in ANN(r)$ and (ii) $at \in ANN(r')$. We will consider case (i), the remaining case is symmetric.

As $at \in r$ we know that there has to be at least one tuple $at' = (d, t, l'.u') \in ANN(r')$. As stated above, at' satisfies C. In this case $at \in ANN(\sigma_C(r))$ and $at' \in ANN(\sigma_C(r))$. But then, $at \in ANN(\sigma_C(r)) \cap ANN(\sigma_C(r'))$, which shows that $\sigma_C(r \cap r') \subseteq \sigma_C(r) \cap \sigma_C(r')$.

- $\sigma_C(r \cap r') \supseteq \sigma_C(r) \cap \sigma_C(r')$.

  Let $at = (d, t, l, u) \in ANN(\sigma_C(r) \cap \sigma_C(r')) = ANN(\sigma_C(r)) \cap ANN(\sigma_C(r'))$. There are two possibilities: (i) $at \in ANN(\sigma_C(r))$ and (ii) $at \in ANN(\sigma_C(r'))$. We will consider the first one, proof for the second case is symmetric.

  As $at \in ANN(\sigma_C(r))$ we know that at satisfies C and also that $at \in ANN(r)$. Also, as $at \in ANN(\sigma_C(r)) \cap ANN(\sigma_C(r'))$, there has to be at least one tuple $at' = (d, t, l', u') \in ANN(\sigma_C(r'))$. Clearly, at' also satisfies C and $at' \in ANN(r')$. But in this case $\{at, at'\} \subseteq ANN(r \cap r')$ and as at satisfies C, $at \in ANN(\sigma_C(r \cap r'))$ proving that $\sigma_C(r \cap r') \supseteq \sigma_C(r) \cap \sigma_C(r')$. □

Just as in the case of Theorem 44 the proof of this theorem relies on the fact that all data-identical annotated tuples with the same temporal field either satisfy select condition C together or do not satisfy it together. This is true for data and temporal select conditions, but it is not true for probabilistic select conditions as the example below demonstrates.

Example 12.0.1 For simplicity, we illustrate the problems in combining probabilistic selects and intersections using annotated relations. Let annotated relation ann_r contain a tuple $ann = (d, t, 0.3, 0.5)$ and annotated relation ann'_r contain a tuple $ann' = (d, t, 0.5, 0.6)$. Now, let probabilistic selection condition C be $L \leq 0.4$. Clearly, ann satisfies this condition, while ann' does not. $ann_r \cap ann'_r$ will contain both ann and ann' tuples and therefore $\sigma_C(ann_r \cap ann'_r)$ will contain ann.

On the other hand, $\sigma_C(ann_r)$ will contain ann and $\sigma_C(ann'_r)$ will not contain ann', which means that their intersection will not contain ann.

This illustrates that the order in which operations are performed does matter in the case of probabilistic selects.

Theorems 44 and 45 allow us to make the following corollary:

Corollary 8 *Let χ be a combination function, C be a data or temporal select condition and r and r' be two tp-relations. Then*

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

The situation is somewhat better when selection operation follows union. While, because of Theorem 44 it is still impossible to push probabilistic selects into union, it is possible to do so with *multiset union*.

Theorem 46 *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.

- $\sigma_C(r \cup r') \subseteq \sigma_C(r) \cup \sigma_C(r')$.

  Let $at = (d, t, l, u) \in ANN(\sigma_C(r \cup r'))$. In this case at satisfies C and $at \in ANN(r \cup r')$. Two possibilities exist: (i) $at \in ANN(r)$ and (ii) $at \in ANN(r')$. We will consider the first one here, the proof for the other one is symmetric.

  As $at \in ANN(r)$ and at satisfies C, $at \in ANN(\sigma_C(r))$, and therefore $at \in ANN(\sigma_C(r)) \cup ANN(\sigma_C(r')) = ANN(\sigma_C(r) \cup ANN(\sigma_C(r'))$. This proves the desired subset inclusion.

- $\sigma_C(r \cup r') \supseteq \sigma_C(r) \cup \sigma_C(r')$. Let $at = (d, t, l, u) \in ANN(\sigma_C(r) \cup \sigma_C(r')) = ANN(\sigma_C(r)) \cup ANN(\sigma_C(r'))$. As in the previous case, two possible situations exist: (i) $at \in ANN(\sigma_C(r))$ and (ii) $at \in ANN(\sigma_C(r'))$. We consider the first possibility, the proof for the second one will be symmetric.

  As $at \in ANN(\sigma_C(r))$, at satisfies C and $at \in ANN(r)$.
  Then $at \in ANN(r) \cup ANN(r') = ANN(r \cup r')$. As at satisfies C, we concluded that $at \in ANN(\sigma_C(r \cup r'))$. □

This theorem combined with the result of Theorem 44 allows us to draw the following conclusion:

Corollary 9 *Let χ be a combination function, C be a data or temporal select condition and r and r' be two tp-relations. Then*

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

The situation becomes even more interesting when we consider the difference operation. The following two theorems tell us how to push selection down against difference when the select condition is data/temporal and probabilistic.

Theorem 47 *Let C be a data or temporal select condition, and let r and r' be two tp-relations. Then*

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

Proof. We prove the first equality. Second equality can be proven similarly (see also proof of Theorem 48).

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

  Let $at = (d, t, l, u) \in ANN(\sigma_C(r - r'))$. Then at satisfies C and $at \in ANN(r - r')$. In this case $at \in r$ and there are no tuples of the form (d, t, l', u') in r'.

As $at \in r$ and at satisfies C, $at \in ANN(\sigma_C(r))$. Also, we know that no tuple of the form (d, t, l', u') is in $ANN(\sigma_C(r'))$. But then, $at \in ANN(\sigma_C(r)) - ANN(\sigma_C(r')) = ANN(\sigma_C(r) - \sigma_C(r')$, which proves the desired inclusion.

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

  Let $at = (d, t, l, u) \in ANN(\sigma_C(r) - \sigma_C(r'))$. In this case, $at \in ANN(\sigma_C(r))$ and no tuple of the form (d, t, l', u') is in $ANN(\sigma_C(r'))$. Clearly, at satisfies C and as C is either a data or a temporal constraint, any tuple of the form (d, t, l', u') would satisfy C. As $\sigma_C(r) \subset (r)$, $at \in r$. Also, we know that r' does not contain any tuples of the form (d, t, l', u') since otherwise, $\sigma_C(r')$ would have contained them. But then, $at \in ANN(r - r')$ and therefore $at \in ANN(\sigma_C(r - r'))$, which shows that $\sigma_C(r - r') \supseteq \sigma_C(r) - \sigma_C(r')$. □

Theorem 48 *Let C be a probabilistic select condition, and let r and r' be two tp-relations. Then*

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

Proof.

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

  Let $at = (d, t, l, u) \in ANN(\sigma_C(r - r'))$. Then at satisfies C and $at \in ANN(r - r')$. In this case $at \in r$ and there are no tuples of the form (d, t, l', u') in r'.

  As $at \in r$ and at satisfies C, $at \in ANN(\sigma_C(r))$. From this and the fact that no tuple of the form (d, t, l', u') is in r' we conclude that $at \in ANN(\sigma_C(r) - r')$.

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

  Let $at = (d, t, l, u) \in ANN(\sigma_C(r) - r')$. In this case, $at \in ANN(\sigma_C(r))$ and no tuple of the form (d, t, l', u') is in r'. Clearly, at satisfies C and as $\sigma_C(r) \subset (r)$,

$at \in r$. But then, $at \in ANN(r - r')$ and therefore $at \in ANN(\sigma_C(r - r'))$, which shows that $\sigma_C(r - r') \supseteq \sigma_C(r) - \sigma_C(r')$. $\square$

Notice that in the proof of this theorem we never used the fact that C is a probabilistic condition. Therefore this is also a proof that the abovementioned equality holds for the data and temporal select conditions (the second equality of Theorem 47).

Chapter 13

Temporal Probabilistic Databases: A Summary of Results

In this chapter we present a brief summary of our research in Temporal Probabilistic Databases as described in Chapters 9, 10 ,11 and 12 of this thesis.

TP-relations: Storing Temporal Probabilistic Data Compactly

One of the main reasons for developing a new framework for managing temporal uncertainty is that most traditional frameworks for dealing with uncertain information [64], [22], [7] explicitly stored probabilistic information for each instance (timepoint) in the database. In a situation when the space of values for the temporal attribute of a tuple is fairly large, explicitly storing every instance is, as noted by Dyreson and Snodgrass [34, 35] prohibitively expensive.

We have developed a new format for storing temporal probabilistic data. Temporal Probabilistic Tuples (TP-tuples), the units of data storage in our framework, can compactly store information about the probability of an (instantaneous) event occurring at thousands and thousands of different timepoints.

A TP-tuple consists of two parts: data part and TP-case statement. The data part of a TP-tuple is just a collection of relational data fields, stored and managed according to the classical theory of relational databases [105]. In our model, the data fields store the description of the events. A TP-case statement is a special structure designed to store all the necessary temporal and probabilistic information.

The anatomy of a typical TP-tuple is shown in the table below.

Data Part				**TP-Case Statement**				
Description of event				Temporal Info	Probabilistic Info			
Item	Source	Destination	H	C	D	L	U	δ
$I1$	*Rome*	*Vienna*		$(10/10/1999 \sim 12/10/1999)$	$(10/10/1999 \sim 15/10/1999)$	0.4	0.75	u
				$(20/10/1999 \sim 20/10/1999)$	$(16/10/1999 \sim 25/10/1999)$	0.25	0.5	$b, 0.5$

Each case of a TP-case statement contains five attributes, labeled C, D, L, U and δ. Out of these five, one, C is a temporal constraint containing the information about the time when the event described by the data part could have occurred. The remaining quadruple contains the probabilistic information necessary for

1. Computing the probability that the event described by the data part of the tuple could have occurred at some time point described by C.

2. Computing the probability that the event described by the data part of the tuple could have occurred at some time point in any subset of timepoints described by C.

To accommodate this, we store the distribution function δ which specifies how the probability weight is distributed among individual timepoints, its domain, described by a temporal constraint D, and the upper and lower bounds of the probability interval L

and U used to determine the probability intervals for the timepoints in the solution of C. We require that $sol(C) \subseteq sol(D)$. When $sol(C) = sol(D)$, L and U represent the probability interval for C. Otherwise, the probability interval for C will be

$$[L \cdot \sum_{t \in sol(C)} \delta(D,t),\ \ U \cdot \sum_{t \in sol(C)} \delta(D,t)].$$

While this data storage format is inconvenient for display, it contains all information, necessary to correctly compute probability intervals for every subset of $sol(C)$. But since outputting the TP-tuple (especially the one with $sol(C) \neq sol(D)$) in this format can potentially confuse a user, (e.g. by suggesting that L and U values are really the probability interval for C), we have introduced a special *presentation operation* and a corresponding *presentation format* for a TP-tuple. In the presentation format, only the C constraint and the *corresponding probability interval* $[L \cdot \sum_{t \in sol(C)} \delta(D,t),\ U \cdot \sum_{t \in sol(C)} \delta(D,t)]$ are displayed in addition to the data part. Presentation operation Prs converts TP-tuples from their storage format into the output-friendly presentation format. Here is the presentation format of the tuple from the table above.

Item	Source	Destination	C	L	U
$I1$	*Rome*	*Vienna*	$(10/10/1999 \sim 12/10/1999)$	0.2	0.375
			$(20/10/1999 \sim 20/10/1999)$	0.015625	0.03125

Physically keeping TP-tuples in their storage format while outputting it in presentation format allows us to store all the necessary information for current and future query processing, and provide clear unambiguous output.

Annotated Relations,

Theoretical Annotated Temporal Algebra (TATA),

Temporal Probabilistic Algebra (TPA)

After defining the format for storing Temporal Probabilistic data, our main goal became defining the query language which would allow efficient access to the information and would produce correct results. While Dyreson and Snodgrass [34, 35] proposed an extension of Temporal SQL to query the data stored in compact format, we know of no attempts to specify *relational algebra* for such data, in fact Lakshmanan et. al [64] specify it as an open problem for the case of generic Probabilistic Databases.

Our work on creating the appropriate relational algebra for TP-databases can be separated into three parts.

First, we define annotated tuples and annotated relations which are designed to store explicit information about the probabilities of events similarly to the approach taken by ProbView [64]. We describe the process of annotating the TP-relations, by converting every TP-tuple into a collection of annotated tuples which explicitly store the probability of an event described by the data part of the TP-tuples occurring at each timepoint specified by the TP-tuple's TP-case (one timepoint per annotated tuple). While inefficient for the implementation purposes due to the possible massive expansion, annotated tuples are flat and simple to interpret and define relational algebra on.

Second, we defined Theoretical Annotated Temporal Algebra (TATA), a relational algebra that works on annotated relations. We define compaction, intersection, union, selection, difference, cartesian product, projection and join. Definitions of TATA operations in most cases are straightforward, based on the assumptions we make about the operations:

- **Compaction.** This is the analog of duplicate elimination in classical relational algebra. TATA accepts multiple compaction operations, based on the way the user wants to combine the probability intervals of the tuples being compacted.

- **Intersection.** Intersection operation in TATA is viewed as the operation that looks for information common to two relations. Intersection operation proceeds in two steps. First, multiset intersection finds the tuples which contain common information in the two relations and puts them together in one relation. Next, compaction is applied to that relation to combine together (possibly) different probability intervals for the same events and same times contained in different relations. Thus, TATA accepts multiple intersection operations based on the compaction strategy used at the second stage.

- **Union.** Union is very similar to intersection and also is performed in two steps. In the first step the multiset union operation creates the multiset union of the two input relations. The second step, as in case of intersection, is compaction. Union operations are also parameterized by compaction strategies.

- **Difference.** Difference of two annotated relations is the operation that returns the information about the events and their possible occurrences contained in the first relation which is not contained in the second. An interesting feature of the difference operation is that it does not manipulate probabilities.

- **Selection.** There are three types of information stored in annotated relations: data, temporal, probabilistic, and all three can be used in selection conditions of select queries. Data selects are similar to their classical relational counterparts and select annotated tuples based on the contents of their data parts. Temporal selects query the temporal attributes of the annotated tuples and Probabilistic selects query their probabilistic attributes. Each complex query can be broken down into a combination of purely data, purely temporal and purely probabilistic selects.

- **Cartesian Product.** Pure relational cartesian product of two annotated relations would have created a relation with two temporal attributes and two probabilistic intervals, which contradicts the definition of an annotated relation. Instead we are using a “temporal join” which constructs an annotated relation which contains information about the probability of the conjunctions of events described by two input annotated relations (only the annotated tuples which share the temporal attribute are merged into a tuple in the output). Cartesian product operation is parameterized by the assumptions made about the relationships between the events in the input relations (e.g., independence, positive correlation, lack of any knowledge about a possible dependence).

- **Projection.** Projection in TATA is a slightly simplified operation. First, only data fields can be projected out of an annotated relation. Second, when projected out, the values of the fields that are a part of the relation’s primary key remain stored in a special hidden field, which is an obligatory part of the data part of any annotated tuple. Traditionally, projecting out some attributes may result in the occurrence of data-identical tuples in the resulting relation. As data-identical tuples are treated as describing the same event, we would be forced to consider such disjunctive events in our framework. The problem however is that disjunctive events are not instantaneous (e.g.,a disjunction of two instantaneous events is an event that can occur twice - once for each disjunct). Preserving the values of key fields in the hidden field allows us to project out these fields, but keeping tuples’ identity intact, as the contents of the hidden fields also get compared when checking data-identicity of tuples.

- **Join.** Join is defined in a standard way as a combination of selection, projection and cartesian product. As cartesian product has to be parameterized by the assumption about the relationship between the events in the input relations, join also has to be parameterized by such assumptions.

Finally, with TATA playing the role of denotational semantics, we designed Temporal Probabilistic Algebra, a relational algebra which works directly on TP-relations while implementing the semantics of TATA operations. Given an operation Op^A of TATA and Op^T of TPA we say that Op^T correctly implements Op^A iff $ANN \circ Op^A = Op^T \circ ANN$. We showed that compaction, intersection, union, selection, difference, cartesian product, projection and join as defined in TPA correctly implement the respective TATA operations.

In addition, three operations, specific to the format of TP-tuples and therefore not found in TATA are considered in TPA.

- **Presentation** operation has been discussed above. As the result of this operation is not in a TP-relation format, it cannot be used inside a TPA expression. But the results of TPA queries may (at the end) be converted into presentation format by applying Prs operation to them.

- **Compression** refers to a family of different heuristic-driven operations which attempt to decrease the size of TP-case statements in TP-tuples. While not necessary for the rest of the algebra to work, compressions allow the user to keep TP-tuples in a terse format. This may become an issue after a number of joins/cartesian products have been performed, as these operations can increase the number of TP-cases in the results.

- **Tightening** is an operation that checks the upper bounds of the in the TP-case statement and tries to exclude redundant information from them.

Temporal Probabilistic Algebra definitions are mostly done in algorithmic manner, describing how the resulting relation is constructed from the input relations. Where definitions of operations are declarative (such as in the case of compaction operations), they are accompanied by algorithms that perform the operations. The correctness proofs guarantee that the operations preserve the ideas expressed declaratively in TATA.

Query Equivalences in TPA and TATA: towards Temporal Probabilistic Query Optimization

As with every newly introduced query language, query optimization problem arises in our work as well. In this thesis, the foundation of the future query optimization work is presented: the query equivalences which hold in TPA and TATA.

While some of the equivalences proved in the thesis are similar to query equivalences in classical relational algebra [105], there is a number of different results as well. One of the main features of TATA and TPA w.r.t. query equivalence is that different types of select exhibit different behavior with temporal select being the best target for potential optimization and probabilistic select being the worst.

The equivalence results proved in this thesis are summarized in Table 13.1.

Equivalence	**Condition**
$\sigma_C(\pi_{\mathcal{F}}(r)) \equiv \pi_{\mathcal{F}}(\sigma_C(r))$	C does not involve fields from $\mathcal{F}$
$\sigma_{C_1}(r \times_\alpha r') \equiv \sigma_{C_1}(r) \times_\alpha r'$ $\sigma_{C_2}(r \times_\alpha r') \equiv r \times_\alpha \sigma_{C_2}(r')$	C_1 is data condition involving only fields of r C_2 is data condition involving only fields of r'
$\sigma_C(r \times_\alpha r') \equiv \sigma_C(r) \times_\alpha \sigma_C(r') \equiv$ $\equiv r \times_\alpha \sigma_C(r') \equiv \sigma_C(r) \times_\alpha r'$	C is a temporal condition
$\sigma_C(\kappa_\chi(r)) \equiv \kappa(\sigma_C(r'))$	C is data or temporal condition
$\sigma_C(r \cap r') \equiv \sigma_C(r) \cap \sigma_C(r') \equiv$ $\equiv r \cap \sigma_C(r') \equiv \sigma_C(r) \cap r'$	C is a data or temporal condition
$\sigma_C(r \cap_\chi r') \equiv \sigma_C(r) \cap_\chi \sigma_C(r') \equiv$ $\equiv r \cap_\chi \sigma_C(r') \equiv \sigma_C(r) \cap_\chi r'$	C is a data or temporal condition
$\sigma_C(r \cup r') \equiv \sigma_C(r) \cup \sigma_C(r')$	C is an arbitrary condition
$\sigma_C(r \cup_\chi r') \equiv \sigma_C(r) \cup_\chi \sigma_C(r')$	C is a data or temporal condition.
$\sigma_C(r - r') \equiv \sigma_C(r) - r'$	C is an **arbitrary** condition
$\sigma_C(r - r') \equiv \sigma_C(r) - \sigma_C(r')$	C is a **data** or **temporal** condition.

