ABSTRACT

Title of dissertation: MANAGING UNCERTAINTY AND ONTOLOGIES IN DATABASES

Edward Hung, Doctor of Philosophy, 2005

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

Nowadays a vast amount of data is generated in Extensible Markup Language (XML). However, it is necessary for applications in some domains to store and manipulate uncertain information, e.g. when the sensor inputs are noisy, or we want to store data that is uncertain. Another big change we can see in applications and web data is the increasing use of ontologies to describe the semantics of data, i.e., the semantic relationships between the terms in the databases.

As such information is usually absent from traditional databases, there is tremendous opportunity to ask new kinds of queries that could not be handled in the past. This provides new challenges on how to manipulate and maintain such new kinds of database systems.

In this dissertation, we will see how we can (i) incorporate and manipulate uncertainty in databases, and (ii) efficiently compute aggregates and maintain views on ontology databases.

First, I explain applications that require manipulating uncertain information in XML databases and maintaining web ontology databases written in Resource Description Framework (RDF). I then introduce the probabilistic semistructured PXML data model with two formal semantics. I describe a set of algebraic operations and its efficient implementation. Aggregations of PXML instances are studied with two semantics proposed: possible-worlds semantics and expectation semantics. Efficient algorithms with pruning are given and evaluated to show their feasibility. I introduce PIXML, an interval probability version of PXML, and develop a formal semantics for it. A query language and its operational semantics are given and proved to be sound and complete. Based on XML, RDF is a language used to describe web ontologies. RDQL, an RDF query language, is extended to support view definition and aggregations. Two sets of algorithms are given to maintain non-aggregate and aggregate views. Experimental results show that they are efficient compared with standard relational view maintenance algorithms.

MANAGING UNCERTAINTY AND ONTOLOGIES IN DATABASES

by

Edward Hung

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
2005

Advisory Committee:

Professor V.S. Subrahmanian, Chair/Advisor
Dr. Lise Getoor
Professor Hanan Samet
Professor Samir Khuller
Professor Stephen Kudla

© Copyright by
Edward Hung
2005

This dissertation is dedicated to my father and mother.

ACKNOWLEDGMENTS

I would like to give my sincerest thanks to my advisor, Prof. V.S. Subrahmanian, whose supervision and guidance is the greatest support for me to finish this research work and this dissertation.

I would also like to thank Dr. Lise Getoor for her valuable helps and comments in the PXML and PIXML research projects.

Besides, special thanks must be given to Yu Deng, with whom I have worked together for over a year. She has given a lot of help and interesting ideas to me and contributed to the RDF research projects.

I also have to thank for the Croucher Foundation Scholarhips, Jacob K. Goldhaber Travel Grant and the financial support from the Army Research Lab, the National Science Foundation, the CTA on Advanced Decision Architectures and Telecommunications, and the Army Research Office.

As a research student in the Department of Computer Science at the University of Maryland, College Park, my life has been an enjoyable one. For this, I express my thanks to all my colleagues.

Last but not least, I have to thank my father, mother, brother and Pei-Ju for their enduring support and encouragement. Without them, this dissertation could not be finished and I would not be able to obtain the degree of Doctor of Philosophy.

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1

Introduction

1.1 New Challenges in XML Databases

Over the last few years, there has been considerable interest in Extensible Markup Language (XML) databases. A proliferation of semistructured data models have been proposed [64, 58, 76, 7], along with associated query languages [3, 18] and algebras [6, 42]. XML is a simple but very flexible markup language derived from SGML, which is now mainly used for exchange, transmission and manipulation of data on the web [11]. XML tags are not predefined, which means that it gives users flexibility to define their own tags according to their domains and applications.

Nowadays, while data is more often generated in XML for easier data transmission and manipulation over the web, it is necessary for applications in some domains to store and manipulate uncertain information. For example, this occurs when the sensor inputs are noisy. Another big change we can see in applications and web data is the increasing use of ontologies to describe the semantics of data, i.e., the semantic relationships between the terms stored in the databases. As such information is usually absent from traditional databases, there is tremendous opportunity to ask new kinds of queries that could not be handled in the past. This provides new challenges on how to manipulate and maintain such new kinds of database systems. In this dissertation, I will describe how we can (i) incorporate and manipulate uncertain information in databases, and (ii) efficiently compute aggregates and maintain views on ontology databases.

In the following sections, I will briefly introduce the motivations underlying the above two problems, my contributions and the organization of this dissertation.