Table 13.1: Query Equivelences in TATA and TPA

Chapter 14

Beyond the Databases: Temporal Probabilistic Logic Programs

14.1 Temporal Probabilistic Logic Programs for Instantaneous Events

In some applications involving temporal probabilistic information, in addition to the ability to store and query the data provided by TP-Databases, it may be benefitial to be able to draw inferences from this data. Applications that involve prediction and estimation of future events based on the past history are very sensitive to expert opinions. While the statistical data for these applications stored in a TP-Database is not affected by any particular opinion, the conclusions drawn based on this information are. Experts can enrich the system with the knowledge about connections between different events, or even with their view of the information (pessimists woukd expect the true probabilities to be below the ones stored in the TP-Database, while optimists would tend to assume that the true probabilities are higher). Therefore, it seems reasonable to desire the framework that would allow one to reason with temporal probabilistic information.

Temporal Probabilistic Logic Programs, described in this chapter is the first step towards developing such framework.

Our main goal here is to define a framework which would be to a certain degree "compatible" with our TP-Database framework. While below, we will not adhere strictly to the letter of TP-Databases, and use somewhat different notation, the key ideas behind the syntax and semantics of the Logic Programming language introduced below are very similar to those found in TP-Databases. In fact, our use of temporal constraints and calendars (albeit, in a somewhat simplified manner) and probability distribution functions is a direct influence of the data model we have developed for TP-Databases. Similarly, formula tp-functions, which play the role of models of the Temporal Probabilistic Logic Programs (TPLPs) are very similar in spirit to annotated relations.

The desire to follow the ideas behind the TP-Database framework determined our choice of instantaneous events as the only ones allowed in the framework described here. This, actually has some interesting consequences, such as, the proposed TPLP framework not being an extension of classical logic programs. More discussion on this topic can be found in Section 16 (Future Work), as we are planning to develop a new TPLP framework which would incorporate both instantaneous and non-instantaneous events, as well as regular statements, which do not change with time.

We start our description of the Temporal Probabilistic Logic Programming framework with a quick recapture of temporal constraints and probability distribution functions. This is followed by the description of the syntax of temporal probabilistic programs (tp-programs), model theoretic and fixpoint semantics, which are shown to coincide, and finally – a proof procedure, shown to be sound and complete.

14.1.1 Temporal Constraints and Probability Distribution Functions (revisited)

We use temporal constraints and probability distribution functions. We have already described both in the section devoted to Temporal Probabilistic Databases. In our work in Logic Programming, we can assume certain simplifications, which we chose not to in the TP-Database framework. Because of that, we will restate the definitions of temporal constraints and probability distribution functions below, keeping in mind that these definitions have deep roots in the definitions in Sections 9.2.1 and 9.2.2.

Temporal Constraints

We assume here that time is discrete and finite. A calendar τ is any initial segment $\{0, \ldots, t_{\max}\}$ of the set of natural numbers. Elements $t \in \tau$ will be called timepoints. The application developer may choose $t_{\max}$ in any way deemed appropriate for the application.

As witnessed in Section 9.2.1 there exist a variety of formalizations of the notion of a calendar that allow one to refer to timepoints in a more sophisticated format (e.g., by year, month, day of the month, hour and minute), which is commonly used in practice. Regardless of how the calendar is defined, all timepoints in it are linearly ordered. This makes all *finite* calendars to be isomorphic to the initial segments of natural numbers. Using natural numbers as a calendar will allow us to concentrate on the important aspects of the proposed framework and not to spend out time dealing with peculiarities of particular calendars.[1]

[1]In the Temporal Probabilistic Database work one of our main goals was to make TP-database framework implementable and useful for different applications. In the work on Temporal Probabilistic Logic Programs we have a more theoretical interest in establishing what is involved in creating an inference

On the other hand there is also a clear way to extend the proposed temporal framework to work with a calendar defined in such an extended format. In fact our choice of the calendar is dictated mostly by our desire to have a compact temporal constraint language.

The language of temporal constraints based on a given calendar $\tau = \{0, \ldots, t_{\max}\}$ consists of a set of constants $\{0, 1, \ldots t_{\max}\}$ for each element of τ, a set of temporal variables $x, x_1, \ldots$, comparison symbols $=, \neq, >, <, \geq, \leq, \sim$ and boolean connectives $\wedge, \vee, \neg$. Constants and variables are the only terms of the language.

If t_1, t_2, t_3 are terms, and $op \in \{=, \neq, >, <, \geq, \leq\}$, then $t_1 \ op \ t_2$ and $t_1 : t_2 \sim t_3$ are atomic temporal constraints. The set of temporal constraints is defined as the closure of the set of atomic temporal constraints under conjunction, disjunction and negation.

We will be especially interested in the class $\mathcal{C}_1^\tau$ of temporal constraints which use exactly one temporal variable. In what follows we will only consider such constraints.

A **solution set** of an atomic temporal constraint $c \in \mathcal{C}_1^\tau$ (denoted $sol(c)$) is defined as follows:

Constraint	Solution Set
$x = t$	$\{t\}$
$x \neq t$	$\{t' \in \tau \| t' \neq t\}$
$x > t$	$\{t' \in \tau \| t' > t\}$
$x < t$	$\{t' \in \tau \| t' < t\}$
$x \geq t$	$\{t' \in \tau \| t' \geq t\}$
$x \leq t$	$\{t' \in \tau \| t' \leq t\}$
$x : t_1 \sim t_2$	$\{t' \in \tau \| t_1 \leq t' \leq t_2\}$

mechanism under temporal uncertainty, and are more willing to abstract from the "real-world" data.

The solution set of a conjunction of two constraints c and c' is defined as the intersection of $sol(c)$ and $sol(c')$: $sol(c \wedge c') = sol(c) \cap sol(c')$ (here we assume that both constraints have the same temporal variable).

The solution set of a disjunction of two constraints c and c' is defined as union of $sol(c)$ and $sol(c')$: $sol(c \vee c') = sol(c) \cup sol(c')$

The solution set for a negation of a constraint c is defined as $\tau - sol(c)$. In the rest of the paper we will use t as a shorthand for a constraint $x = t$, or for any other constraint c such that $sol(c) = \{t\}$.

Probabilistic Distribution Functions

The second preliminary notion we want to introduce is probabilistic distribution functions. Distribution functions will help us determine how the probability is distributed among the timepoints in the solutions of temporal constraints.

Probabilistic distribution functions help us determine how a probability mass is distributed among the timepoints in the solutions of temporal constraints.

Definition 14.1.1 *Let c be a temporal constraint. A* probabilistic distribution function *δ_c takes as input timepoints $t \in sol(c)$ and returns a real number from the interval $[0, 1]$, such that $\sum_{t \in sol(t)} \delta_c(t) \leq 1$. δ_c is called* complete *if this sum is* equal *to 1.*

In the rest of the chapter, we sometimes abuse notation and write δ and $\delta(c, t)$ instead of δ_c and $\delta_c(t)$ respectively. Furthermore, if c' is a temporal constraint, and $sol(c') \subseteq sol(c)$ then we use $\delta_c(c')$ ($\delta(c, c')$) as shorthand for $\sum_{t \in sol(c')} \delta_c(t)$.

Besides probabilistic distribution functions we will also consider one special class of non-probabilistic distribution functions: $\top_c$ defined as $\top_c(t) = 1$ for all $t \in sol(c)$. Unless $|sol(c)| = 1$, $\top_c$ is clearly not a probabilistic distribution. However, this function will find limited use in our framework, allowing us to represent some information in a compressed form.

If c is such that $sol(c) = \{t\}$ is a singleton, there exists only one complete distribution function associated with it: $\delta_c(t) = 1$. In the rest of the chapter we will use the notation $\sharp$ to refer to such functions. Also, for the purposes of clarity, in the examples that follow for relatively small temporal constraints we will write distribution functions in the form of enumeration: e.g. $\delta = \{0.5, 0.25, 0.125, 0.125\}$ defined on a solution set of, say, the constraint $c = x : 1 \sim 4$ means that $\delta(c, 1) = 0.5$; $\delta(c, 2) = 0.25$ and so forth.

14.1.2 Temporal Probabilistic Programs: Syntax

Let L be a function-free language generated by finitely many constant and predicate symbols. If p is an n-ary predicate symbol and $a_1, \ldots, a_n$ are either constants or 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 14.1.2 *Let $A_1, \ldots A_k$ all be 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 14.1.3 *A* temporal probabilistic annotation *is a quadruple $\langle c, l, u, \delta_c \rangle$ where $c = c(t)$ is a temporal constraint with one free variable t, $0 \leq l \leq u \leq 1$ and δ_c is a probabilistic distribution function defined on the set $sol(c)$.* □

Definition 14.1.4 *Let $F = A_1 * \ldots * A_k$ be an event atom ($* \in \{\vee, \wedge\}$) and $\mu = \langle c, l, u, \delta_c \rangle$ be a tp-annotation. Then $F : \mu$ is a* tp-annotated basic formula *or just an* annotated basic formula. □

Intuitively when F is ground, $F : \langle c, l, u, \delta_c \rangle$ represents the fact that the events described by F happened at some timepoint $t \in sol(c)$ with a probability in the interval $[l, u]$ and that this probability is distributed among the elements of $sol(c)$ according to the probability distribution function δ_c.

Definition 14.1.5 *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 14.1.6 *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 14.1.1 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 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 packages sent to Paris via express-mail. Three facts about shipments complete this simple program.

We should note one inconvenience of our tp-programs. The rules above provide the information about the possible arrival of packages *sent only on day 1*. It would be desirable to have a more generic version of the program which would describe the arrival of the packages sent on any arbitrary date. To do so, one may extend the syntax to employ special *annotation variables* and arithmetics in the temporal constraints. E.g., the first rule of the program above could be transformed to:

arrived($Item$,$Place$): $[x : t + 2 \sim t + 3,\, 0.5, 0.8, \{.5, .3, .2\}]$

$\longleftarrow$ sent($Item$,$Place$): $[x = t, 0.9, 1, \sharp]$.

Here t is an annotation variable. Probabilistic logic programs with variable annotations had been extensively studied by Ng and Subrahmanian [83, 84] and Dekhtyar and Subrahmanian [22] (also Part II), with definitions of annotations borrowed from the work on Generalized Annotated Programs [56] by Kifer and Subrahmanian. Here, we will consider only constant annotations.

14.1.3 Temporal Probabilistic Logic Programs: Model Theory and Fixpoint Semantics

We now provide a model theory and fixpoint procedure to compute the semantics of tp-programs.

As models of our programs we will use formula tp-functions, which are introduced below. Fixpoint procedure then will map formula tp-functions to formula tp-functions

Definition 14.1.7 *Let $[l, u]$ and $[l', u']$ be two subintervals of $[0, 1]$. We define two operations $\oplus$ and $\otimes$ as follows:*

$$[l, u] \otimes [l', u'] = [\max(0, l + l' - 1), \min(u, u')]$$

$$[l, u] \oplus [l', u'] = [\max(l, l'), \min(u + u', 1)]$$

As one can notice, the $\otimes$ and $\oplus$ operations defined above are nothing but the computations of the probability of conjunction ($\otimes$) and disjunction ($\oplus$) of two events under the assumption of ignorance. We have discussed similar operations when studying probabilistic strategies (Chapter 3). The notation is borrowed from [82]. Using these operators we can now define atomic and formula tp-functions.

Definition 14.1.8 *An atomic tp-function is a function $f : B_L \times \tau \longrightarrow 2^{[0,1]}$ such that*

$$\sum_{t \in \tau; f(A,t)=[l,u]} l \leq 1.$$

Given an atomic tp-function f we can extend it onto the set of all basic tp-formulas as as follows:

(i) $h_f(A, t) = f(A, t)$ *for all* $A \in B_L$, $t \in \tau$;

(ii) Let $F = A_1 \wedge \ldots A_k$ *and* $G = B_1 \wedge \ldots B_m$. *Then* $h_f(F \wedge G, t) = h_f(F, t) \otimes h_f(G, t)$.

(iii) Let $F = A_1 \vee \ldots A_k$ *and* $G = B_1 \vee \ldots B_m$. *Then* $h_f(F \wedge G, t) = h_f(F, t) \oplus h_f(G, t)$.

□

We can think of tp-functions in the following way. Atomic tp-functions represent the (known) assignment of probability intervals to statements of the form "Event represented by atom A occurs exactly at time t"[2]. Given an atomic tp-function f a corresponding formula function extends the probability assignment onto conjunctions and disjunctions of atoms by using the ignorance assumption about the relationship between atomic events.

Definition 14.1.9 *An atomic tp-function f is called fully defined iff for all atoms A and timepoints t, $f(A, t) \neq \emptyset$.*

A formula tp-function h_f is called fully defined iff for all event atoms F and timepoints t, $h_f(F, t) \neq \emptyset$. □

A simple lemma shows that the concepts of fully defined atomic and formula tp-functions coincide.

Lemma 16 *An atomic tp-function f is fully defined iff the formula tp-functions h_f is fully defined*

Proof.

- Let f be fully defined. We now show that h_f is also fully defined. Consider formula F. We show that $h_f(F, t) \neq \emptyset$ for all $t \in \tau$ by induction on the size of (number of atoms in) F. The base case, $F = A \in B_L$ and $h_f(F, t) = f(F, t) \neq \emptyset$ since f is fully defined. For induction step assume for all G of size less than n the statement is true and that $F = A_1 * \ldots * A_n$. Then $F = (A_1 * \ldots * A_{n-1}) * A_n$. Then, if $* = \wedge$ then $h_F(F, t) = h_f(A_1 * \ldots * A_{n-1}), t) \otimes h_f(A_n, t)$. Since both $h_f(A_1 * \ldots * A_{n-1}), t) \neq \emptyset$ and $h_f(A_n, t) \neq \emptyset$, $h_f(F, t) \neq \emptyset$. Indeed, if

[2] We will abuse notation and say "event A" instead of "event represented by A" below.

$h_f(A_1 * \ldots * A_{n-1}), t) = [l, u] \subseteq [0, 1]$ and $h_f(A_n, t) = [l', u'] \subseteq [0, 1]$ then $h_f(A_1 * \ldots * A_{n-1}), t) \otimes h_f(A_n, t) = [\max(0, l + l' - 1), \min(u, u')] \neq \emptyset$[3]. If $* = \vee$, $h_f(F, t) = h_f(A_1 * \ldots * A_{n-1}), t) \oplus h_f(A_n, t) = [\max(l, l'), \min(1, u + u')] \neq \emptyset$[4] which proves the induction step.

- Let f NOT be fully defined. Then there exists an atom A and a timepoint t for which $f(A, t) = \emptyset$. But then, $h_f(A, t) = \emptyset$ as well, therefore h_f is not fully defined. □

We can now define an order on atomic tp-functions (and consequently on formula tp-functions).

Definition 14.1.10 *Let f and g be two atomic tp-functions. We say that $f \leq g$ **iff** for all atoms A and for all timepoints t $f(A, t) \supseteq g(A, t)$.*

*Let h_f and h_g be two formula tp-functions. We say that $h_f \leq h_g$ **iff** for all formulas F and for all timepoints t $f(F, t) \supseteq g(F, t)$.* □

Once again, it is easy to show that the order on formula tp-functions is completely dependent of the order of atomic tp-functions.

Lemma 17 *Let f and g be two formula functions. $f \leq g$ **iff** $h_f \leq h_g$.*

Proof. Clearly if $f \not\leq g$ then $h_f \not\leq h_g$ as well as atomic tp-functions and their corresponding formula tp-functions coincide on atoms. $f \not\leq g$ means that there exist some atom A and timepoint t that $f(A, t) \not\supseteq g(A, t)$, which means that $h_f(A, t) \not\supseteq h_g(A, t)$.

---

[3] We know that $l \leq u$ and $l' \leq u'$. $l + l' - 1 \leq \min(l, l')$, $0 \leq \min(u, u')$ and therefore $\max(0, l + l' - 1) \leq \min(u, u')$.

[4] Since $l \leq u$ and $l' \leq u'$, $\max(l, l') \leq u + u'$, also we know that $l \leq 1 and l' \leq 1$.

Now we need to show that if $f \leq g$ then $h_f \leq h_g$. Let F be an arbitrary formula and t be any timepoint. We prove that $h_f(F,t) \supseteq h_g(F,t)$ using induction on size of F.

In the **base case**, size of F is 1 and F is some atom A. But we know that: $h_f(A,t) = f(A,t) \supseteq g(A,t) = h_g(F,t)$.

For the **induction step** let us assume that the statement of the lemma holds for all formulas of size less than k and that size of F is n.

If $F = A_1 \wedge \ldots \wedge A_k$, then $F = (A_1 \wedge \ldots \wedge A_{k-1}) \wedge A_k$. We know that $h_f(A_1 \wedge \ldots \wedge A_{k-1}, t) = [l_1, u_1] \supseteq [l'_1, u'_1] = h_g(A_1 \wedge \ldots \wedge A_{k-1}, t)$ and $h_f(A_k, t) = [l_2, u_2] \supseteq [l'_2, u'_2] = h_g(A_k, t)$. $h_f(F,t) = h_f(A_1 \wedge \ldots \wedge A_{k-1}) \otimes h_f(A_k) = [\max(0, l_1 + l_2 - 1), \min(u_1, u_2)]$ and $h_g(F,t) = h_g(A_1 \wedge \ldots \wedge A_{k-1}) \otimes h_g(A_k) = [\max(0, l'_1 + l'_2 - 1), \min(u'_1, u'_2)]$.

We need to show now that $\max(0, l_1 + l_2 - 1), \min(u_1, u_2)] \supseteq [\max(0, l'_1 + l'_2 - 1), \min(u'_1, u'_2)]$. Since $l_1 \leq l'_1$ and $l_2 \leq l'_2$, we know that $l_1 + l_2 - 1 \leq l'_1 + l'_2 - 1$ and therefore $\max(0, l_1 + l_2 - 1), \max(0, l'_1 + l'_2 - 1)$. Also, since $u_1 \geq u'_1$ and $u_2 \geq u'_2$, we get $\min(u_1, u_2) \geq \min(u'_1, u'_2)$. Combined together this gives us the desired set inclusion.

Similarly we can proceed if $F = A_1 \vee \ldots \vee A_k = (A_1 \vee \ldots \vee A_{k-1}) \vee A_k$. We know that $h_f(A_1 \vee \ldots \vee A_{k-1}, t) = [l_1, u_1] \supseteq [l'_1, u'_1] = h_g(A_1 \vee \ldots \vee A_{k-1}, t)$ and $h_f(A_k, t) = [l_2, u_2] \supseteq [l'_2, u'_2] = h_g(A_k, t)$. Then $h_f(F,t) = h_f(A_1 \vee \ldots \vee A_{k-1}) \oplus h_f(A_k) = [\max(l_1, l_2), \min(1, u_1 + u_2)]$ and $h_g(F,t) = h_g(A_1 \vee \ldots \vee A_{k-1}) \oplus h_g(A_k) = [\max(l'_1, l'_2), \min(u'_1 + u'_2, 1)]$.

We now show that $[\max(l_1, l_2), \min(1, u_1 + u_2)] \supseteq [\max(l'_1, l'_2), \min(u'_1 + u'_2, 1)]$. Since $l_1 \leq l'_1$ and $l_2 \leq l'_2$, $\max(l_1, l_2) \leq \max(l'_1, l'_2)$. Also since $u_1 \geq u'_1$ and $u_2 \geq u'_2$, we get $u_1 + u_2 \geq u'_1 + u'_2$ which means that $\min(u_1 + u_2, 1) \geq \min(u'_1 + u'_2, 1)$. Combined together this gives us the desired set inclusion. $\square$

Let $\mathcal{AF}_{TP}$ denote the set of all atomic tp-functions over language L and calendar τ

and let $\mathcal{FF}_{TP}$ denote the set of all formula tp-functions corresponding to $\mathcal{AF}_{TP}$.

Lemma 18 *$\langle \mathcal{AF}_{TP}, \leq \rangle$ and $\langle \mathcal{FF}_{TP}, \leq \rangle$ are both lattices.*

Proof.

(1) $\langle \mathcal{AF}_{TP}, \leq \rangle$ is a lattice.

To show this we must define operations of join and meet on $\mathcal{AF}_{TP}$. We do it as follows. Let $S = \{[l_1, u_1], [l_2, u_2], \ldots\}$ be a set of subintervals of $[0, 1]$. As $closure(S)$ we will denote the tightest interval $[l, u]$ such that $(\forall i)([l, u] \supseteq [l_i, u_i])$. We note that such an interval must exist as all intervals in S are bounded by $[0, 1]$ and therefore there must be $\inf_{i \to \infty}(l_i)$ and $\sup_{i \to \infty}(u_i)$.
Then $closure(S) = [\inf_{i \to \infty}(l_i), \sup_{i \to \infty}(u_i)]$. Now we can define meet and join.

Let $\mathcal{F}$ be a subset of $\mathcal{AF}_{TP}$. Then

$$\bigsqcup \mathcal{F}(A, t) = \cap_{f \in \mathcal{F}} f(A, t)$$

$$\sqcap \mathcal{F}(A, t) = closure(\{f(A, t) | f \in \mathcal{F})$$

and we can define the bottom at top elements of the lattice as follows: $\top(A, t) = \emptyset$; $\perp(A, t) = [0, 1]$.

It is clear that for any $f \in \mathcal{F}$ $[0, 1] \supseteq \sqcap \mathcal{F}(A, t) \supseteq f(A, t) \supseteq \bigsqcup \mathcal{F}(A, t) \supseteq \emptyset$, hence $\perp \leq \sqcap \mathcal{F} \leq f \leq \bigsqcup \mathcal{F} \leq \top$.

(2) Now we need to define meet and join operations on $\mathcal{FF}_{TP}$. We do this in the following manner. Let $\mathcal{F}$ be a subset of $\mathcal{AF}_{TP}$. Let $\mathcal{H}_{\mathcal{F}}$ be the corresponding subset of $\mathcal{FF}_{TP}$. Then:

$$\bigsqcup \mathcal{H}_{\mathcal{F}} = h_{\bigsqcup \mathcal{F}}$$

$$\sqcap\mathcal{H}_{\mathcal{F}} = h_{\sqcap\mathcal{F}}$$

and $\bot_{\mathcal{F}\mathcal{F}_{TP}} = h_{\bot_{\mathcal{A}\mathcal{F}_{TP}}}$; $\top_{\mathcal{F}\mathcal{F}_{TP}} = h_{\top_{\mathcal{A}\mathcal{F}_{TP}}}$.

By Lemma 17 we know that since for all $f \in \mathcal{F}$ $\sqcap\mathcal{F} \leq f \leq \bigsqcup\mathcal{F}$, it must be the case that $h_{\sqcap\mathcal{F}} \leq h_f \leq h_{\bigsqcup\mathcal{F}}$. □

We are now ready to define what it means for a formula tp-function to satisfy a TP-program.

Definition 14.1.11 (satisfaction) *Let h_f be a formula tp-function.*

- $h_f \models F : [c, l, u, \delta]$ ***iff*** $(\forall t \in sol(c))(h_f(F, t) \subseteq [\delta(c, t) \cdot l, \delta(c, t) \cdot u])$
- $h_f \models F_1 : [c_1, l_1, u_1, \delta_1] \wedge \ldots \wedge F_n : [c_n, l_n, u_n, \delta_n]$ ***iff*** $(\forall 1 \leq i \leq n) h_f \models F_i : [c_i, l_i, u_i, \delta_i]$.
- $h_f \models F : [c, l, u, \delta] \longleftarrow F_1 : [c_1, l_1, u_1, \delta_1] \wedge \ldots \wedge F_n : [c_n, l_n, u_n, \delta_n]$ ***iff*** $h_f \models F : [c, l, u, \delta]$ ***or*** $h_f \not\models F_1 : [c_1, l_1, u_1, \delta_1] \wedge \ldots \wedge F_n : [c_n, l_n, u_n, \delta_n]$.
- $h_f \models (\forall x)F : [c, l, u, \delta]$ *iff* $h_f \models (F : [c, l, u, \delta])(s/x)$ *for every ground term* s.
- $h_f \models (\exists x)F : [c, l, u, \delta]$ *iff* $h_f \models (F : [c, l, u, \delta])(s/x)$ *for some ground term* s.
- h_f *satisfies a tp-program* P *(is a model of* P*)* ***iff*** h_f *satisfies every clause in* P*. A tp-program* P *is called consistent iff it has at least one model.*

We say that tp-program P *satisfies* $F : [c, l, u, \delta]$ *iff every model* h_f *of* P *satisfies* $F : [c, l, u, \delta]$. □

Basically, a tp-formula function h_f satisfies basic formula $F : [c, l, u, \delta]$ iff the value of $h_f(F, t)$ is a subset of the probability interval $[\delta(c, t) \cdot l, \delta(c, t) \cdot u]$ for all timepoints t that are solutions of c.

Definition 14.1.12 *Let P be a tp-program. We define operator $T_P : \mathcal{AF}_{TP} \longrightarrow \mathcal{AF}_{TP}$ as follows:*

$T_P(f)(A,t) = \cap L$, where $L = \{[l,u] | A : [c, l', u', \delta] \longleftarrow Body \in ground(P); h_f \models Body; t \in sol(c); [l,u] = [\delta(c,t) \cdot l', \delta(c,t) \cdot u']\}$.

This operator can extended to map functions from $\mathcal{FF}_{TP}$ to functions from $\mathcal{FF}_{TP}$ as follows:

(i) $A \in B_L$.

$T_P(h_f)(A,t) = T_P(f)(A,t)$.

(ii) $F = A_1 \wedge \ldots \wedge A_k$, $G = B_1 \wedge \ldots \wedge B_m$.

$T_P(h_f)(F \wedge G, t) = T_P(h_f)(F) \otimes T_P(h_f)(G)$.

(iii) $F = A_1 \vee \ldots \vee A_k$, $G = B_1 \vee \ldots \vee B_m$.

$T_P(h_f)(F \wedge G, t) = T_P(h_f)(F) \oplus T_P(h_f)(G)$.

$\square$

The application of the T_P operator to a formula tp-function h_f can be explained as follows. On atoms, T_P operator looks at the clauses of the program P which are satisfied by h_f. For each atom, and for each timepoint covered by the heads of firing rules for this atom, the new value is computed as the intersection of all probability ranges derived from these heads. This is in fact what the T_P operator applied to atoms is doing. Since a formula tp-function is uniquely defined by its restriction to atoms, the extension of the T_P operator onto complex formulas is straightforward, and is done by applying our $\otimes$ and $\oplus$ operators.

It is easy to notice from the definition of T_P that $T_P(h_f) = h_{T_P(f)}$.

The following theorems show that the T_P operator is monotonic.

Theorem 49 *Let $f \leq g$ be two atomic tp-functions. Then $T_P(f) \leq T_P(g)$.*

Proof.

Let $A \in B_L$ and $t \in \tau$.

Let

$$C_1 \equiv A : [c_1, l_1, u_1, \delta_1] \longleftarrow Body_1$$

$$\ldots$$

$$C_k \equiv A : [c_k, l_k, u_k, \delta_k] \longleftarrow Body_k,$$

be all clauses from $ground(P)$ such that $t \in sol(c_i), 1 \leq i \leq k$ (i.e. all clauses from P which contain information about the probability of A at time t).

We will use the following proposition:

Proposition 17 *Let $h_f \leq h_g$ and $F : [c, l, u, \delta]$ be a basic formula. Then $h_f \models F : [c, l, u, \delta] \Longrightarrow h_g \models F : [c, l, u, \delta]$.*

Proof. By definition of satisfaction, $h_f \models F : [c, l, u, \delta]$, means $(\forall t \in sol(c)) h_f(F, t) \subseteq [\delta(c, t) \cdot l, \delta(c, t) \cdot u]$. Since $h_f \leq h_g$, we know that $(\forall t \in sol(c)) h_g(F, t) \subseteq h_f(F, t)$. Combining the two subset relations we get $(\forall t \in sol(c)) h_g(F, t) \subseteq [\delta(c, t) \cdot l, \delta(c, t) \cdot u]$ which in turn means $h_g \models F : [c, l, u, \delta]$ □

From Proposition 17 it follows (since $Body_i$ is a conjunction of basic formulas) that $(\forall 1 \leq i \leq k) h_f \models Body_i \Longrightarrow h_g \models Body_i$. Let $\mathcal{C}_{h_f}$ and $\mathcal{C}_{h_g}$ denote the sets of clauses from $\{C_1, \ldots, C_k\}$ such that their bodies are satisfied by h_f and h_g respectively. From the above we know that $\mathcal{C}_{h_f} \subseteq \mathcal{C}_{h_g}$.

Let $L_{h_f} = \{[l, u] | A : [c, l', u', \delta] \longleftarrow Body \in \mathcal{C}_{h_f}\}$ and $L_{h_g} = \{[l, u] | A : [c, l', u', \delta] \longleftarrow Body \in \mathcal{C}_{h_g}\}$. Clearly, $L_{h_f} \subseteq L_{h_g}$.

By definition of T_P operator $T_P(f)(A,t) = \cap L_{h_f}$ and $T_P(h_g)(A,t) = \cap L_{h_g}$. Since $L_{h_f} \subseteq L_{h_g}$, it has to be the case that $\cap L_{h_f} \supseteq \cap L_{h_g}$, i.e., $T_P(h_f)(A,t) \supseteq T_P(h_g)(A,t)$ which proves the monotonicity of the T_P operator. $\square$

Now we can prove that T_P is monotonic on formula tp-functions as well.

Theorem 50 *Let $h_f \leq h_g$ be two formula tp-functions. Then $T_P(h_f) \leq T_P(h_g)$.*

Proof. The theorem is an (almost) immediate corollary of Theorem 49 We know that $T_P(h_f) = h_{T_P(f)}$ and $T_P(h_g) = h_{T_P(g)}$. Since $h_f \leq h_g$, we also know that $f \leq g$. Then, by Theorem 49, $T_P(f) \leq T_P(g)$. But then, by Lemma 17, $T_P(h_f) = h_{T_P(f)} \leq h_{T_P(g)} = T_P(h_g)$. $\square$

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

Definition 14.1.13 *Let P be a tp-program.*

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

$\square$

The following theorem establishes a close connection between the least fixpoint of the T_P operator and the model theory for tp-programs. Its first part establishes a crucial property of formula tp-functions w.r.t. satisfaction of tp-programs. Other parts serve to justify our iterative procedure by showing that it reaches a fixpoint for any tp-program P at ω and that this fixpoint satisfies P.

Theorem 51 *1. $h_f \models P$* **iff** *$T_P(h_f) \leq h_f$*

2. *If $lfp(T_P)$ is fully defined then $lfp(T_P) \models P$.*

3. *If $lfp(T_P)$ is fully defined, then $T_P^\omega = lfp(T_P)$. Consequently, if $lfp(T_P)$ is fully defined then $T_P^\omega \models F : \mu$ iff $P \models F : \mu$.*

Proof

1. $h_f \models P$ **iff** $T_P(h_f) \leq h_f$.

   - $h_f \models P \Longrightarrow T_P(h_f) \leq h_f$.

     Let $h_f \models P$. We need to show that $T_P(h_f) \leq h_f$. By Lemma 17 it is sufficient to show that $T_P(f) \leq f$, i.e., $(\forall A \in B_L)(\forall t \in \tau)(T_P(f)(A,t) \supseteq f(A,t))$.

     Let us consider an arbitrary fixed timepoint $t \in \tau$. By the definition of the T_P operator, $T_P(f)(A,t) = T_P(h_f)(A,t) = \cap L$ where $L = \{[l', u'] | A : [c, l, u, \delta] \longleftarrow Body \in ground(P); t \in sol(c); h_f \models Body; [l', u'] = [\delta(c,t) \cdot l, \delta(c,t) \cdot u]\}$.

     For reference, let us denote by $\mathcal{C}_{T_P}$ the set $\{A : [c, l, u, \delta] \longleftarrow Body \in ground(P) | t \in sol(c); h_f \models Body; \}$.

     For any clause $C = A : [c, l, u, \delta] \longleftarrow Body \in \mathcal{C}_{T_P}$ $h_f \models C$ (since h_f is a model of P it satisfies every program clause) and also $h_f \models Body$. By definition of satisfaction for formula tp-functions (Definition 14.1.11) $h_f \models A : [c, l, u, \delta]$. Therefore, $h_f(A,t) = f(A,t) \subseteq [\delta(c,t) \cdot l, \delta(c,t) \cdot u]$. Since $[\delta(c,t) \cdot l, \delta(c,t) \cdot u] \in L$ we now know that $(\forall [l', u'] \in L) f(A,t) \subseteq [l', u']$. But this means that $f(A,t) \subseteq \cap_{[l',u'] \in L}[l', u'] = T_P(f)(A,t)$.

     From the last inequality and Lemma 17 it follows that

     $T_P(f) \leq f$, i.e. $T_P(h_f) \leq h_f$.

- $h_f \models P \Longleftarrow T_P(h_f) \leq h_f$.

  Now, suppose we know that $T_P(h_f) \leq h_f$. We have to show that $h_f \models P$.

  Consider some atom A and timepoint t. We know that $f(A,t) \subseteq T_P(f)(A,t) = T_P(h_f)(A,t)$. As we recall, $T_P(f)(A,t) = \cap L$ where $L = \{[l',u'] | A : [c,l,u,\delta] \longleftarrow Body \in ground(P); t \in sol(c); h_f \models Body; [l',u'] = [\delta(c,t) \cdot l, \delta(c,t) \cdot u]\}$. Again let $\mathcal{C}_{T_P}$ denote the set $\{A : [c,l,u,\delta] \longleftarrow Body \in ground(P) | t \in sol(c); h_f \models Body; \}$.

  We know that $f(A,t) \subseteq T_P(h_f)(A,t) = \cap_{[l',u'] \in L}[l',u']$. Consider some clause $C = A : [c,l,u,\delta] \longleftarrow Body \in ground(P)$, such that $h_f \models Body$. From above, we know that $(\forall t \in sol(c))(h_f(A,t) \subseteq [\delta(c,t) \cdot l, \delta(c,t) \cdot u])$. This means that $h_f \models A : [c,l,u,\delta]$ and therefore, $h_f \models C$.

  Thus we have shown that for any program clause for which h_f satisfies the body, it also satisfies the head. But this means that h_f satisfies all program clauses (the remaining clauses will be vacuously satisfied) and therefore $h_f \models P$.

2. If $lfp(T_P)$ is fully defined then $lfp(T_P) \models P$.

   This is an almost direct corollary of the previous part of the theorem. By definition of the least fixpoint of an operator $T_P(lfp(T_P)) = lfp(T_P)$, i.e., $T_P(lfp(T_P)) \leq lfp(T_P)$ if $lfp(T_P)$ is fully defined. But then, by the last inequality, $lfp(T_P) \models P$.

3. If T_P^ω is fully defined, then $T_P^\omega = lfp(T_P)$.

   Assume T_P^ω is fully defined. To show that $T_P^\omega = lfp(T_P)$ we show that $T_P(T_P^\omega) = T_P^\omega$. Since T_P is monotonic, $T_P^\omega \leq T_P(T_P^\omega)$. We now show that $T_P(T_P^\omega) \leq T_P^\omega$.

   By definition, given an atom A and a timepoint t, $T_P^\omega(A,t) = \bigcap_{i \leq \omega} T_P^i(A,t)$. Then,

$T_P(T_P^\omega)(A,t) = \cap L$ where $L = \{[l', u'] | A : [c, l, u, \delta] \longleftarrow Body \in ground(P);$ $T_P^\omega \models Body; t \in sol(c); [l', u'] = [\delta(c,t) \cdot l, \delta(c,t) \cdot u]\}$.

Let us denote by $\mathcal{C}_{T_P}$ the set $\{A : [c, l, u, \delta] \longleftarrow Body \in ground(P) | T_P^\omega \models Body; t \in sol(c); [l', u'] = [\delta(c,t) \cdot l, \delta(c,t) \cdot u]\}$.

Consider a clause $C = A : [c, l, u, \delta] \longleftarrow Body \in \mathcal{C}_{T_P}$. As $T_P^\omega \models Body$, i.e. $\cap_{i \leq \omega} T_P^i \models Body$, and since $T_P^0, T_P^1, \ldots$ is an ascending sequence of formula tp-functions, there has to be some k, such that $T_P^k \models Body$. Let us consider the minimal such k and denote is by k_C. Notice that such k_C exists for each $C \in \mathcal{C}_{T_P}$.

Now let $k_A = \max_{C \in \mathcal{C}_{T_P}}(k_C)$. We know that $T_P^{k_A} \models Body$ for all clauses $C = A : [c, l, u, \delta] \longleftarrow Body \in \mathcal{C}_{T_P}$, which means that $T_P^\omega(A,t) = T_P^{k_A}(A,t)$. But then

$T_P(T_P^\omega)(A,t) = T_P(T_P^{k_A})(A,t) = T_P^{k_A+1}(A,t) \subseteq T_P^\omega(A,t)$. From this and Lemma 17 we get $T_P(T_P^\omega) \leq T_P^\omega$, which, combined with monotonicity of T_P gives us $T_P^\omega = T_P(T_P^\omega)$, i.e., T_P^ω is a fixpoint of T_P.