1.2 Uncertainty in XML

The semistructured data model has the advantage of not placing hard constraints on the *structure* of the data. However, a particular semistructured instance specifies deterministic relationships between objects. In cases where we would also like to avoid hard constraints on the object-level structure, it is desirable to have a model that allows us to represent uncertainty over the relationships between objects in the semistructured model. This uncertainty is necessary when relationships between objects and values for attributes of objects are not known with absolute certainty.

There are numerous applications (including financial, image processing, manufacturing and bioinformatics) for which a probabilistic XML data model is quite natural and for which a query language that supports probabilistic inference provides important functionality. Probabilistic inference supports capabilities for predictive and 'what-if' types of analysis. For example, consider the use of a variety of predictive programs[10] for the stock market. Such programs usually return probabilistic information. If a company wanted to export this data into an XML format, they would need methods to store probabilistic data in XML. The financial marketplace is a hotbed of both predictive and XML activity (e.g. the FIX standard for financial data is XML based). There is the same need to store probabilistic data in XML for programs that predict expected energy usage and cost, expected failure rates for machine parts, and in general, for any predictive program. Another useful class of applications where there is a need for probabilistic XML data is image processing programs that process images (automatically) using image identification methods and store the results in an XML database. Such image processing algorithms often use statistical classifiers[44] and often yield uncertain data as output. If such information is to be stored in an XML database, then it would be very useful to have the ability to automatically query this uncertain information. Another important application is in automated manufacturing monitoring and diagnosis. A corporate manufacturing floor may use sensors to track what happens on the manufacturing floor. The results of the sensor readings may be automatically piped to a fault diagnosis program that may identify zero, one, or many possible faults with a variety of probabilities on the space of faults. When such analysis is stored in a database, there is a natural need for probabilities. In addition to these types of applications, information extraction using probabilistic parsing of input sources may also result in a semistructured instance in which there is uncertainty. For example, the NSIR system for searching documents at the University of Michigan[70] returns documents based along with probabilities. Likewise, Nierman and Jagadish. point out the use of probabilistic semistructured databases in protein chemistry[61].

While there has been a great deal of work on supporting uncertainty in relational models [12, 54, 56, 50, 19, 24, 33, 30], to date, there has been little work on supporting uncertainty in semistructured models. There are a few exceptions including [16] and [61]. Dekhtyar et al.[16] proposed a model that allows probabilistic information to be stored using semistructured databases. My proposal does the opposite: I extend the semistructured data model so that paths in such a model can include probabilistic information. More closely related to my work is the work of Nierman and Jagadish[61], in which a tree-structured probabilistic database is proposed. I will show that their model is a special case of my probabilistic semistructured PXML model.

1.3 Ontologies in XML

The World Wide Web Consortium (W3C) has recently designated Resource Description Framework (RDF) as a web recommendation endorsed by approximately 300 companies, bringing it close to the status of being a de facto standard for web semantics. RDF data is basically XML data written in a specific structure with some specific tags defined to describe the ontology (relationships) between resources. Though RDF has many complex features, the basic idea is to describe "resource, property, value" triples specifying that a given resource has a given value for the described property. RDF databases are expected to store such triples about vast numbers of web pages and other information resources so that users can query the web using a sophisticated, database style query language rather than using simple keyword search supported by most current web search engines. This has caused a growing interest in maintaining *databases* of RDF data [71] for the purpose of discovering web resources using a richer query language than that offered by current web search engines. An indication of current interest in RDF is the RDQL query language from Hewlett Packard[41], and its impressive (prototype) Jena RDF database system[43]. Other RDF query languages include RQL[47] and SeRQL[62]. Methods to express RDF views with RQL were first described in [79]. *However, to date, the problem of incremental maintenance of materialized* RDF *views has not been studied.*

In this dissertation, I study the problem of efficiently maintaining views over RDF databases. There is currently no industry-wide consensus on the best RDF query language. I have chosen to build view maintenance algorithms on top of HP's RDQL language as RDQL is one of the leading industry contenders to become a standard RDF query language. The problem of maintaining RDF views is different from that of maintaining views in XML databases, graph-structured databases (GSDBs)[87], object-oriented databases and relational databases for several reasons. (i) GSDB data such as XML assume a rooted graph model, whereas RDF databases assume a general graph model. (ii) Materialized GSDB views contain a set of nodes, whereas RDF views are rich enough to return not only nodes, but also graphs and other combinations. (iii) The method used to specify views are quite different. These differences persist when considering object-oriented and relational databases as well which in many ways are even less expressive than GSDBs.