$T_P(T_P^\omega) \leq T_P^\omega$ also means that $T_P^\omega \models P$. What is left to show is that T_P^ω is the **least** fixpoint of T_P.

Let h_f be a formula tp-function and $T_P(h_f) = h_f$. Immediately, by part 2 of this theorem, $h_f \models P$. Also $T_P^0 \leq h_f$, as $T_P^0 = \perp$.

Let $T_P^i \leq h_f$. Since T_P is monotonic, $T_P(T_P^i) = T_P^{i+1} \leq T_P(h_f) = h_f$. Now we know that for all natural i, $T_P^i \leq h_f$. But then $T_P^\omega = \sqcup_i T_P^i \leq h_f$, from which it follows that T_P^ω is the least fixpoint of T_P. $\square$

14.1.4 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.

First, we briefly recall the specifics of unification in annotated logic programming languages.

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 [82] and it was proved that though most general unifiers are not necessarily unique, a corresponding notion of a *maximally* general unifier (max-gu) exists. Section 5.1 contains more information on it, and in what follows we will use the notions described there.

The key concept used in the proof procedure is introduced in the next definition.

Definition 14.1.14 (Explosions) *Let* $F : [c, l, u, \delta]$ *be a basic formula. The* explosion *of* $F : [c, l, u, \delta]$*, denoted* $\mathcal{E}(F : [c, l, u, \delta])$ *is defined as* $\{F : [t, \delta(c, t) \cdot l, \delta(c, t) \cdot u, \sharp] | t \in sol(c)\}$*. The* exploded basic formula *for* $F : [c, l, u, \delta]$*, denoted* $\mathcal{E}_{\mathcal{F}}(F : [c, l, u, \delta])$*, is* $\bigwedge_{t \in sol(c)} F : [t, \delta(c, t) \cdot l, \delta(c, t) \cdot u, \sharp]$.
Let $r = F : [c, l, u, \delta] \longleftarrow Body$ *be a tp-clause. Then the* explosion *of* r*, denoted* $\mathcal{E}(r)$ *is defined as* $\{F : [t, \delta(c, t) \cdot l, \delta(c, t) \cdot u, \sharp] \longleftarrow Body | t \in sol(c)\}$.
If P *is a tp-program, then the* explosion *of* P*, denoted* $\mathcal{E}(P)$ *is defined as* $\{\mathcal{E}(r) | r \in P\}$.
$\square$

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 14.1.15 (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 $(F \wedge G) : [t, \max(0, l + l' - 1), \min(u, u'), \sharp] \longleftarrow Body \wedge Body'$[5] 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$^i(P)$.*

□

---

[5]Here and in $\vee$-composition rule that follows, we assume that when F and G are combined to form the head of the new rule, their free variables are standardized apart. Similarly, free variables in $Body$ and $Body'$ are also standardized apart.

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

Lemma 19 *For every clause $r \in$ CL(P), $P \models r$.*

Proof. Let C be a clause from CL(P). We prove $P \models C$ by induction on the number of iterations it took add C to CL(P).

- **Base Case.** $C \in$ CL$^0(P)$.

  Two cases are possible.

  1. $C \in \mathcal{E}(P)$.

     In this case C has the form $F : [t, l, u, \sharp] \longleftarrow Body$ and there exists a tp-clause C' of the form $F : [c', l', u', \delta'] \longleftarrow Body$, where $t \in sol(c)$, $[l, u] = [l' \cdot \delta'(c', t), u' \cdot \delta'(c', t)]$ and $C \in \mathcal{E}(C')$.

     Since $C' \in P$, $P \models C'$. Let h_f be a tp-function such that $h_f \models P$ (and hence $h_f \models C'$) and $h_f \models Body$. In this case, $h_f \models F : [c', l', u', \delta']$, i.e, for all ground substitutions Θ of F, $(\forall t' \in sol(c))(h_f(F\Theta, t') \subseteq [l' \cdot \delta'(c', t'), u' \cdot \delta'(c', t')])$.

     From here, it follows that since $t \in sol(c')$, it is the case that $h_f(F\Theta, t) \subseteq [l' \cdot \delta'(c', t), u' \cdot \delta'(c', t)] = [l, u]$. But then, $h_f \models (F : [t, l, u, \sharp])\Theta$, for all ground substitutions Θ of F, which means that $h_f \models F : [t, l, u, \sharp]$, $h_f \models C$ and therefore $P \models C$.

  2. C is of the form $A : [t, 0, 1, \sharp] \longleftarrow$, $A \in B_L$. Let $h_f \models P$. Clearly, $h_f(A, t) \subseteq [0, 1]$ and therefore $h_f \models A : [t, 0, 1, \sharp] \longleftarrow$, yielding $P \models A : [t, 0, 1, \sharp] \longleftarrow$.

- **Induction Step.** Let $P \models C'$ for all tp-clauses $C' \in$ CL$^j(P)$. We need to show that $P \models C$ for all tp-clauses $C \in$ CL$^{j+1}(P)$.

  Let $C \in$ CL$^{j+1}(P)$. Four cases are possible.

1. $C \in \mathrm{CL}^j(P)$. In this case, by induction hypothesis, $P \models C$.

2. $C \notin \mathrm{CL}^j(P)$. C was added to $\mathrm{CL}^{j+1}(P)$ via the Clarification rule.

   This means that C has a form $(F : [t, \max(l, l'), \min(u, u'), \sharp] \longleftarrow Body \wedge Body')\Theta$, and there are two clauses C_1 and C_2 in $\mathrm{CL}^j(P)$, such that C_1 is of the form $F : [t, l, u, \sharp] \longleftarrow Body$; C_2 is of the form $F' : [t, l', u', \sharp] \longleftarrow Body$; F and F' are unifiable via max-gu Θ.

   By induction hypothesis, $P \models C_1$ and $P \models C_2$. Let h_f be a formula tp-function which is a model of P and let $h_f \models Body$ and $h_f \models Body'$.

   Since $h_f \models Body$, $h_f \models Body\Theta$ and also since $h_f \models C_1$, $h_f \models F : [t, l, u, \sharp]$ and $h_f \models (F : [t, l, u, \sharp])\Theta$.

   Similarly, $h_f \models Body'\Theta$ and $h_f \models (F : [t, l', u', \sharp])\Theta$.

   From here we conclude that $h_f \models (Body \wedge Body')\Theta$. Also, $h_f \models (F : [t, l, u, \sharp])\Theta$ means that for all Θ', such that $F\Theta\Theta'$ is ground $h_f(F\Theta\Theta', t) \subseteq [l, u]$ and similarly, $h_f \models (F' : [t, l', u', \sharp])\Theta$ means that for all Θ'', such that $F'\Theta\Theta''$ is ground $h_f(F'\Theta\Theta'', t) \subseteq [l', u']$. Since $F\Theta = F'\Theta$, their set of ground extensions coincide, and therefore for any ground extension $F\Theta\Theta'$ of $F\Theta$, we get $h_f(F\Theta\Theta') \subseteq [l, u]$ and $h_f(F\Theta\Theta') \subseteq [l', u']$, yielding $h_f(F\Theta\Theta') \subseteq [l, u] \cap [l', u'] = [\max(l, l'), \min(u, u')]$.

   The latter means that $h_f \models (F : [t, \max(l, l'), \min(u, u'), \sharp])\Theta$ and therefore $h_f \models (F : [t, \max(l, l'), \min(u, u'), \sharp] \longleftarrow Body \wedge Body')\Theta$. Therefore $h_f \models C$ as desired.

3. $C \notin \mathrm{CL}^j(P)$. C was added to $\mathrm{CL}^{j+1}(P)$ via the $\wedge$-composition rule.

   This means that C has a form $F \wedge G : [t, \max(0, l + l' - 1), \min(u, u'), \sharp] \longleftarrow Body \wedge Body'$, and there are two clauses C_1 and C_2 in $\mathrm{CL}^j(P)$, such that C_1 is of the form $F : [t, l, u, \sharp] \longleftarrow Body$; C_2 is of the form $G : [t, l', u', \sharp] \longleftarrow$

$Body$.

By induction hypothesis, $P \models C_1$ and $P \models C_2$. Let h_f be a formula tp-function which is a model of P and let $h_f \models Body$ and $h_f \models Body'$ (hence $h_f \models Body \wedge Body'$).

Since $h_f \models Body$, for any ground substitution of F, $h_f \models Body\Theta$ and also since $h_f \models C_1$, $h_f \models F : [t, l, u, \sharp]$ and $h_f \models (F : [t, l, u, \sharp])\Theta$.

Similarly, $h_f \models Body'\Theta'$ and $h_f \models (G : [t, l', u', \sharp])\Theta'$ for any ground substitution.

Since Θ is a ground substitution for F and Θ' is a ground substitution for G, $\Theta\Theta'$ is a ground substitution for $F \wedge G$. From the above and by the definition of h_f, $h_f(F\Theta, t) \subseteq [l, u]$ and $h_f(G\Theta', t) \subseteq [l', u']$. Then $h_f((F \wedge G)\Theta\Theta', t) \subseteq [l, u] \otimes [l'u'] = [\max(0, l + l' - 1), \min(u, u')]$. Therefore $h_f \models (F \wedge G) : [t, \max(0, l + l' - 1), \min(u, u'), \sharp]$ and therefore $h_f \models C$, which means that $P \models C$.

4. $C \notin$ CL$^j(P)$. C was added to CL$^{j+1}(P)$ via the $\vee$-composition rule. The proof is similar to the one for $\wedge$-composition case.

   C has a form $F \vee G : [t, \max(l, l'), \min(u + u', 1), \sharp] \longleftarrow Body \wedge Body'$, and there are two clauses C_1 and C_2 in CL$^j(P)$, such that C_1 is of the form $F : [t, l, u, \sharp] \longleftarrow Body$; C_2 is of the form $G : [t, l', u', \sharp] \longleftarrow Body$.

   By induction hypothesis, $P \models C_1$ and $P \models C_2$. Let h_f be a formula tp-function which is a model of P and let $h_f \models Body$ and $h_f \models Body'$ (hence $h_f \models Body \wedge Body'$).

   Since $h_f \models Body$, for any ground substitution of F, $h_f \models Body\Theta$ and also since $h_f \models C_1$, $h_f \models F : [t, l, u, \sharp]$ and $h_f \models (F : [t, l, u, \sharp])\Theta$.

   Similarly, $h_f \models Body'\Theta'$ and $h_f \models (G : [t, l', u', \sharp])\Theta'$ for any ground substitution.

Since Θ is a ground substitution for F and Θ' is a ground substitution for G, $\Theta\Theta'$ is a ground substitution for $F \wedge G$. From the above and by the definition of h_f, $h_f(F\Theta, t) \subseteq [l, u]$ and $h_f(G\Theta', t) \subseteq [l', u']$. Then $h_f((F \vee G)\Theta\Theta', t) \subseteq [l, u] \otimes [l'u'] = [\max(l, l'), \min(u + u', 1)]$. Therefore $h_f \models (F \vee G) : [t, \max(l, l), \min(u+u', 1), \sharp]$ and therefore $h_f \models C$, which means that $P \models C$. $\square$

Notice that the syntax of CL(P) is somewhat different from the syntax of P, since we allow the rules in CL(P) to have non-atomic heads. However this extension is supported by our definitions of satisfaction on formula tp-functions (Definition 14.1.11). We are now ready to proceed with the resolution procedure. First we define tp-queries.

Definition 14.1.16 *A* tp-query *is an expression of the form* $\exists(H_1 : [c_1, l_1, u_1, \delta_1] \wedge \ldots \wedge H_n : [c_n, l_n, u_n, \delta_n)])$*, where* $c_1, \ldots, c_n$ *are temporal constraints and* $[l_i, u_i] \subseteq [0, 1]$ *for all* $1 \leq i \leq n$ *and* $\delta_1, \ldots, \delta_n$ *are probability distributions functions.*
The explosion of a tp-query Q*, denoted* $\mathcal{E}_{\mathcal{F}}(Q)$ *is an expression of the form:*

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

$\square$

A tp-query consists of of a conjunction of queries of the form "is it true that the probability that event F occurs some time in $sol(c)$ with probability in interval $[l, u]$ distributed according to δ?" The explosion of a tp-query is a conjunction of the explosions of all basic formulas in the query.

Intuitively, tp-resolution will proceed as follows. Given a tp-query Q, its explosion will be constructed. Each conjunct in the explosion will then then treated as an independent subquery. Since each conjunct in the explosion is annotated by information about its probability at one particular timepoint, the resolution search procedure will proceed using the closure CL(P) of the tp-program P. At each step, an appropriate rule from CL(P) will be selected and the explosion of its body will replace the subquery in the explosion of the query. When the current query expression becomes empty, the resolution succeeds. Formally, we define tp-resolution as follows.

Definition 14.1.17 *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 14.1.18 *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 $\langle Q_1, r_1, \Theta_1\rangle \ldots \langle Q_s, r_s, \Theta_s \langle \ldots$ 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 $\langle Q_1, r_1, \Theta_1\rangle \ldots \langle Q_s, r_s, \Theta_s\rangle$ 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.*

If Θ_is are not restricted to be max-gu's, we call the tp-refutation unrestricted. □

We are now ready to state the soundness and completeness results for tp-resolution procedure. The proposition below will be useful in our proof of soundness.

Proposition 18 $h_f \models F : [c, l, u, \delta]$ *iff* $h_f \models \mathcal{E}_{\mathcal{F}}(F : [c, l, u, \delta])$.

Proof. First we note that $h_f \models \mathcal{E}_{\mathcal{F}}(F : [c, l, u, \delta])$ is equivalent (according to Definition 14.1.14 of Explosions) to $h_f \models F : [t, l', u', \sharp]$ for all $F : [t, l', u', \sharp] \in \mathcal{E}(F : [c, l, u, \delta])$.

- $h_f \models F : [c, l, u, \delta] \Longrightarrow h_f \models \mathcal{E}_{\mathcal{F}}(F : [c, l, u, \delta])$.

  Let $F : [t, l', u', \sharp] \in \mathcal{E}(F : [c, l, u, \delta])$. We show that $h_f \models F : [t, l', u', \sharp]$.

  By Definition 14.1.14 $[l', u'] = [\delta(c, t) \cdot l, \delta(c, t) \cdot u]$. $h_f \models F : [c, l, u, \delta]$ i.e., $h_f(F, t'') \subseteq [\delta(c, t''), \cdot l, \delta(c, t'') \cdot u]$ for all $t'' \in sol(c)$. Since $t \in sol(c)$ (by Definition 14.1.14), we get $h_f(F, t) \subseteq [\delta(c, t), \cdot l, \delta(c, t) \cdot u]$, i.e., $h_f(F, t) \subseteq [l', u']$. From this, $h_f \models F : [t, l', u', \sharp]$ as desired.

- $h_f \models F : [c, l, u, \delta] \Longleftarrow h_f \models \mathcal{E}_{\mathcal{F}}(F : [c, l, u, \delta])$.

  We know that $h_f \models F : [t, l', u', \sharp]$ for all $F : [t, l', u', \sharp] \in \mathcal{E}(F : [c, l, t, \delta])$. Therefore, by Definition 14.1.14 of Explosions, $(\forall t \in sol(c))(h_f(F, t) \subseteq [l', u'])$. Also, by the same definition, $[l', u'] = [\delta(c, t) \cdot l, \delta(c, t) \cdot u]$, i.e., $(\forall t \in sol(c))(h_f(F, t) \subseteq [[\delta(c, t) \cdot l, \delta(c, t) \cdot u]$. Then, by definition of satisfaction formulas by formula tp-functions, $h_f \models F : [c, l, u, \delta]$. $\square$

Theorem 52 *[Soundness of tp-refutation]*

Let P be a tp-program, and Q be an initial query. If there exists a tp-refutation of $Q \equiv \exists(F_1 : [c_1, l_1, u_1, \delta_1] \wedge \ldots \wedge F_n : [c_n, l_n, u_n, \delta_n])$ from P with the answer substitution Θ then $P \models \forall((F_1 : [c_1, l_1, u_1, \delta_1] \wedge \ldots \wedge F_n : [c_n, l_n, u_n, \delta_n])\Theta)$.

Proof. Let $\langle Q_1, C_1, \Theta_1 \rangle \ldots \langle Q_n, C_n, \Theta_n \rangle$ be a tp-refutation of Q from P. The proof of soundness proceeds by induction on n.

- **Base Case.** $n = 1$.

  In this case $Q_1 = \mathcal{E}_{\mathcal{F}}(Q) = F : [t, l, u, \sharp]$ and in fact $Q = Q_1$ (since every step involves exactly one conjunct of the explosion of a query, the refutation of length

1 means that the explosion of the query has a single conjunct, equivalent to the query itself).

Also, $C_1 \equiv G : [t, l_1, u_1, \sharp] \longleftarrow \in$ CL(P) and $F\Theta_1 = G\Theta_1$, and $[l_1, u_1] \subseteq [l, u]$.

Now, let $h_f \models P$. Since $C_1 \in$ CL(P), by Lemma 19 $h_f \models C_1$, therefore, $h_f \models G : [t, l_1, u_1, \sharp]$. This means that $h_f \models (G : [t, l_1, u_1, \sharp])\Theta_1$. Therefore, for any ground substitution γ, $h_f(G\Theta_1\gamma, t) = h_f(F\Theta_1\gamma) \subseteq [l_1, u_1] \subseteq [l, u]$. But then, $h_f \models (F : [t, l, u, \sharp])\Theta_1$.

Since $Q = Q_1 = F : [t, l, u, \sharp]$ and $\Theta = \Theta_1$, $h_f \models Q\Theta$ and therefore $P \models Q\Theta$.

- **Induction Step.** Let the theorem hold for any tp-refutation $\langle Q_2, C_2, \Theta_2\rangle \ldots \langle Q_n, C_n, \Theta_n\rangle$ of size $n - 1$. We show that the statement of the theorem holds for all tp-refutations of size n.

  Consider a tp-refutation $\langle Q_1, C_1, \Theta_1\rangle\langle Q_2, C_2, \Theta_2\rangle \ldots \langle Q_n, C_n, \Theta_n\rangle$.

  Let $h_f \models P$ and let $Q_1 = F_1 : [t_1, l_1, u_1, \sharp] \wedge \ldots \wedge F_k : [t_k, l_k, u_k, \sharp]$.

  Let C_1 be a (renamed version of a) clause in CL(P): $C_1 \equiv G : [t, l, u, \sharp] \longleftarrow Body$ and let for some $1 \leq i \leq k$, $F_i\Theta_1 = G\Theta_1$, $[l, u] \subseteq [l_i, u_i]$ and $t = t_i$. Then, $Q_2 = (F_1 : [t_1, l_1, u_1, \sharp] \wedge \ldots \wedge F_{i-1} : [t_{i-1}, l_{i-1}, u_{i-1}, \sharp] \wedge \mathcal{E}_{\mathcal{F}}(Body) \wedge F_{i+1} : [t_{i+1}, l_{i+1}, u_{i+1}, \sharp] \wedge \ldots \wedge F_k : [t_k, l_k, u_k, \sharp])\Theta_1$.

  By induction hypothesis, $h_f \models Q_2\Theta_2 \ldots \Theta_n$.

  Therefore, $h_f \models \mathcal{E}_{\mathcal{F}}(Body)\Theta_2 \ldots \Theta_n$. By Proposition 18, $h_f \models Body\Theta_2 \ldots \Theta_n$. Since $C_1 \in$ CL(P) $h_f \models C_1$. Combined together, the last two expressions yield $h_f \models (G : [t, l, u, \sharp])\Theta_1\Theta_2 \ldots \Theta_n$, and as $G\Theta_1 = F_i\Theta_1$ and $[l, u] \subseteq [l_i, u_i]$ we get $h_f \models (F_i : [t, l_i, u_i])\Theta_1\Theta_2 \ldots \Theta_n$.

  From $h_f \models Q_2\Theta_2 \ldots \Theta_n$ we infer

$$
\begin{aligned}
h_f \models \quad & (F_1 : [t_1, l_1, u_1, \sharp] \wedge \ldots \wedge F_{i-1} : [t_{i-1}, l_{i-1}, u_{i-1}, \sharp] \\
& \wedge F_{i+1} : [t_{i+1}, l_{i+1}, u_{i+1}, \sharp] \wedge \ldots \wedge F_k : [t_k, l_k, u_k, \sharp])\Theta_1\Theta_2 \ldots \Theta_n.
\end{aligned}
$$

Combining together the last two expressions we obtain the desired $h_f \models (F_1 : [t_1, l_1, u_1, \sharp] \wedge \ldots \wedge F_k : [t_k, l_k, u_k, \sharp])\Theta_1 \ldots \Theta_n$, i.e., $h_f \models Q_1\Theta_1 \ldots \Theta_n$. By Proposition 18, as $Q_1 = \mathcal{E}_{\mathcal{F}}(Q)$ $h_f \models Q\Theta_1 \ldots \Theta_n$.

This completes the soundness proof. □

In order to prove a completeness theorem we have to establish first a number of facts just as was the case with HPLP framework (Section 5.2). We adopt the Max-gu lemma and the Lifting Lemma for tp-refutations from [69].

Lemma 20 *(***Max-gu Lemma***). Let Q be a query that has an unrestricted tp-refutation from a tp-program P. Then, Q has an tp-refutation of the same length and if $\Theta_1, \ldots, \Theta_m$ are the unifiers form the unrestricted hp-refutation, and $\Theta'_1, \ldots, \Theta'_m$ are the max-gu's from the hp-refutation, then, for some γ $\Theta_1 \ldots \Theta_m = \Theta'_1, \ldots, \Theta'_m\gamma$.*

Lemma 21 *(***Lifting Lemma***) Let P be a tp-program, Q be a query, Θ be a substitution. Let $Q\Theta$ have an tp-refutation from P. Then Q has a tp-refutation from P of the same length. Also, if $\Theta_1, \ldots, \Theta_m$ are the max-gu's from the refutation of $Q\Theta$ and $\Theta'_1, \ldots, \Theta'_m$ are the max-gu's from the refutation of Q then, for some substitution γ: $\Theta\Theta_1 \ldots \Theta_m = \Theta'_1, \ldots, \Theta'_m\gamma$.*

We can now prove the completeness theorem for tp-refutations.

Theorem 53 *[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.

We break the proof of this theorem into a number of steps. First we show that it is sufficient to prove the statement of Theorem 53 for the case when Q is of the form $F : [t, l, u, \sharp]$.

Proposition 19 *Let $F : [c, l, u, \delta]$ and $G : [c', l', u', \delta']$ be two ground basic tp-formulas which have tp-refutations from tp-program P.*
Then so does $F : [c, l, u, \delta] \wedge G : [c', l', u', \delta']$.

Proof. Let $\langle F : [c, l, u, \delta], C_1, \Theta_1 \rangle \ldots \langle Q_l^F, C_l, \Theta_l \rangle$ be the tp-refutation for $F : [c, l, u, \delta]$. Let $\langle G : [c', l', u', \delta], D_1, \Gamma_1 \rangle \ldots \langle Q_k^G, D_k \Gamma_k \rangle$ be the tp-refutation for $G : [c', l', u', \delta']$. Then, as $F : [c, l, u, \delta]$ and $G : [c, l, u, \delta]$ are ground, the following will be a tp-refutation for $F : [c, l, u, \delta] \wedge G : [c', l', u', \delta']$:

$$\langle F : [c, l, u, \delta] \wedge G : [c', l', u', \delta'], C_1, \Theta_1 \rangle, \langle Q_2^F \wedge G : [c', l', u', \delta'], C_2, \Theta_2 \rangle \ldots$$

$$\langle Q_l^F \wedge G : [c', l', u', \delta'], C_l, \Theta_l \rangle, \langle G : [c', l', u', \delta'], D_1, \Gamma_1 \rangle \ldots \langle Q_k^G, D_k \Gamma_k \rangle$$

$\square$

Combining the results of Proposition 19 and Proposition 18 we get that in order to prove the completeness result, we have to prove the following lemma.

Lemma 22 *Let P be a tp-program and let $P \models \exists(F' : [t, l, u, \sharp])$. Then, there exists a tp-refutation of $F' : [t, l, u, \sharp]$ from P.*

Proof. $P \models \exists(F' : [t, l, u, \sharp])$ implies that there exists a ground substitution Θ such that $P \models (F' : [t, l, u, \sharp])\Theta$. Let $F = F'\Theta$. By Lifting Lemma, to prove this lemma we need to show that there exists a tp-refutation of $F : [t, l, u, \sharp]$ from P.

$P \models F : [t, l, u, \sharp]$ implies that $lfp(T_P) \models F : [t, l, u, \sharp]$ or, since $lfp(T_P) = T_P^\omega$, $T_P^\omega \models F : [t, l, u, \sharp]$, i.e., $T_P^\omega(F, t) \subseteq [l, u]$.

By definition of T_P^ω, there exists an ordinal $k \le \omega$ such that $T_P^{k-1}(F,t) \not\subseteq [l,u]$ and $T_P^k(F,t) \subseteq [l,u]$.

We prove that $F : [t,l,u,\sharp]$ has a tp-refutation using induction on k.

Base Case. $k = 0$.

$T_P^0(F,t) = [0,1] \subseteq [l,u]$, therefore $[l,u] = [0,1]$. Let $F = A_1 * \ldots * A_n$, where $* \in \{\wedge, \vee\}$. We notice that $[0,1] \otimes [0,1] = [0,1] \oplus [0,1] = [0,1]$. For each atom A_i $\mathrm{CL}^0(P)$ contains a tp-clause $C_i^* \equiv A_i : [t,0,1,\sharp] \longleftarrow$. Then, $\mathrm{CL}^n(P)$ will contain the clause $C^* \equiv A_1 * \ldots A_n : [t,0,1,\sharp] \longleftarrow$ obtained from C_i^* by recreated applications of either $\wedge$-composition or $\vee$-composition rule (depending on whether $*$ connective is $\wedge$ or $\vee$).

Then, $\langle F : [t,0,1,\sharp], C^*, e\rangle$, where e is an empty substitution is a tp-refutation of $F : [t,0,1,\sharp]$.

Induction Step. Assume that for all formulas $G : [t',l',u',\sharp]$, $T_P^{k-1}(G,t') \subseteq [l',u']$ implies existence of a tp-refutation of $G : [t',l',u',\sharp]$ from P.

Let $T_P^{k-1}(F,t) \not\subseteq [l,u]$, but $T_P^k(F,t) \subseteq [l,u]$. We show that $F : [t,l,u,\sharp]$ has a tp-refutation.

We proceed by induction on the structure of F.

- **Base Case** F is an atom: $F = A \in B_L$.

  $T_P^k(A,t) = T_P(T_P^{k-1})(A,t) = \cap L$, where $L = \{[l',u'] | A : [c,l'',u'',\delta] \longleftarrow Body \in ground(P); T_P^{k-1} \models Body; t \in solc; [l',u'] = [\delta(c,t)\cdot l'', \delta(c,t)\cdot u''].$

  Let $\mathcal{C}_L = \{A : [c,l'',u'',\delta] \longleftarrow Body \in ground(P); T_P^{k-1} \models Body; t \in sol(c); \}$, and let $\mathcal{C}_L = \{C_1^*, \ldots C_r^*\}$, where

  $$C_i^* \equiv A : [c_i, l_i'', u_i'', \delta_i] \longleftarrow Body_i.$$

  Now, for each C_i^*, let $\bar{C}_i \in P$ be such a rule that $C_i^* \in ground(\bar{C}_i)$, and let Θ_i be such a substitution that $C_i^* = \bar{C}_i\Theta_i$. We will represent $\bar{C}_i$ as

$$\bar{C}_i \equiv A_i' : [c_i, l_i'', u_i'', \delta_i] \longleftarrow Body_i',$$

where $A = A_i'\Theta_i$ and $Body_i = Body_i'\Theta_i$.

Since all $\bar{C}_i$ $(1 \leq i \leq r)$ are in P, CL$^0(P)$ will contain the following r clauses:

$$C_i' \equiv A_i' : [t, l_i', u_i', \sharp] \longleftarrow Body_i',$$

for $1 \leq i \leq r$, where $[l_i', u_i'] = [\delta_i(c_i, t) \cdot l_i'', \delta_i(c_i, t) \cdot u_i'']$.

Let $\Theta_1', \ldots \Theta_r'$ be such substitutions that for all $1 \leq i \leq r$, $C_i^* = C_i'\Theta_i'$.

Then, CL$^r(P)$(actually, CL$^{\log_2 r}(P)$) will contain the following clause:

$$\hat{C} \equiv (A_1' : [t, \max(l_1', \ldots l_r'), \min(u_1', \ldots u_r'), \sharp] \longleftarrow Body_1' \wedge \ldots \wedge Body_r')\hat{\Theta},$$

where $T\hat{heta}$ is the max-gu of $A_1', \ldots A_r'$ (such a max-gu exists because we know that all A_i' are unifiable with A).

Let Θ^* be such substitution that $A = A_1\hat{\Theta}\Theta^*$.

We know that for all $1 \leq i \leq r$, $T_P^{k-1} \models Body_i$. Therefore, by induction hypothesis and Proposition 19 we know that there exists a tp-refutation for $Body_i \wedge \ldots \wedge Body_i$. Let

$$\langle \mathcal{E}_{\mathcal{F}}(Body_i \wedge \ldots \wedge Body_i), C_1'', \Theta_1'' \rangle \ldots \langle Q_s, C_s'', \Theta_s'' \rangle$$

be this tp-refutation. But then,

$$\langle F : [t, l, u, \sharp], \hat{C}, \Theta' \rangle \langle \mathcal{E}_{\mathcal{F}}(Body_i \wedge \ldots \wedge Body_i), C_1'', \Theta_1'' \rangle \ldots \langle Q_s, C_s'', \Theta_s'' \rangle$$

is a tp-refutation of $F : [t, l, u, \sharp]$ from P.

- *Induction Step.* Assume that all formulas $G \equiv A_1 * \ldots A_m$ ($* \in \{\wedge, \vee\}$), such that $T_P^k \models G : [t', l', u', fat1]$ have a tp-refutation from P, for $m < n$. We need to consider two cases.

  Case 1. $F = A_1 \wedge \ldots \wedge A_n$, $T_P^k \models F : [t, l, u, \sharp]$.

  Let $G = A_1 \wedge \ldots \wedge A_{n-1}$ and $H = A_n$[6]. The $F \equiv G \wedge H$ and $T_P^k \models (G \wedge H) : [t, l, u, \sharp]$, i.e. $T_P^k(G \wedge H, t) = T_P^k(G, t) \otimes T_P^k(H, t) \subseteq [l, u]$. Since T_P operator is monotonic, we know that $T_P^\omega(G, t) \subseteq T_P^k(G, t)$ and $T_P^\omega(H, t) \subseteq T_P^k(H, t)$.

  Let $T_P^k(G, t) = [l_1, u_1]$ and $T_P^k(H, t) = [l_2, u_2]$. From above, $T_P^\omega \models G : [t, l_1, u_1, \sharp]$ and $T_P^\omega \models H : [t, l_2, u_2, \sharp]$, i.e., $P \models G : [t, l_1, u_1, \sharp]$ and $P \models H : [t, l_2, u_2, \sharp]$.

  By induction hypothesis, both $G : [t, l_1, u_1, \sharp]$ and $H : [t, l_2, u_2, \sharp]$ have tp-refutations. Let these refutations respectively be

$$\langle G : [t, l_1, u_1, \sharp], C_1^G, \Theta_1^G \rangle \ldots \langle Q_s^G, C_s^G, \Theta_s^G \rangle$$

  and

$$\langle H : [t, l_2, u_2, \sharp], C_1^H, \Theta_1^H \rangle \ldots \langle Q_s^H, C_s^H, \Theta_s^H \rangle.$$

  Let us consider the clauses C_1^G and C_1^H. According to the definition of tp-refutations these clauses are contained in CL(P) and should be of the following form:

[6] This is one of the possible divisions of F into two non-empty parts. Any such division would work for the purpose of our proof. The reason for that is the fact that the for any formula tp-function, the probability intervals associated by it with complex formulas are uniquely determined by the probability intervals associated with the atoms that compose these formulas. This is due to the fact that $\oplus$ and $\otimes$ operations used to compute the probability intervals for disjunctions and conjunctions are both associative and commutative functions as shown in Chapter 3.

$$C_1^G \equiv G' : [t, l_1', u_1', \sharp] \longleftarrow Body_1'$$

$$C_1^H \equiv H' : [t, l_2', u_2', \sharp] \longleftarrow Body_2',$$

where G' and G are unifiable via max-gu Θ_1^G, H' and H are unifiable via max-gu Θ_1^H, $[l_1', u_1'] \subseteq [l_1, u_1]$ and$[l_2', u_2'] \subseteq [l_2, u_2]$.

Since C_1^G is in CL(P), there exists some iteration i_G on which it was added to CL$^{i_G}(P)$. Similarly, since C_1^H is in CL(P), it was added on some iteration i_H to CL$^{i_H}(P)$. Let $w = \max(i_G, i_H)$. Then CL$^{w+1}(P)$ will contain the following clause (added by the rule of $\wedge$-composition):

$$C^* \equiv G' \wedge H' : [t, \max(l_1' + l_2' - 1, 0), \min(u_1', u_2'), \sharp] \longleftarrow Body_1' \wedge Body_2'.$$

We know that $[l, u] \subseteq [l_1, u_1] \otimes [l_2, u_2]$. We also know that $[l_1, u_1] \subseteq [l_1', u_1']$ and $[l_2, u_2] \subseteq [l_2', u_2']$. Since $\otimes$ is monotonic, $[l, u] \subseteq [l_1', u_1'] \otimes [l_2', u_2'] = [\max(l_1' + l_2' - 1, 0), \min(u_1', u_2')]$. Also, since $F = G \wedge H$, F is unifiable with $G' \wedge H'$ via max-gu $\Theta_1^G \Theta_1^H$.

Finally, we know that $Q_2^G = \mathcal{E}_{\mathcal{F}}(Body_1' \Theta_1^G)$ and $Q_2^H = \mathcal{E}_{\mathcal{F}}(Body_2' \Theta_1^H)$.

From all of the above we conclude that

$$\langle (G \wedge H) : [t, l, u, \sharp], C^*, \Theta_1^G \Theta_1^H \rangle \langle (\mathcal{E}_{\mathcal{F}}(Body_1') \wedge \mathcal{E}_{\mathcal{F}}(Body_2')) \Theta_1^G \Theta_1^H, C_2^G, \Theta_2^G \rangle \ldots$$

$$\langle Q_S^G \wedge \mathcal{E}_{\mathcal{F}}(Body_2') \Theta_1^H, C_s^G, \Theta_s^G \rangle^7 \langle \mathcal{E}_{\mathcal{F}}(Body_2') \Theta_1^H, C_2^H, \Theta_2^H \rangle \ldots \langle Q_r^H, C_r^H, \Theta_r^H \rangle$$

will be a tp-refutation of $F : [t, l, u, \sharp]$ from P.

Case 2. $F = A_1 \vee \ldots \vee A_n$, $T_P^k \models F : [t, l, u, \sharp]$.

[7] We use the fact that $Body_1'$ and $Body_2'$ are standardized apart to discard the applications of substitutions.

In this case, the proof is the same as in Case 1, when $\wedge$ is replaces everywhere with $\vee$ and $\otimes$ operator and its results are replaced with $\oplus$ operator and its results.

This concludes the induction steps of both levels of double inductions and proves the lemma. $\square$

This completes the proof of completeness of tp-refutations.

In all fairness we should note that tp-refutation procedure is inefficient. The explosion operation, which lies at its core, is applied to the entire program and entire query, regardless of any specific aspects of either which may allow for optimization. Drawing a parallel with TP-databases, using tp-refutation in query processing is similar to annotating a TP-database and running TATA queries on it, in place of using TPA queries on the original TP-relations. Tp-refutations ignore the fact that certain information is presented compactly in a tp-program and fails to capitalize on that even in a simple case when a tp-program consists of a single tp-clause $F : [c, l, u, \delta] \longleftarrow$ and the incoming query is $F : [c, l, u, \delta]$.

At this stage it is clear that in order for tp-programs to be a feasible framework for reasoning about temporal probabilistic information, better proof procedures have to be specified. At present we are working on one such procedure, which instead of creating an explosion for the program and the query, will attempt to match "pieces" of the query to "pieces" of tp-clauses from a tp-program (e.g. if a clause contains information about timepoints 4 – 9 and the query asks about timepoints 5 – 12, this procedure will attempt to resolve the overlapping part of the two intervals, 5 – 9, in one step rather than breaking off the refutation for each individual timepoint).

Another reason why tp-refutations are inefficient is due to the fact that query processing requires an enormous pre-computation stage during which the closure of a tp-program is computed. This is extremely expensive operation. In the HPLP framework

(Section 5.3), we have described in detail how one can create a dynamic proof procedure based on the system of inference rules to overcome this problem. Supplying a dynamic analog of tp-refutations, which would not resort to constructing closures of tp-programs is more or less straightforward: the rules which guide the creation of the closure are substituted with inference rules which allow dynamic derivation of only the necessary clauses. However, since tp-refutations are inefficient for more than just this reason, it seems that a better solution would be to develop a framework that deals with the other shortcomings of tp-refutations first, and then come up with its dynamic analog.

In this respect, tp-refutations, being shown to be sound and complete w.r.t. the semantics given above, provide a good measuring stick for other proof procedures. As was the case with the HPLP framework, proving soundness and completeness of a proof procedure against the semantics is a fairly complicated process. On the other hand, proving that one proof procedure is a conservative extension of another is a much simpler task (as seen in the proofs of soundness and completeness of HR_P-refutations vs. hp-refutations). Similarly, we expect that it will be easier to show that a new proof procedure for tp-programs is a conservative extension of tp-refutations, than to prove its soundness and completeness from scratch.

Part IV

Conclusions

Chapter 15

Related Work

In this section we summarize the related work.