1.4 Contributions

The first contribution of this dissertation is a flexible probabilistic representation for semistructured data that supports arbitrary distributions over the relationships between an object and its children and arbitrary distributions over the object's value. As we will see, my model does not require the semistructured instance to be tree structured; however, it does require that the probabilistic dependencies are acyclic.

The second major contribution is a formal characterization of the probabilistic semantics of the model. This connection is missing in previous approaches to represent probabilistic semistructured data. In particular, I propose two semantics - the first semantics (or "global" semantics) is a possible-worlds-based approach that hypothesizes that the world is always certain, but it is us who are uncertain about what is true. According to this semantics, a probabilistic XML database instance is shorthand for a set of (ordinary) semistructured database instances. I show how the definition of a PXML instance formally defines a set of "compatible" semistructured instances, and how the PXML instance can induce a probability distribution over the set of compatible instances. The second semantics ("local semantics") exploits a factorization of the probability distribution. I show several important results: (i) the local semantics can be embedded in the global semantics, (ii) the converse is not always true - I identify *conditions* under which it is true, and (iii) I provide a nontrivial construction for a Bayesian network that encodes the local semantics. This means that for some simple queries, Bayesian inference can be used to reason with the local semantics (but not the global).

The third major contribution is an algebra that supports querying probabilistic semistructured data, including selection, projection, and cartesian product (join can be handled as a combination of cartesian product and selection). One of the important features of my PXML algebra is that all operations occur directly on PXML instances.

The fourth major contribution is the introduction of aggregate operations on PXML instances. I provide two semantics for PXML aggregate operators. The possible-worlds semantics returns a *set of possible answers* to aggregate queries (together with associated probabilities). Intuitively, these possible answers correspond to the evaluation of the aggregate query in different possible worlds. The second semantics is an expected value semantics. I then show how PXML instances can be *directly manipulated* so that the need to explicitly compute compatible instances is avoided. Pruning methods are also proposed.

The fifth contribution is a prototype implementation of PXML. I have conducted a suite of experiments, which show that PXML can be effectively implemented. Experiments with PXML aggregates show the performance of algorithms for both semantics with the clear outcome that the expected value semantics is more practical to compute. On the other hand, pruning techniques used in the possible-worlds semantics also produce good approximate answers in acceptable running time.

The sixth contribution is that I extend PXML to use interval probabilities rather than point probabilities to represent uncertainty. The result is the PIXML probabilistic interval data model. I then provide two alternative formal semantics for PIXML. The first semantics is a declarative (model-theoretic) semantics. The second semantics is an operational semantics that can be used for computation. In the formal W3C specification of XML, an instance is considered as an ordered rooted tree in which cycles can possibly appear[80]. I assume that an instance is an acyclic graph - this assumption will be needed to provide a coherent semantics to PIXML databases. However I do not restrict attention to tree-structures. I also provide an operational semantics that is provably correct for a class of queries over a large class of probabilistic instances called tree-structured instances.

The seventh contribution is that I describe how to extend a commercial RDF language called RDQL (proposed by Hewlett Packard) to support views and aggregations. I provide algorithms called IMA, DMA, TMA, RMA to incrementally maintain views when insertions, deletions, triple modifcations and resource modifications are made to an RDF database instance. I implement a prototype of these algorithms and experimentally show that the my new algorithms are significantly better than the use of standard view maintenance algorithms on relational representations of RDF databases.

The eighth contribution is that I extend the problem of maintaining non-aggregate RDF views to maintaining aggregate RDF views. I propose the CAA (Compute Aggregates Algorithm) algorithm to efficiently compute *aggregate* operations such as COUNT,SUM,AVG,MIN,MAX and so on. CAA can also handle GROUPBY queries. I subsequently define algorithms to maintain aggregate views. These are views involving aggregate queries. I split aggregate functions into two categories - *distributive* and *non-distributive* aggregates. I provide algorithms (called AMI and AMD) to maintain aggregate views when insertions and deletions are made. In addition, I provide methods to maintain aggregate views when triples are modified (called AMT) and when resources (called AMR) are modified. I also note that RDF databases can be easily stored in relational form. As a consequence, standard algorithms to maintain aggregate relational views can be implemented to maintain RDF views. I have implemented this strategy and compared it to my implementation of AMI, AMD, AMR and AMT – my algorithms are much faster than performing view maintenance on the relational version. The results show that, when the database is updated, my incremental maintenance algorithms work much faster than a complete recomputation by an order of 10 to 1000 and about 1.8 to 1109 times faster than the relational implementation.