15.1 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 [32]. Shapiro was one of the first to develop results in fuzzy logic programming [96]. Baldwin [6] was one of the first to introduce *evidential* logic programming and a language called FRIL. Van Emden [107] was one of the first to provide formal semantical foundations for logic programs that was later extended by Subrahmanian [100] and then completely generalized in a succession of papers by Blair and Subrahmanian[10], and Fitting[40], Ginsberg [42], and applied to databases by Kifer and Li [55] and Kifer and Subrahmanian [56]. 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 [82, 83, 84], developed techniques for probabilistic logic programming (PLP) under the assumption of ignorance (see Section 2.2 for more detailed description of this work). Their work built upon earlier work on probabilistic logics due to Fagin, Halpern and and others [36, 39, 47, 1], Hailperin [46] and Nilsson [85].

Hybrid Probabilistic Logic Programs described here have been inspired in part by the work of Ng and Subrahmanian on Probabilistic Logic Programs. The syntax of hp-programs is derived from the syntax of PLPs and the semantics has a lot in common. In fact, an hp-program which uses only ignorance probabilistic disjunctive and conjunctive strategies is a valid PLP in the framework of Ng and Subrahmanian.

Besides being a semantical extension of PLPs, our work also complements that of [82, 83, 84] in scope. While Ng and Subrahmanian concentrated on the expressivity aspects of their framework and on extending it to a non-monotonic case ([84] provides stable semantics for PLPs), we have concentrated on finding reasonable proof procedures and on algorithms and complexity analysis for the problems of entailment and consistency.

Kiessling and his group [45, 103, 54, 95] 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. In many respects this work is similar to Bayesian Nets [86] in the importance which independence and conditional independence play in their system. The inference rules used to derive new information employ probabilistic intervals. However, their system is not tailored for reasoning under assumptions about the relationship between the events other from the independence assumption.

There has been significant body of work on Bayesian inference in Logic Programming, in part due to the similarity between the problem of abduction in Logic Programming [1] and the problem of computing posterior probability in Bayesian inference. Poole

[1]Given a logic program P and a set of facts O known to be true the problem of abduction is finding the

in his work on Probabilistic Horn Abduction [88, 89] presented an elegant system of probabilistic abduction.

In Poole's framework, the logic program consists of two parts. One is a standard set of Prolog rules. The second part contains the descriptions of probability distributions over different atoms of the system. Syntactically it is expressed as $disjoint([h_1 : p_1, h_2 : p_3, \ldots, h_k : p_k])$ where $p_1 + \ldots + p_k = 1$ and p_i is the probability that h_i is true. Implicit is the assumption that h_i and h_j are mutually exclusive whenever $i \neq j$.

Poole shows how Bayesian Nets ([86]) can be expressed in his framework. The process of computing posterior probabilities of atoms, which plays the role of abduction is also described.

A number of differences exist between this work and ours. We have not yet studied abduction for hp-programs, but this can be a rather interesting problem. On the other hand, independence assumption plays a key role in Poole's framework where it is assumed that all hypotheses are independent. In fact, in [88], Poole states that he specifically tailored the design of his framework so that it would not be possible to express any dependency between the hypothesis even in syntax. While later, he notes that since his system allows for representation of any Bayesian Net and hence dependencies can in fact be represented via the introduction of additional hypotheses to account for them, he admits that in general this can lead to exponential blowup in the number of atoms in the program (and hence, in the exponential blowup in the size of the program itself). "We are not claiming that we are getting something for nothing" concludes Poole his discussion of dependencies among hypotheses in his system in [88].

Ngo and Haddawy in a series of papers [80, 81] developed the frameworks for conducting Bayesian inference in Logic Programming settings. Similarly to Poole, their

possible explanation(s) of O from P.

framework has integrity constraints which define which atoms are mutually exclusive and form a probability distribution. Unlike Poole, however, the probabilities are specified not with the integrity constraints but in the rule base which contains rules expressing unconditional and conditional probabilities present in the system. Such a system naturally captures the semantics of Bayesian Nets, and in fact, [80] shows that their fixpoint and model theories coincide with it. However, it also means that the approach of Ngo and Haddawy, similarly to that of Poole's [88, 89] is well suited only for situations when the atoms in the programs are independent.

While, the claim of [80] to improve upon the original results of Ng and Subrahmanian [82] is true (apart from the fact that Ngo and Haddawy use point probabilities while PLPs of [82] work with intervals), the addition of variable annotations to PLPs [83] allows the Probabilistic Logic Programs to express Bayesian inference as well. Hp-programs, being a conservative extension of PLPs, inherit this feature as well.

Lukasiewicz [71, 74, 72, 73, 75] has recently studied a number of probabilistic logic programming paradigms ranging from the analogs of Bayesian Nets [71] to the adaptations of multi-valued logics to provide probabilistic semantics for nonmonotonic logic programs [72, 73]. Unlike Poole [88] and Ngo and Haddawy [80, 81], Lukasiewicz in all his work on Probabilistic Logic Programming is using interval probabilities, which makes his work more similar to that of Lakshmanan's group [66, 65] (see below) and both the work of Ng and Subrahmanian [82, 83, 84] and the HPLP framework described in the thesis. In fact, Probabilistic Logic Programs described in [71] were in part inspired by the desire to overcome the complexity of the possible worlds semantics of PLPs of Ng and Subrahmanian [82, 83]. This however restricted the scope of the framework to that of representing conditional probabilities and conditional reasoning.

In [74, 72], Lukasiewicz presents a probabilistic semantics for logic programs which is based on multi-valued logic ([71] is devoted to monotonic logic programs, disjunctive logic programs are considered in [72]). The idea behind the approach is to treat the domain of probabilistic values as a lattice of truth values. This approach allows Lukasiewicz to derive a number of nice results about his framework. However, extending multi-valued logic to probabilities as done in [74, 72] has its own drawbacks. In particular, in this framework, the probability of a conjunction of two events is computed as the meet of the probabilities of the events, which on the lattice described in [74, 72] is the minimum of the two probabilities, i.e. $Pr(F \wedge H) = \min(Pr(F), Pr(H))$. Similarly, the probability of disjunction is computed as join of the probabilities, which happens to be the maximum of the two: $Pr(F \vee H) = \max(Pr(F), Pr(H))$.

As we recall from Chapter 3 the above formulas for computing the probability of complex event given probabilities of simple events are applied when the simple events are positively correlated. Therefore, the applicability of the multi-valued probabilistic semantics is restricted to the situations when every pair of events under consideration is positively correlated. Unfortunately, this is a very unlikely scenario in most real world applications.

Perhaps the most significant related work is the elegant work of Lakshmanan's group [66, 65, 67], which have been developing a parametric framework to represent varied probabilistic strategies in logic programs [67].

In Probabilistic Deductive Databases [66] each rule of a deductive database has two intervals $\langle [a, b], [c, d] \rangle$ associated with it. The first interval represents the certainty or confidence that the head of the rule is true given the truth of the body of the rule. The second interval represents the uncertainty or doubt in the implication defined by the rule. The first interval is interpreted as the subjective probability of the head of the rule given that the body of the rule is true.

As in our work on Hybrid Probabilistic Logic Programs, Lakshmanan and Sadri introduce different relationships (modes in their terminology) between events in Probabilistic Deductive Databases. They consider four different modes (independence, ignorance, positive correlation, minimal overlap/negative correlation). Probabilistic strategies, considered in our work allow for these and other relationships to exist in HPLP framework.

Syntactically, hp-programs and Probabilistic DBs of Lakshmanan and Sadri [66] are quite different. A probabilistic rule in [66] is defined as

$$(A \stackrel{\alpha}{\longleftarrow} B_1, \ldots B_n; \mu_r, \mu_p),$$

where $\alpha = \langle [a, b], [c, d] \rangle$ is the confidence level of the rule as described above, μ_r is the mode for combining together confidence information within the rule and μ_p is the mode for combining together the information about the head of the rule defined from different derivations. The probabilistic information present in the rules refers to the probability of the implication defined by the rule. In hp-programs the probability intervals are always associated with basic formulas and the notion of $\longleftarrow$ in the rules is that of reversed modus ponens, as is the case with classical Logic Programming.

A probabilistic rule

$$(A \stackrel{\langle [a,b],[c,d] \rangle}{\longleftarrow} B_1, \ldots B_n; \mu_r, \mu_p),$$

of Lakshmanan and Sadri's [66] framework can be translated (ignoring the doubt interval) into an hp-clause

$$A : c_{\mu_r}([a, b], [V, W]) \longleftarrow (B_1 \wedge_{\mu_r} \ldots \wedge_{\mu_r} B_n) : [V, W]^2$$

[2] In the semantics of Lakshmanan and Sadri [66], the probabilities of atoms $B_1, \ldots, B_n$ get combined under the assumption (using the mode) μ_r, exactly the same way our framework finds the probability of

One important semantical distinction between hp-programs and probabilistic deductive databases is in how the results of derivations get combined. [66] assumes that probabilistic intervals for the same formula A obtained from different rules (derivations) can be combined using any of the different modes (such as ignorance, independence, positive or negative correlation), represented in the rule by μ_p.

While, this creates a rather flexible framework, we find this approach somewhat counterintuitive as independence, positive correlation and other modes describe relationships between different events. Therefore, if one knows that probability interval of A is α and probability interval of B is β, one can combine these intervals using the formulas for probability under independence, positive correlation etc... However, knowing two different estimates α and β for the probability interval of one event A does not mean that the real probability interval for this event can be obtained by combining α and β using the same assumption, because these assumptions do not apply when two different probability intervals of the same event are considered.

In the HPLP framework different probability intervals for the same event are combined using intersection, a decision that followed Ng and Subrahmanian [82]. The reasoning behind this is quite simple: given probabilistic interval α from source $s1$ and probabilistic interval β from source $s2$ for formula F , $\alpha \cap \beta$ is the maximal interval on which both sources $s1$ and $s2$ agree. If the probability of F is in the intersection, then both sources were correct.

This approach means that an hp-program that consists of two clauses $A : [0, 0] \longleftarrow$ and $A : [1, 1] \longleftarrow$ is inconsistent as the probability interval associated with A by this

---

$B_1 \wedge_{\mu_r} \ldots \wedge_{\mu_r} B_n$. Furthermore, the probability of the conjunctions of $B_i's$ gets combined with the probability of the implication in their rule using the same assumption μ_r. The latter is reflected in the annotation of the head in our translation of the rule.

program is $[0, 0] \cap [1, 1] = \emptyset$. This interpretation is in line with our general intuition about the probability.

The analog of this program in the framework of Lakshmanan and Sadri is not necessarily inconsistent. Depending on the choice of the mode which combines together the probabilities of the same event, the probability interval associated with A may be $\emptyset$ (for intersection) or $[0, 1]$ (for the approach that does a logical "or" of the two statements of the program, i.e. states "The probability of A is either $[0, 0]$ or $[1, 1]$".

Another important difference between HPLPs and Probabilistic DDBs is the fact that only atomic heads are allowed in the latter framework. While this disadvantage can be compensated via the introduction of new atoms in the framework to represent complex events, the straightforward translation of hp-programs into Probabilistic DDBs becomes quite counterintuitive. Also, the framework does not allow one to use the knowledge of the probability of a complex event to find the probability interval for a simple event which is a part of it, which is one of the key features of hp-programs.

All in all, the two frameworks, while different in syntax and semantics do share a considerable intersection in expressive power. However, neither framework really subsumes 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 contrast, our work offers a variety of cache-based query processing algorithms that complement their query optimization work, and which we expect will lead to new view-based query optimization methods.

Fuhr and his group have also worked in the area of probabilistic logic programming [38]. Most of the work of his group is related to Information Retrieval, hence Probabilistic Datalog developed in [38] was intended as the language for expressing search queries. The independence assumption played a key role in Probabilistic Datalog – individual atoms are assumed to be conditionally independent following the traditional Information Retrieval assumption used by most probabilistic methods that the distributions of different terms in the documents of the collections are conditionally independent. While arguably a very simplified view (e.g., occurrences of terms "Turing Machine" and "computability" in Computer Science literature are hardly independent), this assumption has been proven to lead to fairly simple, fast and reliable retrieval methods [19, 104]. But the design of Probabilistic Datalog requires one to carry with each atom the full path by which it had been derived in order to avoid incorrect probabilistic computations. Unlike the language of Poole, which does not allow the user to incorporate a dependency between two atoms in a program, it is possible to do so in Fuhr's Probabilistic Datalog. Without path information, Probabilistic Datalog would always compute probability under the assumption of independence, which would make computation incorrect in case a dependency was somehow specified in the program. Path information allows Fuhr to correct the computation of probability without the need to explicitly introduce different dependencies between atoms, although, arguably this is a fairly inconvenient if not clumsy attempt to overcome the problems that arise when only independence assumption is present. Another difference between Probabilistic Datalog and hp-programs is the use of point probabilities in the former, as well as (traditional) lack of support for computing probabilities of simple events from the probabilities of complex events.

15.2 Probabilistic Databases

Most work on uncertainty in databases can be separated into two categories: qualitative uncertainty, such as managing incomplete information, and uncertainty expressed as probability. Some of the work on qualitative uncertainty in databases is described below in Sections 15.3.2 and 15.3.3. In Section 15.4.2 we will describe the work on representing uncertainty in databases with fuzzy measures which does not fall in either of the categories above.

Among the quantitative approaches to dealing with uncertainty in databases, the ones rooted in probability theory seem to be prevailing. This, to a certain degree, stands in contrast with Logic Programming, where multiple paradigms of uncertain reasoning based on Fuzzy Logic, multivalued logics and lattices have been developed prior to probabilistic approaches. Quite possibly, this can be explained by the fact it was relatively hard to establish non-probabilistic measures of uncertainty on "real life" data with a high degree of credibility. On the other hand, probability as a statistical measure has been studied for a long time and found its place in data analysis a long time ago.

While the Temporal Probabilistic Database work described in this thesis is not a generic probabilistic database framework, in large part its development had been inspired by both the strengths and shortcomings of such frameworks developed in the past decade. In Sections 2.4 and 2.3 we have already described some of the features of probabilistic database frameworks. Here, we recall some of the main points from Section 2.4 and draw parallels with our work to show the scope of the influence of these works on our framework. We devote the next section to comparison of our work with work on uncertainty in Temporal Databases.

The pioneering work in the theory of probabilistic databases is due to Cavallo and Pittarelli [15]. The relational database model proposed by them would associate a probability distribution with every relation in the database. The relational operations defined on such relations are projection and join. In addition, Cavallo and Pittarelli study the information content of a probabilistic relation as well as probabilistic data dependencies.

Most of the work that followed [15], however, used a completely different underlying data model. The main shortcoming of [15] was that any probabilistic relation was in reality one uncertain tuple. The restriction that the probabilities associated with tuples in a probabilistic relation form a proper probability distribution (i.e. add up exactly to 1) really meant that a probabilistic relation was nothing more than a description of one discrete random variable and the probability associated with each tuple was the probability of this tuple being the value of the underlying random variable. Unfortunately, this was not a convenient extension of a classical relational data model. Traditionally, a tuple in a relation is viewed as describing a certain statement or event and tuples from one relation describe multiple events or statements that are similar in structure. In contrast, in the probabilistic model of Cavallo and Pittarelli, every relation describes uncertainty about exactly one statement/event (discrete random variable).

A more desirable way to extend the classical relational model to handle probabilistic information seems to be to assume that each tuple in the probabilistic relational model represents a discrete random variable. This way, a probabilistic relation will contain information about many statements/events of similar structure, only now, there would be uncertainty about the exact nature of these events/statements captured with the use of probabilities. It is not surprising that in the 1990s, the work on probabilistic extensions of database theory moved in that direction.

The influential work of Barbara, Garcia-Molina and Porter [7] highlighted both the advantages and shortcomings of the abovementioned approach and served as an excellent starting point for a number of future works.

In the data model of [7], it is assumed that tuples have certain primary key attributes, which define the identity of each tuple and that some of the non-key attributes may be stochastic, meaning that the values of such attributes are subject to discrete probability distributions. The data model also allowed for a joint probability distribution to be specified for the values of more than one attribute. Barbara et. al. defined projection, selection and join operations on their probabilistic relations. The framework also allowed for the information stored in stochastic attributes to be incomplete: a value "*" would have to be added to the set of possible values of the stochastic attribute with appropriate probability value accounting for missing information.

While restrictive at places, the framework developed in [7] showed how probabilistic information in databases should be treated. The data model proposed by Barbara et. al. is easy-to-understand and provides an efficient way to store probabilistic information. However, some aspects of the relational algebra left room for improvement. Allowing missing data in the relations turned out to be a potential cause of problems as the relational algebra operations on such data have been shown to be lossy, i.e. not necessarily leading to the correct probability distributions in the resulting relations. Some other issues that arise naturally when dealing with probabilistic information have been avoided in [7] due to restrictions put forth on the data model and operations. For example, projection can project out only non-key attributes of probabilistic relations meaning that probabilistic tuples never lose their identity. This and the lack of set-theoretical operations allowed the authors to avoid addressing the important problem of combining together information about the same event/statement originating from different sources, known as coalescence or compaction of information. Another serious restriction of the framework was the use of point probabilities and the assumption of independence of all probability distributions in the tuple stemming from it.

An important paper on the topic is by Dey and Sarkar [28] who propose an elegant `1NF` approach to handling probabilistic databases. Their paper is a significant improvement on the work of Barbara et al. [7]. 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;

(iv) removes assumptions about deterministic keys used in [7];

(v) defines first complete probabilistic relational algebra;

(vi) recognizes the problem of combining probabilistic data from different sources and proposes coalescence operation to deal with it.

The `1NF` representation used by them is basically an annotated representation of the probabilistic relations in Barbara, Garcia-Molina, Porter framework [7] (similar to the annotated relations in our TP-Database framework).

Many of our operators, such as union, intersection and compaction are influenced by similar operations in their relational algebra despite the fact that the semantics of some of these operations is somewhat different.

The framework of Dey and Sarkar, however has a number of shortcomings. The main problem with their relational algebra is that for certain operations to produce correct probabilities some extra assumptions about the dependencies between the information contained in the database are made. And in some cases, such as the difference operation, which we have already described in Section 2.4, it is hard to associate a probabilistic meaning with the result of their operation.

Combined together, the work of Barbara et al. [7] and Dey and Sarkar [22] suggests how a probabilistic database framework should be organized and what features it should have.

As in [7], storing probabilistic relations in a "compact" form, by associating with each tuple one discrete random variable (uncertain event/statement) has an advantage of providing a logical and intuitive data organization. Using the 1-NF form in the tradition of Dey and Sarkar [22] should allow one to express the meaning of the complex "compact" probabilistic relations in a format which is close to classical relational data model. Relational algebra on 1-NF "annotated" probabilistic relations will be very similar to the classical relational algebra with "special" treatment awarded only to the field(s) containing the probability. It will also be fairly transparent. Using this relational algebra as denotational semantics, one can then define the relational algebra on compact probabilistic relations.