1.5 Organization

Chapter 2 describes the PXML model, algebra and aggregation. I first start with a motivating example and propose the Probabilistic XML (PXML) model of probabilistic semistructured databases. I define the semantics for probabilistic semistructured databases and propose an extension of the relational algebra operators to apply to probabilistic semistructured databases. Then, I describe a formal model of probabilistic aggregates with algorithms to compute them efficiently. I then present experimental results to evaluate the efficiency of algorithms.

In Chapter 3, I show how to extend the PXML model to the PIXML probabilistic interval data model which uses interval probabilities rather than point probabilities. I then provide two alternative formal semantics for PIXML with an operational semantics for queries.

In Chapter 4, I first introduce the reader to the basics of RDF and RDF aggregates and describe how to extend a commercial RDF language called RDQL (proposed by Hewlett Packard) to support aggregations. I propose algorithms to compute aggregates, maintain non-aggregate views and aggregate views of RDF databases. I then present experimental results which show that, when the database is updated, my incremental maintenance algorithms work much faster than a complete recomputation and the relational implementation.

I discuss related work in Chapter 5. Chapter 6 concludes this dissertation.

Chapter 2

Probabilistic XML Model, Algebra and Aggregation

In this chapter, I describe the PXML model, algebra and aggregation. I first start with a motivating example in Section 2.1. In Section 2.2, I propose the Probabilistic XML (PXML) model of probabilistic semistructured databases. Then in Section 2.3, I define two semantics for probabilistic semistructured databases. The first is a *global* semantics (in a sense to be made precise) while the second is a *local* semantics. I show that the two semantics are equivalent. In Section 2.4, I propose *probabilistic point queries*, that return the probabilities that particular objects exist satisfying some constraints. Then, in Section 2.5, I propose an extension of the relational algebra operators to apply to probabilistic semistructured databases. I define the operations of select, project and Cartesian product. I also give algorithms that exploit the local semantics, and result in efficient computation of the results of these algebraic operations. Then, in Section 2.6, I describe a formal model of probabilistic aggregates. Section 2.7 contains algorithms to compute probabilistic aggregates efficiently. Section 2.8 contains experimental results that evaluate the efficiency of algorithms implementing the PXML algebra and aggregation.

2.1 Motivating Examples

In this section, I provide two applications as our motivating examples used throughout this chapter to illustrate my proposed PXML model, semantics, algebra, query and aggregate operators.

2.1.1 A Bibliographical Application

As our first running example, I will use a bibliographic domain. This example is rather simple, but I assume it will be accessible to all readers. In this case, I assume that the uncertainty arises from the information extraction techniques used to construct the bibliography. Consider a citation

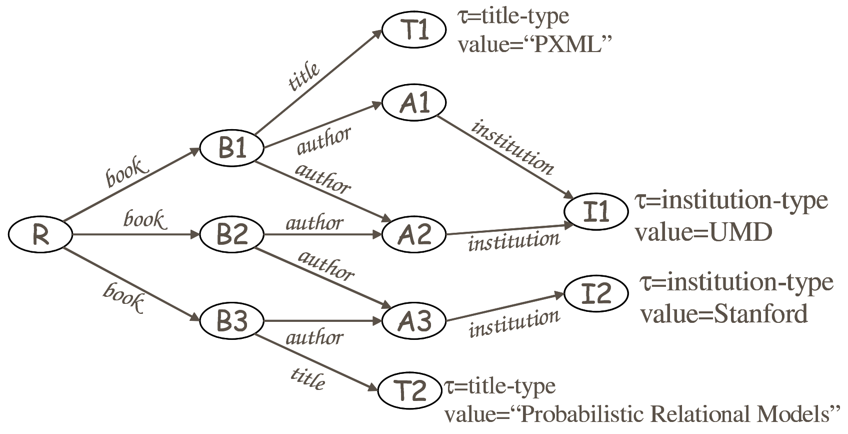


Figure 2.1: A semistructured instance for a bibliographic domain.

index such as Citeseer [1] or DBLP [2]. In Citeseer, the indexes are created by crawling the web, and operations include parsing postscript and PDF documents. Often, there will be uncertainty over the existence of a reference (have we correctly identified a bibliographic reference?), the type of the reference (is the reference a conference paper, a journal article or a book?), the existence of subfields of the reference such as author, title and year, the identity of the author (does Hung refer to Edward Hung or Sheung-lun Hung or many other tens of authors with "Hung" as their last names or first names?). In such environments, uncertainty abounds.

Semistructured data is a natural way to store such data because for an application of this kind, we have some idea of what the structure of data looks like (e.g. the general hierarchical structure alluded to above). However, semistructured data models do not provide support for uncertainty over the relationships in an instance. In this paper, I extend this model to naturally store the uncertainty that we have about the structure of the instance as well. Furthermore, we will see how my algebraic operations and query mechanisms can perform the following manipulations:

1. Find a list of authors (without titles, institutions, etc.) and return an object which allows further querying.

[1]`http://citeseer.nj.nec.com/cs/`

[2]`http://www.informatik.uni-trier.de/~ley/db/`

2. Merge two databases together.

3. Find the probability that a particular individual is the author of some book.

2.1.2 A Surveillance Application

Another example is a surveillance application where a battlefield is being monitored. Image processing methods are used to classify objects appearing in images. Some objects are classified as vehicle convoys or refugee groups. Vehicle convoys may be further classified into individual vehicles, which may be further classified into categories such as tanks, cars, armored personnel carriers. However, there may be uncertainty over the number of vehicles in a convoy as well as the categorization of a vehicle. For example, image processing methods often use statistical models to capture uncertainty in their identification of image objects. Further uncertainty may arise because image processing methods may not explicitly extract the identity of the objects. Semistructured data is a natural way to store such data because for a surveillance application of this kind, we have some idea of what the structure of data looks like (e.g. the general structure described above). However, the above example demonstrates the need for a semistructured model to store uncertain information in uncertain environments.

Aggregate queries are natural queries for users to ask in such applications. To date, we are aware of no formal model of aggregate computations in probabilistic XML databases. Examples of queries that users may wish to ask include: *How many convoys are there (in some collection of images)? How many tanks are there in total? On the average, how many tanks are there in a convoy? What is the ratio of the total number of tanks to the total number of trucks?* In more complex examples, there are many other important queries. If convoys include an estimate of the number of soldiers per vehicle, we may be interested in the total number (sum) of soldiers. We may also be interested in the average number of soldiers per convoy, the average number of soldiers per tank, and so on.

2.2 Probabilistic Semistructured Data Model

In this section, I introduce the PXML model. I first review the definition of a semistructured data model. I then introduce the syntax of PXML followed by the semantics of PXML.

2.2.1 Semistructured Data Model

I start by recalling some simple graph concepts.

Definition 2.2.1 *Let V be a finite set of vertices, $E \subseteq V \times V$ be a set of edges and $\ell : E \rightarrow \mathcal{L}$ be a mapping from edges to a set $\mathcal{L}$ of strings called* labels*. The triple $G = (V, E, \ell)$ is an* **edge labeled directed graph**.

As usual, a graph is *rooted* iff there is a distinguished node called the root such that for every node in the graph, there is a path in the graph from the root to that node. Unless otherwise noted, we will assume that G is rooted.

Definition 2.2.2 *Suppose $G = (V, E, \ell)$ is any rooted, edge-labeled directed graph. For $o \in V$:*

- *The* **children** *of o, denoted $\mathsf{C}(o)$, is the set $\{o' \mid (o, o') \in E\}$.*
- *The* **parents** *of o, $\mathsf{parents}(o)$, is the set $\{o' \mid (o', o) \in E\}$.*
- *The* **descendants** *of o is the set $\mathsf{des}(o) = \{o' \mid$ there is a directed path from o to o' in $G\}$, i.e., o's descendants include o's children as well as the children of o's descendants.*
- *The* **non-descendants** *of o is the set $\mathsf{non\text{-}des}(o) = \{o' | o' \in V \wedge o' \notin \mathsf{des}(o) \cup \{o\}\}$, i.e., all vertices except o's descendants are o's non-descendants.*
- *We use $\mathsf{lch}(o, l)$ to denote the* **set of children of o with label l**. *More formally,*

$$\mathsf{lch}(o, l) = \{o' \mid (o, o') \in E \wedge \ell(o, o') = l\}.$$

- *A vertex o is called a* **leaf** *iff $\mathsf{C}(o) = \emptyset$.*

I also introduce a set $\mathcal{T}$ of **types**, each type $T \in \mathcal{T}$ has an associated finite **domain**, denoted $\mathsf{dom}(T)$. For each object, $\tau(o)$ returns the type of the object and I have a function val which maps an object o to a value in the domain of $\tau(object)$.