The features to be improved upon w.r.t. [7] and [22] include

- using interval probabilities instead of point probabilities;
- making sure that results of all operations of relational algebra provide correct probabilities for events/statements in their results
- treating incomplete information in a manner not leading to lossy operations.

Perhaps this roadmap for defining probabilistic database framework is the most important influence of [7] and [22] on our work. Lakshmanan et. al [64] came close to following through on the agenda above in defining their ProbView framework, stopping short of providing a relational algebra for their compact data model. In our work on Temporal Probabilistic Databases we have followed this path to the end, providing data models for both compact and annotated temporal probabilistic relations, defining relational algebras on both data models and proving that they coincide in semantics.

As mentioned multiple times throughout the thesis, our work in Temporal Probabilistic Databases builds on top of the ProbView system for probabilistic databases[64]. Prob-

View 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 ProbView has been given earlier in Section 2.3. As mentioned above, the main improvement of Temporal Probabilistic Databases over ProbView is methodological: we have provided relational algebra for our compact data model, TP-relations. Not being able to operate directly on compact relations made ProbView vulnerable to situations when domains of attribute values could be large, as ProbView would have to physically store one tuple for each possible combination of values.

At this point, to our knowledge, the problem of defining full relational algebra on compact probabilistic relations in generic case is still open and would be an interesting problem to study.

15.3 Temporal Uncertainty in Databases

15.3.1 Comparison with Dyreson and Snodgrass

Dyreson and Snodgrass [34, 35] 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. In fact, Dyreson and Snodgrass were the first to consider the use of probability distribution functions rather than explicit probability values in [34].

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 [34, 35] 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 an improvement over the framework of Dyreson and Snodgrass in the manner described below.

- **Methodology.** Our approach was to extend/improve/modify probabilistic databases to handle uncertain temporal information successfully and efficiently. Dyreson and Snodgrass extend temporal databases with features for dealing with probabilities. An advantage of our approach is that it can be naturally extended to handle other specific kinds of uncertainty, such as spatial and spatio-temporal uncertainty.

- **SQL vs. Algebra.** Dyreson and Snodgrass present a version of SQL for temporally indeterminate databases (extending well-known TSQL language [109]). In contrast, we present a relational algebra and prove that all our algebraic operations are correct.

- **Base Relations.** In the Dyreson and Snodgrass [35] framework, the base relations are temporal relations. In effect, base relations in [35] 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 [35] 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 [35] which we do not handle — for example, in TP Databases, we have assumed tuples have only one indeterminate temporal attribute while [35] allows more than one. Also, we have no analog of correlation credibility or ordering plausibility.

- **Distribution Functions.** In [35], all PDFs are assumed to be complete, meaning that the framework of [35] does not allow for missing/incomplete information. In contrast, in TP-Databases, we allow both complete and incomplete PDFs.

- **Independence Assumptions.** In [35], 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 [64] tradition of working with events whose relationship is parameterized by probabilistic strategies.

- **Relational Algebra Operations.** Our algebra supports a host of operations that do not appear to be supported in the Dyreson-Snodgrass framework [35]. 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 efficient means to check consistency.

15.3.2 Other Work In Temporal Databases

Snodgrass was one of the first to model indeterminate instances in his doctoral dissertation [99] — he proposed the use of a model based on three valued logic. Dutta [33] 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 [31]. The use of fuzzy logic in Logic Programming and Databases is summarized in Section 15.4.2 below.

There has been a large body of work on qualitative uncertainty in temporal databases. This work concentrates on representing incomplete information and qualitative relations between different statements/events represented in the databases (e.g., stating that event e_1 occurred after event e_2).

Gadia [41] proposes an elegant model to handle incomplete temporal information. He models events that are completely known, events that are unknown but are known to have occurred, events that are known if they have occurred, and events that are unknown even if they occurred. Gadia [41] shows that his model is sound. However, he makes no use of probabilistic information, in fact he uses no quantitative methods at all.

Some frameworks for dealing with uncertainty in temporal (and not only) databases rely heavily on research in constraint databases.

15.3.3 Constraint Databases

While constraint databases by themselves are designed to contain *certain* information, the importance of the use of constraints in temporal databases cannot be understated. Constraints can also be helpful in describing some qualitative uncertainty.

Stemming from the work of Jaffar and Lassez [51] and Kanellakis, Kuper and Revesz [52, 53] constraint logic programming and databases (see also [93, 76]) have emerged as important fields of research in the past decade.

In a classical relation R which has the schema $(A_1, \ldots, A_n)$ over domain $D_1 \times \ldots \times D_n$ a tuple $(a_1, \ldots, a_n)$ can be viewed as a collection of equalities : $\{A_1 = a_1, \ldots, A_n = a_n\}$. Following this idea, Kannellakis et al. introduce a notion of a *generalized tuple* over database schema, as a quantifier-free conjunction of constraints on the variables representing field names of the database schema.

For example, for database schema (x, y, z) where all three variables range over reals, the following are generalized tuples: $x + y < z$, $x < y \wedge x \cdot y \leq z^2$, $x = 1 \wedge x < 5 \wedge z = x + y - 1$, etc.

Intuitively, a classical relational tuple $(a_1, \ldots, a_n)$ represents a single point in the space $D_1 \times \ldots \times D_n$. On the other hand, a generalized tuple can potentially represent multiple points from the domain of the relation. In fact, as the examples of generalized tuples above indicate, it is possible to represent infinitely many points of the domain by one generalized tuple. Thus, *constraint relations*, which are defined as collections of generalized tuples over the same domain and *constraint databases*, defined as collections of constraint relations can serve as finite representations of infinite relations.

The key problems of the theory of constraint databases are the design and analysis of algorithms for efficient processing of queries to constraint databases and analysis of different query and constraint languages for their expressive power and computational complexity. These questions are out of the scope of review here.

Constraints have found extensive use in temporal databases for replacing a timestamp in a temporal relation with a temporal constraint specifying an interval during which the information contained in the tuple was true. This allows compact storage of temporal information in the database. In fact, anytime we talk about compact representation of the data of some sort, implicit with this is the assumption that such compactness is achieved by replacing single values in specific attributes with constraints that describe more than one value.

Our TP-Database framework actively employs temporal constraints to represent information about the possible times when an event could have occurred and about the domains of probability distribution functions associated with TP-tuples.

But constraints also have found use in helping define frameworks for dealing with certain kinds of uncertainty. In particular, applications of constraints to uncertainty capitalize on the fact that constraints can be used to represent relationships between different attributes of the tuple, and, with the right syntax, relationships between the attributes of different tuples.

An important work in this area is that of Koubarakis [59, 60] who proposes the use of constraints for representing temporal data. Like our work on TP-Databases, Koubarakis uses constraints to represent when an event occurs. In the framework proposed by him, each tuple in a temporal relation is timestamped with a unique identifier. In addition to timestamp attribute, each tuple contains a constraint attribute which can involve timestamp identifiers from any tuples of the relations. Thus, Koubarakis' framework allows stating the facts that event e_1 occurred between 8 and 11 AM, and that event e_2 occurred after 12pm, as well as explicitly stating that e_2 occurred after e_1 in the constraint field of either of the tuples.

This approach stands in contrast with the TP-Database framework. The constraint language of Koubarakis is richer than the temporal constraint language employed by us. In fact, we specifically chose a simple constraint language to make our framework efficiently implementable.

Another important body of work is that of Brusoni *et al.*[14] 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 [26], it is not a part of this thesis.

15.4 Other Reasoning With Uncertainty Paradigms

15.4.1 Bayesian Nets

Bayesian Nets [86] are the most popular AI paradigm for reasoning with probabilistic information.

Given a set U of random variables with known domain (or domains), a Bayesian Net represents the conditional dependencies between different random variables. A Bayesian Net is a directed graph whose nodes represent variables from U. A directed arc from variable x_i to variable x_j means that variable x_j is conditionally dependent on x_i. Each node of such a graph is augmented with a probability distribution: a root node (i.e. node with no ancestors) x is labeled with a probability distribution $P(x)$, while an internal node y, all of whose ancestors are $z_1, \ldots, z_k$ is labeled with a conditional probability distribution $P(y|z_1, \ldots z_n)$. The probability distributions labeling the nodes of a Bayesian Net represent the *a priori* (prior) probabilities existing in the system.

One of the advantages of a Bayesian Net is that it can be used both to compute prior probabilities, by propagating probability distributions from the roots all the way down to the nodes requested and to compute *a posteriori* (posterior) probabilities, when the values of some of the random variables in the net become observed. The latter is done by changing the probability distributions on the observed nodes to reflect the observations, and by propagating the probabilities up and down the net to create new probability distributions induced by the observations.

As mentioned in Section 15.1, a number of people have proposed Logic Programming paradigms closely tied to Bayesian Nets and Bayesian reasoning. These include the works of Kiessling and his group [45, 103, 54, 95], Poole [88, 89] and Ngo and Haddawy [80, 81].

Ng and Subrahmanian in [83] show how the use of variable annotations in their language of Probabilistic Logic Programs allows them to express conditional probabilities and Bayesian updates. Given atoms A and B, expressing the fact that the probability of A given B lies in the interval $[\alpha, \beta]$ can be done with a clause

$$(A \wedge B) : [\alpha \cdot V_1, \beta \cdot V_2] \longleftarrow B : [V_1, V_2].$$

Furthermore, the application of Bayes rule of conditional probability

$$Prob(B|A) = \frac{Prob(A|B) \cdot Prob(B)}{Prob(A)}$$

(assuming $Prob(A) \neq 0$) can be represented with a clause

$$(BA) : [V_1 \cdot V_2/V_3, V_1 \cdot V_2/V_3] \longleftarrow (AB) : [V_1, V_1] \wedge B : [V_2, V_2] \wedge A : [V_3, V_3]$$

where AB stands for the event $A|B$.

Since our Hybrid Probabilistic Logic Programming framework is a strict extension of the work of Ng and Subrahmanian, it inherits the ability to represent conditionalization and Bayesian updates from [83] (the $(A \wedge B)$ conjunction will become $(A \wedge_{igc} B)$).

Some fairly obvious differences (like the use of point probabilities in Bayesian Nets vs. intervals in HPLP framework) aside, there are some deeper differences between the two frameworks.

The inference in Bayesian Nets proceeds by propagating the probability computation from one node, representing "simple" random variable to another. A Bayesian Net presents a nice graphical representation of the independence of one set of random variables on another (unconditional or given a third set of random variables). However, computing a probability of a conjunction of two random variables given a particular relationship between them, and hypothesizing using this information is not done in Bayesian Nets. Similarly Bayesian Nets are not usually used to estimate the probability of simple events given the probability of their conjunction or disjunction.

So, in a sense Bayesian Nets and HPLP framework represent somewhat different approaches to reasoning about probabilities. Bayesian Nets represent the known dependencies between random variables (events) in the system and require probability distributions and conditional probability distributions to be known. A typical query to a Bayesian Net is "What is the probability of X given that Y and Z occurred?" where X, Y and Z represent simple events.

On the other hand, hp-clauses represent deductive knowledge a user may have about his/her domain. This knowledge is about probability intervals of different events but it need not be Bayesian. The user may know/assume that if the probability of event F falls in interval μ, then the probability of another event G will be in interval μ' and hp-programs allow the user to express this knowledge without questioning its source. A typical query to an hp-program P is "Is it true that the probability of event F lies in the interval $[a, b]$?" (an alternative query may ask for the tightest probability interval for F derivable from P). Here, F is an arbitrary event, possibly a conjunction or disjunction of two or more simple events under some assumption about their relationship. The latter stands in contrast with Bayesian Nets approach which only takes into account the knowledge about the probabilities of simple events (and conditional probabilities) when performing computations.

Unlike the works of Poole [88] and Ngo and Haddawy [80], which are closely tied to Bayesian Nets and have specific syntax to specify that certain ground atoms form a probability distribution, HPLP framework provides no special means for doing so. However, it is possible to express the conditions that have to hold for a group of ground atoms to form a probability distribution: hp-clauses can express that disjunction of all atoms has a probability interval $[1, 1]$, while all pairwise conjunctions have probability intervals $[0, 0]$ associated with them.

15.4.2 Fuzzy Set Theory and Fuzzy Logic

Fuzzy Set Theory and Fuzzy Logic [108] have been a permanent feature in different studies of uncertain reasoning and uncertain data management since early 80s.

The main advantage of Fuzzy Logic when applied to reasoning with uncertainty and management of uncertain data is relative simplicity of the mathematics involved when compared to the use of probability theory. On the other side, however, is the issue of the intuition behind the fuzzy measures. While the notion of a probability of an event has clear meaning even for a non-mathematician, due to the long history of the field and connections to statistics and gambling, the notion of a fuzzy measure of an event or statement is fairly new, and not intuitively connected with any particular meaning outside of the narrow circuit of mathematicians and engineers. This issue seems to be the main obstacle to the use of fuzzy logic based methods to deal with real life data, as when the user receives an answer "Event e has a fuzzy measure of 0.78", (s)he may have problems interpreting it.

However, fuzzy logic has found successful use in a number of applied areas from signal processing to business decision making, and therefore, whenever the dataset to be dealt with accepts fuzzy measures of uncertainty and people dealing with the data are familiar with the concepts, fuzzy logic may be applied. So, while the application of fuzzy logic to problems is not universal (just as is the case for probabilistic methods), there are some important domains where the information is treated as being fuzzy.

Prade has championed the use of Fuzzy Logic in Databases and Logic Programming since the early/mid- 1980s [90, 91]. In 1989 Dubois and Prade [31] proposed the extensions of Temporal Database framework to manage uncertain data using Fuzzy Logic. In this latter work, Dubois and Prade model ill-known dates by associating with each date identifier a a fuzzy function π_a from timepoints of the calendar to interval $[0, 1]$. $\pi_a(t) = 0$ is interpreted as "we are certain that a is not equal to t" and $\pi_a(t) = 1$ is interpreted as "it is completely possible that $a = t$". The latter interpretation means that there may be more than one timepoint t for which $\pi_a(t) = 1$. For such fuzzy dates Dubois and Prade construct a theory that allows one to compare dates, rank dates in ascending order and perform a host of other operations on them. All these operations yield fuzzy (possibilistic) functions.

Apart from the use of probabilities vs. possibilities, the main difference between our TP-Database framework and the framework of [31] is in the scope. Dubois and Prade in [31] concentrate exclusively on different relationships between dates in the tradition of the seminal work of Allen [3] in the area of Temporal Databases. TP-Database framework concentrates on creating the data model for efficiently storing temporal probabilistic information and relational algebra for efficiently querying it.

15.4.3 Temporal Reasoning

Over the past 25 years there has been a lot of interest in temporal reasoning in various areas of Computer Science (see, e.g., the work of Pnueli on Temporal Logic for program verification [87]).

In Artificial Intelligence, problems related to reasoning about change have existed for over 30 years. In 1969, McCarthy and Hayes [77] introduced the situation calculus approach to reasoning about. Their attempts to use it were met with the famous *frame problem* In the early 1980s, McCarthy [78] and Reiter [92] introduced a number of formalisms (circumscription, default logic) for non-monotonic reasoning.

Implicit in this work was the assumption that the reasoning of an agent takes place in time. In the framework of situation calculus, the agent (in order to "defeat" the frame problem) needs to be able to conclude that unless explicitly affected by actions, facts persist over time. In [48], Hanks and McDermott argue that traditional problems of non-monotonic reasoning are closely connected with reasoning in time.

While time has been "virtually" present in AI research ever since the situation calculus [77], it became the object of research in AI only in the 1980s with the developments in non-monotonic reasoning and temporal logics. On one hand, research in temporal logics lead to Temporal Logic Programming [2, 8] as the means of expressing information and reasoning about temporal dependencies between different events. On the other hand, the development of research in planning demanded explicit reasoning about time.

Lin and Dean in [68] identified the problem of temporal reasoning as follows:

> "Given a set of events, a set of ordering constraints on the events, and a description of the cause-an-effect relationships involving the associated event types,…to determine whether a given set of propositions can all be true after some admissible event sequence …is the essence of the temporal reasoning problem …"

Dean and McDermott [21] describe the *temporal imagery* based approach to temporal reasoning. Intuitively, temporal imagery is the process of reasoning about relationships between events based on a temporal dependency graph (temporal map) for a collection of events. Dean and McDermott call their "brand" of temporal reasoning *shallow*, as, in their work, it is reduced to a sequence of "simple" steps. Given a collection of events and partial information about temporal dependencies among them, together with a goal of describing the situation at the next time instant, shallow temporal reasoning on each iteration consists of the following steps:

1. a set of candidate hypotheses is generated;
2. one of hypotheses is selected from the candidate set;
3. based on the hypothesis selected, a prediction is made;
4. "unforeseen" consequences are observed and introduced into the framework for response.

In [21] a (LISP-like) language for expressing different properties of events and dependencies (including temporal dependencies) between them is defined. The framework contains a notion of a reference point to specify the time instant at which the computation starts, but also has facilities to handle calendars (i.e., absolute time).

All the work cited above does not deal with quantitative uncertainty. While the acceptance of the fact that temporal reasoning (and non-monotonic reasoning in general) is universal, no mechanisms for dealing with it are considered in [77, 21, 48, 2, 68] and related work.

This does not, however, mean that there have been no work on bringing uncertain (and in particular probabilistic) inference to temporal reasoning.

Hart and Sharir [50] have introduced and studied Probabilistic Temporal Logics for verification of probabilistic programs. Their logics had the standard temporal logic syntax based on logics of branching time [9], but instead of Kripke structures, Markov chains were used as interpretations (models). Thus, one could not express statements about probabilities of events in their logic, rather, a probability distribution was associated with the set of all possible paths in the model.

Dean and Kanazawa [20] propose a probabilistic framework for solving the temporal projection problem, formulated by Dean and McDermott [21] as the problem of computing the consequences of a set of conditions/observations and a set of cause-and-effect relations. Dean and Kanazawa use probability distribution functions to represent the properties of statements to persist (i.e., keep their truth value) in time.

Godo and Vila [43] apply fuzzy logic to temporal reasoning. Their framework, in part based on the work of Dubois and Prade [31], describes temporal possibilistic inference. Godo and Vila consider two types of propositions in their system: temporal and atemporal. With temporal propositions, fuzzy temporal constraints, containing information about their possible occurrence are associated. [43] provides a sound inference system for manipulating such fuzzy constraints.

There had been a number of problem-oriented frameworks [44, 13, 49] for reasoning with uncertain (probabilistic) temporal information. Hanks *et al.*[49] proposes mechanisms for dealing with uncertain "exogenous" temporal events (i.e., events that can instantly and forcibly cause change in the current state) in the context of a medical expert system. The proposed model is equivalent to Partially Observable Markov Decision Processes [79]. Gumb [44] describes TEMPRO, an information management system, which is capable of interpreting incoming sensor data about terrain under uncertainty, which involves temporal uncertainty (incomplete or uncertain information about the position of different objects on terrain at different time instants). Bouzid and Mouaddib [13] apply fuzzy measures to the problem of distributed scheduling.