It is important to note that my graphs are not restricted to trees— in fact, the above definition allows graphs. As we will see later, while I allow semistructured data instances to be graph-structured, I require the probabilistic dependencies among the nodes to be acyclic.

As I plan to build upon existing models of semistructured databases, I start by recapitulating the definition of a semistructured instance from [1]. I start by assuming the existence of some arbitrary but fixed set $\mathcal{O}$ of strings called object-ids (oids for short), and a set $\mathcal{T}$ of types. Each type $T \in \mathcal{T}$ has an associated finite domain, $\mathsf{dom}(T)$.

Definition 2.2.3 *A* **semistructured instance** $\mathcal{S}$ *over a set of objects* $\mathcal{O}$*, a set of labels* $\mathcal{L}$*, and a set of types* $\mathcal{T}$*, is a 5-tuple* $\mathcal{S} = (V, E, \ell, \tau, \mathsf{val})$ *where:*

1. $G = (V, E, \ell)$ *is a rooted, directed graph where* $V \subseteq \mathcal{O}$*,* $E \subseteq V \times V$ *and* $\ell : E \to \mathcal{L}$*;*
2. τ *associates a type in* $\mathcal{T}$ *with each leaf object* o *in* G*.*
3. val *associates a value in the domain* $\mathsf{dom}(\tau(o))$ *with each leaf object* o*.*

I illustrate the above definition through an example from the bibliographic domain.

Example 2.2.1 *Figure 2.1 shows a graph representing a part of the bibliographic domain. The instance is defined over the set of objects* $\mathcal{O} = \{R, B1, B2, B3, T1, T2, A1, A2, A3, I1, I2\}$*. The set of labels is* $\mathcal{L} = \{book, title, author, institution\}$*. There are two types, title-type and institution-type, with domains given by:* $\mathsf{dom}(\textit{title-type}) = \{\textit{PXML}, \textit{Probabilistic Relational Models}\}$ *and* $\mathsf{dom}(\textit{institution-type}) = \{\textit{Stanford}, \textit{UMD}\}$*. The graph shows that the relationships between the objects in the domain and the types and values of the leaves.*

2.2.2 The PXML Probabilistic Data Model

In this section, I develop the basic syntax of the PXML probabilistic data model. Before defining the important concept of a probabilistic instance, I need to introduce some intermediate concepts.

Definition 2.2.4 (probability distribution) *A probability distribution w.r.t.* S *is a mapping* $\mathcal{P} : S \to [0, 1]$ *where* $\Sigma_{s \in S} \mathcal{P}(s) = 1$*.*

A central notion that allows us to provide coherent probabilistic semantics is that of a weak instance. A weak instance describes the objects that can occur in a semistructured instance, the labels that can occur on the edges in an instance and constraints on the number of children an object might have. I will later define a probabilistic instance to be a weak instance annotated with probabilistic information that will provide us with a distribution over semistructured instances consistent with the weak instance.

Definition 2.2.5 *A* **weak instance** $\mathcal{W}$ *with respect to* $\mathcal{O}$, $\mathcal{L}$ *and* $\mathcal{T}$ *is a* 5*-tuple* $\mathcal{W} = (V, \mathsf{lch}, \tau, \mathsf{val}, \mathsf{card})$ *where:*

1. $V \subseteq \mathcal{O}$.
2. *For each object* $o \in V$ *and each label* $l \in \mathcal{L}$, $\mathsf{lch}(o, l)$ *specifies the objects that* **may** *be children of* o *with label* l*. I assume that for each object* o *and distinct labels* l_1, l_2, $\mathsf{lch}(o, l_1) \cap \mathsf{lch}(o, l_2) = \emptyset$. [3]
3. τ *associates a type in* $\mathcal{T}$ *with each leaf object.*
4. val *associates a value in* $dom(\tau(o))$ *with each leaf object* o.
5. card *is a mapping that constrains the number of children with a given label* l. card *associates with each object* $o \in V$ *and each label* $l \in \mathcal{L}$*, an integer-valued interval* $\mathsf{card}(o, l) = [min, max]$*, where* $min \geq 0$*, and* $max \geq min$*. I use* $\mathsf{card}(o, l).min$ *and* $\mathsf{card}(o, l).max$ *to refer to the lower and upper bounds respectively.*

A weak instance implicitly defines, for each object and each label, a set of potential sets of children. Consider the following example.