The work on temporal and uncertain temporal reasoning described above bears little similarity with our work on Temporal Probabilistic Databases. Even obvious issues, such as use of point probabilities, reliance on temporal logic (vs. our use of calendars to represent explicit time) and Markov chain semantics (for some frameworks [50, 49]) aside, there are still significant differences. Our work on Temporal Probabilistic Databases concentrates on issues of representing massive amount of information in a relational database format and conveniently querying it, which are not addressed in the work on temporal reasoning.

Temporal Probabilistic Logic Programs stand closer to the work on uncertain temporal reasoning described here both in goals and in approach. A subtle, yet important difference lies in the fact that our framework provides for a way of storing and querying temporal probabilistic information, while most of the research on temporal uncertainty described here [20, 13, 49] is concentrated around solving planning problems.

Chapter 16

Future Research

In this chapter the future directions of my research in the area of reasoning with probabilistic information and management of uncertain data is described.

16.1 nDimensional Probabilistic Databases

As mentioned in Section 15.3.1, when designing the TP-Database framework we chose to use generic probabilistic databases as the basis and add temporal features to them. One benefit is that it naturally extends to the management of probabilistic information on more complex domains. In particular, we are interested in designing a framework for managing nDimensional Probabilistic Information.

The n-space in which 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 designing a 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 we did 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.

16.2 Other Future Work

In this section we itemize other directions in which the results of this thesis could be extended.

- **Optimization for TP-Databases.** Our next step in the study of Temporal Probabilistic Databases is to design and implement techniques for query optimization for Temporal Probabilistic Algebra. We have already made one step in this direction by establishing a number of query equivalence results (Chapter 12). Robert Ross has also designed and implemented some index structures for TP-Databases.

  In the near future, we plan to develop cost models for TP-Databases and design enumeration techniques for query execution plans and efficient algorithms for the exploration of the search space of query execution plans.

- **View Maintenance and Updates for TP-Databases.** A key feature of TP-Databases is that there is no unique way to combine information from different sources into one relation. This was the reason why instead of a single compaction operation we have defined a family of compaction operations allowing the user to decide how to combine information according to his/her current needs and knowledge/assumptions about the data.

  Similarly, there is no unique way to update TP-Relations with new information. Our plan is to develop update methods for TP-Databases which would allow the user to choose the strategy for combining temporal probabilistic information currently stored in the relations with the newly added information.

  Related to this is the problem of incremental view maintenance. TP-Database views expressed as queries of Temporal Probabilistic Algebra may be fairly costly to recompute after every update made to the base relations, and in the databases where base relations are often updated, recomputing views from scratch would be prohibitive. We plan to develop the methods for incremental propagation of the changes made to base TP-relations onto any materialized views that can be potentially affected.

- **From Historic Databases to TP-Relations.** In many applications of TP-Databases, TP-Relations have to be created from pre-existing historic (temporal) databases. In fact, some of the most promising applications of TP-Databases are centered around gathering statistical information about the behavior of an application domain (such as the operations of a company, or the behavior of computer network) over a period of time and then combining it into a TP-Database in order to facilitate convenient and efficient querying.

  In order to successfully apply TP-Databases to such applications one has to be able to convert the information contained in the historic database documenting the said behavior into a TP-Database format. In that respect, a number of problems arise that need to be solved.

  - **Automated Conversion.** Given a historic database documenting the behavior of the system, our goal is to create TP-Database which would represent the necessary information about the statistics contained in the historic database (our assumption is that statistical information about past behavior can be used to estimate the probability of similar behavior in the future).

    Clearly, this process cannot be manual, as the size of the historic database can be very large and the manipulations on the data needed to extract temporal probabilistic components of the resulting TP-Relations may be quite complicated. Therefore, the first problem we are facing is the design of the system which would take as input a historic database and produce "on spec" the TP-Database summarizing the historic database (we envision that the "spec" would contain the description of what exact information should be contained in the output).

– **Incremental Updates.** Clearly, the process of creating a TP-Database from a historic database can be quite complex and time-consuming. However, the important feature of historic databases is that they are constantly being updated with the new information. Therefore, a TP-Database created from a historic database will merely represent the information contained in historic database at one time instant. With the addition of new information to the historic database, the discrepancy between the content of TP-Database created previously and the current content of the historic database can become fairly significant.

  We are interested in developing methods which would allow us to

  (a) Detect "critical mass" of updates to the historic database which should cause the update in the corresponding TP-Database. This "critical mass" can be determined based on certain conditions/ restrictions specified by the user.

  (b) Incrementally update the TP-Database in order to account for the new information. The incremental update procedures should be fast and update only the information that changed.

  We are also interested in developing evaluation techniques which would allow us to tell when to use incremental updates and when it is easier and "cheaper" to recreate the TP-Database from scratch.

We plan to develop the methods and techniques for converting historic information into TP-Databases and to implement them.

- **Extended Temporal Probabilistic Logic Programs.** One of the restrictions of the TPLP framework described in the thesis is that it is not an extension of classical logic programming. The reason for that is the assumption that all events represented by the atoms of a tp-program are instantaneous. This makes it impossible to represent probabilistic statements (such as "the probability that Mike is the father of Alex is $90 - 95\%$"). This framework also cannot deal with non-instantaneous events.

  The model theory for Temporal Probabilistic Logic Programs can be easily extended to allow for both probabilistic statements whose probability is *independent* of time and non-instantaneous events, i.e., events that can have a duration or reoccur. Our goal is to create a Logic Programming framework that supports such a model theory.

  One of the reasons why we are interested in such an extension of TPLP framework is because we expect that the analogs of non-instantaneous events will arise naturally in the case when we are trying to deal with nDimensional Probabilistic information. Knowing how to deal with such events should help us develop appropriate methods in the nDimensional Probabilistic framework.

- **Back to Probabilistic Databases.** While we have successfully developed TPA - the relational algebra that operates on compact representations of information (TP-relations), we have done it for a specific case of temporal probabilistic information. The general question of creating a relational algebra on a compact representation of a generic probabilistic information, asked by Lakshmanan et al. in [64] is, to our knowledge, still open.

Going back to the generic probabilistic database framework and solving this problem should present an interesting challenge.

Chapter 17

Conclusions

We have developed and studied Database and Logic Programming techniques to reason with probabilistic and temporal probabilistic information. Although each framework we have developed sports its own features, they share the same background and there is a general approach behind them all. In our approach to all the frameworks described in this thesis we paid close attention to the following two key issues:

- Clear intuition behind our semantics.

  This principle affected quite a few design decisions we made in the HPLP and TP-Database frameworks, including but not limited to the treatment of different derivations of the same formulas in HPLP and the design of operations in Temporal Probabilistic Algebra.

- 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 11.5) with the difference operation in ProbView ([64])). In cases like this, the first principle of providing more intuitive operations outweighs the compatibility principle, but in general we feel that the ideas behind all our frameworks are very similar.

The main contributions of our work on probabilistic reasoning and management of probabilistic information 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. Notions of consistency of TP- and annotated relations.
  - Definition of TATA (Theoretical Annotated Temporal Algebra) – the relational algebra on annotated relations;

  - Definition of TPA (Temporal Probabilistic Algebra) – the relational algebra on TP-relations;
  - Proof of correctness of implementation of TATA operations by TPA operations.
  - Description of query rewrite rules for TPA.

- **Temporal Probabilistic Logic Programs.**
  - Definition of syntax, model theory and fixpoint semantics for the extension of Ng and Subrahmanian's PLPs ([82]) onto temporal probabilistic framework.
  - Description of a sound and complete proof procedure.

BIBLIOGRAPHY

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

[2] M. Abadi, Z. Manna (1989). Temporal Logic Programming, in *Journal of Symbolic Computation*, 8,3, pp. 277–295.

[3] J.F. Allen. (1984) Towards a General Theory of Time and Action, *Artificial Intelligence*, 23, pps 123–154.

[4] F. Bacchus, On Probability Distributions of Possible Worlds, (1988), *Proc. Fourth Workshop on Uncertainty in Artificial Intelligence*, pps. 15 -21.

[5] F.Bacchus, *Representing and Reasoning With Probabilistic Knowledge*,(1990) MIT Press, Cambridge, Mass.,

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

[7] 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.

[8] M. Baudinet (1989). Temporal Logic Programming is Complete and Expressive, in *Proceedings 1989 ACM Symp. on Principles of Programming Languages*, pp. 267–280.

[9] M. Ben-Ari, A. Pnueli, Z. Manna (1983). The Temporal Logic of Branching Time, *Acta Informatica*, 20, pp. 207–226.

[10] H. Blair and V.S. Subrahmanian. Paraconsistent Logic Programming, *Theoretical Computer Science*, Vol. 68, pp 135–154, 1989.

[11] L.Blum, M.Shub, S. Smale (1989) On a theory of computation and complexity over the real numbers: NP-completeness, recursive functions and universal machines *Bulletin (New Series) of the American Mathematical Society*, 21(1) (1989) pp. 1–46.

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

[13] M. Bouzid, A-I. Mouaddib (1998) Uncertain Temporal Reasoning for the Distributed Transportation Scheduling Problem. in *Proceedings 5th Workshop on Temporal Representation and Reasoning, TIME '98*, pp. 21–28.

[14] 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.

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

[16] 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.

[17] L. Chiatarro, C. Combi, Temporal Indeterminacy in Deductive Databases: An Approach Based on Event Calculus, (1997) *in Proc. ARTDB'97, LNCS vol 1553*, pp 212-227, 1998

[18] J. Clifford, A. Croker, A. Tuzhilin, (1994) On Completeness of Historical Relational Query Languages (1994), *ACM Trans. on Database Systems*, vol 19, March 1994, pp. 64–116.

[19] F. Crestani, M. Lalmas, C. J. van Rijsbergen, I. Campbell: (1998) "Is This Document Relevant? ... Probably": A Survey of Probabilistic Models in Information Retrieval., *ACM Computing Surveys*, 30(4), pp. 528-552.

[20] T. Dean, K. Kanazawa (1988). Probabilistic Temporal Reasoning, in *Proceedings of the 7th National Conference on Artificial Intelligence*, AAAI Press / The MIT Press, vol 2., pp. 524–528.

[21] T.Dean, D. McDermott (1987) Temporal Database Management, *Artificial Intelligence*, 32, pp 1–55.

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

[23] A. Dekhtyar and V.S. Subrahmanian. (2000) Hybrid Probabilistic Programs, *Journal of Logic Programming*, Vol 43, Issue 3, pp 187–250.

[24] A. Dekhtyar, M. Dekhtyar and V.S. Subrahmanian. (1999) Temporal Probabilistic Logic Programs, *Proc. 1999 International Conference on Logic Programming*, MIT Press.

[25] A. Dekhtyar, R.Ross and V.S. Subrahmanian. (1998) Probabilistic Temporal Databases, I: Algebra, accepted to *ACM Transactions on Database Systems*, full version available as University of Maryland tech report CS-TR-3987.

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

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

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

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

[30] D. Dubois and H. Prade. (1991) Certainty and Uncertainty of Vague Knowledge and Generalized Dependencies in Fuzzy Databases. In *Proceedings International Fuzzy Engineering Symposium*, pp. 239–249, Yokohama, Japan, 1988.

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

[32] 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.

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

[34] C. Dyreson and R. Snodgrass. (1991) Historic Indeterminacy in Databases, University of Arizona technical report.

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

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

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

[38] N. Fuhr. (1995) Probabilistic Datalog – a Logic for Powerful Retrieval Methods., in E.Fox, P.Ingwerson, R. Fidel (eds.), *Proceedings of the 18th Annual ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 282–290, ACM, New-York.

[39] 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.

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

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

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

[43] L. Godo, L. Vila (1995). Possibilistic Temporal Reasoning based on Fuzzy Temporal Constraints, in *Proceedings 1995 IJCAI*, pp. 1916–1923.

[44] R. Gumb (1987). Filming a Terrain under Uncertainty Using Temporal and Probabilistic Reasoning, in *Proceedings AAAI-87 Sixth National Conference on Artificial Intelligence*, AAAI Press, pp. 116–120.

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

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

[47] J. Halpern, (1989) An Analysis of First Order Logics of Probability, *Artificial Intelligence*, 46, pp. 311-350

[48] S. Hanks, D. McDermott (1987). Nonmonotonic Logic and Temporal Projection, *Artificial Intelligence*, 33, pp. 379–412.

[49] S.Hanks, D. Madigan, J. Garvin (1995). Probabilistic Temporal Reasoning with Endogenous Change, in *Proceedings, 11th Conference on Uncertainty in Artificial Intelligence (UAI-95)*.

[50] S. Hart, M. Sharir (1986). Probabilistic Propositional Temporal Logics, *Information and Control*, 70(2/3), pp. 97–155.

[51] J. Jaffar, J-L. Lassez, Constraint Logic Programming (1987), *Proceedings of 14th ACM Symp. on Principles of Programming Languages*, pp. 111-119.

[52] P. Kanellakis, G. Kuper, P. Revesz, Constraint Query Languages (1990), in *Proceedings of the 9th ACM Symposium on Principles of Database Systems*, pp. 299 - 313.

[53] P. Kanellakis, G. Kuper, P. Revesz, Constraint Query Languages (1995), *J. of Computer and System Sciences*, 51(1): 26-52 .

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

[55] 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.

[56] 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.

[57] A.N. Kolmogorov. (1974) *Osnovnye Ponyatiya Teorii Veroyatnostey (Basic Notions of the Probability Theory)*, Moscow, Nauka, 2nd ed.

[58] H.Korth and A. Silberschatz. (1991) *Database Systems Concepts. Second Edition*, McGraw-Hill Inc.

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

[60] 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.

[61] M. Koubarakis, The Complexity of Query Evaluation in Indefinite Temporal Constraint Databases, (1997), *Theoretical Computer Science* 171(1-2): 25-60 (1997)

[62] R. Kowalski, M. Sergot, A Logic-based Calculus of Events, (1986), *New Generation Computing* 4(1): 67-95

[63] 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.

[64] 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.

[65] 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.

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

[67] V.S. Lakshmanan and N. Shiri. (1997) A Parametric Approach with Deductive Databases with Uncertainty, *Logic in Databases*, 1996, pp. 61–81.

[68] S-H. Lin, T. Dean (1993). Exploiting Locality in Temporal Reasoning, in *Proceedings 2nd European Workshop on Planning*.

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

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

[71] T. Lukasiewicz. (1998), Probabilistic Logic Programming, *Proc. 13th European Conference on Artificial Intelligence*, pp. 388-392, J. Wiley & Sons.

[72] Thomas Lukasiewicz. (1999) Many-Valued Disjunctive Logic Programs with Probabilistic Semantics. In *Proceedings of the 5th International Conference on Logic Programming and Nonmonotonic Reasoning, El Paso, Texas, USA, December 1999*. Volume 1730 of LNAI, pp. 277-289, Springer, 1999.

[73] Thomas Lukasiewicz. (1999) Many-Valued First-Order Logics with Probabilistic Semantics. In *Proceedings of the Annual Conference of the European Association for Computer Science Logic, Brno, Czech Republic, August 1998*. Volume 1584 of *Lecture Notes in Computer Science*, pages 415-429, Springer, 1999.

[74] Thomas Lukasiewicz. (1998), Probabilistic and Truth-Functional Many-Valued Logic Programming. *In Proceedings of the 29th IEEE International Symposium on Multiple-Valued Logic*, pages 236-241, Freiburg, Germany, May 1999.

[75] Thomas Lukasiewicz. (2000) Probabilistic Logic Programming with Conditional Constraints. Technical Report Nr. 1843-00-01, Institut fur Informationssysteme, Technische Universitat Wien, May 2000.

[76] K. Mariott, P.Stuckey, *Programming with constraints: an introduction*, (1998), MIT Press.

[77] J. McCarthy, P.J. Hayes (1969). Somne Philosophical Problems From the Standpoint of Artificial Intelligence, in B. Meltzer, D. Michie (eds.) *Machine Intelligence*, 4, Edinburgh University Press, pp. 463–502.

[78] J. McCarthy (1980). Circumscription – a Form of Non-Monotonic Reasoning, *Artificial Intelligence*, 13, pp. 27–39.

[79] G.Monahan (1982). A Survey of Partially Observable Markov Decision Processes: Theory, Models and Algorithms, *Management Science*, 28, pp. 1–16.

[80] L. Ngo, P. Haddawy. (1995) Probabilistic Logic Programming and Bayesian Networks, *ASIAN 1995*, pp. 286 – 300

[81] L. Ngo, P.Haddawy. (1997) Answering Queries from Context-Sensitive Probabilistic Knowledge Bases, *Theoretical Computer Science (special issue on uncertainty in databases and deductive systems)*, 171:147-177, 1997.

[82] R. Ng and V.S. Subrahmanian. (1993) Probabilistic Logic Programming, *Information and Computation*, 101, 2, pps 150–201, 1993.

[83] 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.

[84] R. Ng and V.S. Subrahmanian. (1995) Stable Semantics for Probabilistic Deductive

Databases, *Information and Computation*, 110, 1, pps 42-83.

[85] N. Nilsson. (1986) Probabilistic Logic, *Artificial Intelligence*, 28, pp 71–87.

[86] J. Pearl (1988) *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*, Morgan Kaufmann, San Mateo, CA.

[87] A. Pnueli (1977). The Temporal Logic of Programs. In *Proc. 18th IEEE Symp. on Foundations of Computer Science*, pp. 46–57.

[88] D. Poole. (1993) Probabilistic Horn Abduction and Beyesian Networks, *Artificial Intelligence*, 64(1), pp. 81-129.

[89] D. Poole (1993) Logic Programming, Abduction and Probability: a top-down anytime algorithm for estimating prior and posterior probabilities, *New Generation Computing*, 11(3-4), 377-400.

[90] H. Prade (1984) Lipski's approach to incomplete information databases restated and generalized in the setting of Zadeh's possibility theory, *Information Systems* 9(1), pp 27–42.

[91] H. Prade, C. Testemale. (1984) Generalizing Database Relational Algebra for the Treatment of Incomplete/Uncertain Information and Vague Queries, *Information Sciences* 34: pp. 115–143.

[92] R. Reiter (1980). A Logic for Default Reasoning, *Artificial Intelligence*, 13, pp. 81–132.

[93] P Revesz. (1995) Constraint Databases: A Survey, *Semantics in Databases 1995*, pp 209–246

[94] S. Ross. (1997) *Introduction to Probability Models*, Academic Press.

[95] 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.

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

[97] R. Shapiro, J Heskett. (1985) *Logistics Strategy. Cases and Concepts*, West Publishing Company.

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

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

[100] 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.

[101] 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.

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

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

[104] H. Turtle, W.B. Croft. (1990) Inference Networks for Document Retrieval, *Proceedings of ACM SIGIR 1990*, pp 1–24.

[105] J.D. Ullman. (1989) *Principles of Database and Knowledge Base Systems*, Computer Science Press, 1989.

[106] A. Urquhart. The Complexity of Propositional Proofs, *The Bulletin of Symbolic Logic*, 1(4):425-467, December 1995.

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

[108] L. Zadeh. (1974) Fuzzy Logic and Its Application to Approximate Reasoning, *IFIP Congress 1974*, pp. 591–594

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