Example 2.2.2 *Consider a weak instance with* $V = \{R, B1, B2, B3, T1, T2, A1, A2, A3, I1, I2\}$. *We may have* $\mathsf{lch}(R, book) = \{B1, B2, B3\}$ *indicating that B1, B2 and B3 are possible book-children of R. Likewise, we may have* $\mathsf{lch}(B1, author) = \{A1, A2\}$*. If* $\mathsf{card}(B1, author) = [1, 2]$*, then B1 can have between one and two authors. The set of possible* author*-children of B1 is thus*

[3] This condition says that two edges with different labels cannot lead to the same child; this condition can be relaxed, I make it here to simplify exposition.

$\{\{A1\}, \{A2\}, \{A1, A2\}\}$. *Likewise, if* $\mathsf{card}(A1,\ institution) = [1,1]$ *then* $A1$ *must have exactly one (primary) institution.*

I formalize the reasoning in the above example below.

Definition 2.2.6 *Suppose* $\mathcal{W} = (V, \mathsf{lch}, \tau, \mathsf{val}, \mathsf{card})$ *is a weak instance and* $o \in V$ *and* l *is a label. A set* c *of objects in* V *is a* **potential** l*-child set of* o *w.r.t. the above weak instance iff:*

1. *If* $o' \in \mathsf{c}$ *then* $o' \in \mathsf{lch}(o, l)$ *and*
2. *The cardinality of* c *lies in the closed interval* $\mathsf{card}(o, l)$*.*

I use the notation $\mathsf{PL}(o, l)$ *to denote the set of all potential* l*-child sets of* o*.*

As $\mathsf{PL}(o, l)$ denotes the set of all potential child sets of o with labels l, I define the set of all potential child sets of o with *any* label as the following:

Definition 2.2.7 *Suppose* $\mathcal{W} = (V, \mathsf{lch}, \tau, \mathsf{val}, \mathsf{card})$ *is a weak instance and* $o \in V$*. Let* H_o *be a set of all the sets of potential l-child sets of object o,*

$$H_o = \bigcup_{l \in \{\mathsf{PL}(o,l) \mid (\exists o') o' \in \mathsf{lch}(o,l)\}} \mathsf{PL}(o, l).$$

(Note that H_o *is a multiset.) The* **potential child set** *of* o*, denoted* $\mathsf{PC}(o)$*, is a hitting set*[4] *of* H_o*.*

Once a weak instance is fixed, $\mathsf{PC}(o)$ is well defined for each o. I will use this to define the *weak instance graph* below. We will need this in the definition of a probabilistic instance.

Definition 2.2.8 *Given a weak instance* $\mathcal{W} = (V, \mathsf{lch}, \tau, \mathsf{val}, \mathsf{card})$*, the* **weak instance graph***,* $\mathcal{G}_{\mathcal{W}} = (V, E)$*, is a graph over the same set of nodes* V*, and for each pair of nodes* o *and* o'*, there is an edge from* o *to* o' *iff* $\exists c \in \mathsf{PC}(o)$ *such that* $o' \in c$*.*

Figure 2.2 shows a weak instance graph for the bibliographic domain. An important requirement when defining a probabilistic semantics is that the probabilities of all potential child sets sum to 1.

[4]Suppose $\mathbf{S} = \{S_1, \ldots, S_n\}$ where each S_i is a set. A ***hitting set*** for $\mathbf{S}$ is a set H such that (i) for all $1 \le i \le n$, $H \cap S_i \ne \emptyset$ and (ii) there is no $H' \subset H$ satisfying condition (i).

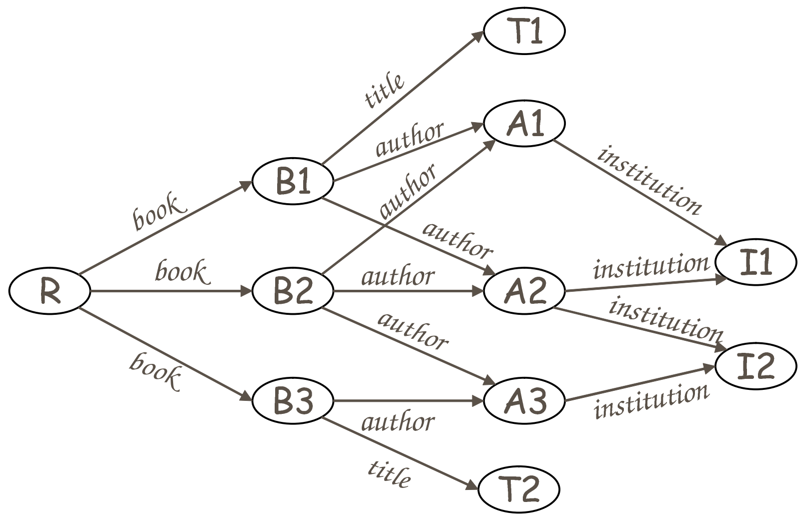


Figure 2.2: A weak instance graph for the bibliographic domain.

Definition 2.2.9 *Suppose* $\mathcal{W} = (V, \mathsf{lch}, \tau, \mathsf{val}, \mathsf{card})$ *is a weak instance. Let* $o \in V$ *be a non-leaf object. An* **object probability function** *(OPF for short) for* o *w.r.t.* $\mathcal{W}$ *is a mapping* $\omega : \mathsf{PC}(o) \rightarrow [0, 1]$ *such that OPF is a legal probability distribution, i.e.,* $\Sigma_{\mathsf{c} \in \mathsf{PC}(o)} \omega(\mathsf{c}) = 1$.

Definition 2.2.10 *Suppose* $\mathcal{W} = (V, \mathsf{lch}, \tau, \mathsf{val}, \mathsf{card})$ *is a weak instance. Let* $o \in V$ *be a leaf object. A* **value probability function** *(VPF for short) for* o *w.r.t.* $\mathcal{W}$ *is a mapping* $\omega : \mathsf{dom}(\tau(o)) \rightarrow [0, 1]$ *such that VPF is a legal probability distribution, i.e.,* $\Sigma_{\mathsf{v} \in \mathsf{dom}(\tau(o))} \omega(\mathsf{v}) = 1$.

An object probability function provides the model theory needed to study a single non-leaf object (and its children) in a probabilistic instance to be defined later. It defines the probability of a set of children of an object existing *given* that the parent object exists. Thus it is the conditional probability for a set of children to exist, under the condition that their parent exists in the semistructured instance. As we will see later, it is akin to the conditional probabilities specified in graphical models or Bayesian networks [66], however a key difference is that it describes the local *structure* of the network. Similarly, the value probability function provides the model theory needed to study a leaf object, and defines a distribution over values for the object.

Definition 2.2.11 *Suppose* $\mathcal{W} = (V, \mathsf{lch}, \tau, \mathsf{val}, \mathsf{card})$ *is a weak instance. A* **local interpretation** *is a mapping* $\wp$ *from the set of objects* $o \in V$ *to local probability functions. For non-leaf objects,* $\wp(o)$ *returns an OPF, and for leaf objects,* $\wp(o)$ *returns a VPF.*

Intuitively, a local interpretation specifies, for each object in the weak instance, a local probability function.

Definition 2.2.12 *A* **probabilistic instance** $\mathcal{I}$ *is a* 6*-tuple* $\mathcal{I} = (V, \mathsf{lch}, \tau, \mathsf{val}, \mathsf{card}, \wp)$ *where:*

1. $\mathcal{W} = (V, \mathsf{lch}, \tau, \mathsf{val}, \mathsf{card})$ *is a weak instance and*
2. $\wp$ *is a local interpretation.*

A probabilistic instance consists of a weak instance, together with a probability associated with each potential child set of each object in the weak instance.

Example 2.2.3 *Figure 2.3 shows a very simple probabilistic instance. The set* $\mathcal{O}$ *of objects is the same as in our earlier* PXML *example. The figure shows the potential* lch *of each object; for example,* $\mathsf{lch}(B1, author) = \{A1, A2\}$*. The cardinality constraints are also shown in the figure; for example, object B1 can have 1 to 2 authors and 0 to 1 title. The tables on the right of Figure 2.3 show the local probability models for each of the objects. The tables show the probability of each potential child of an object. For example, if B2 exists, the probability A1 is one of its authors is* 0.8.

The components $\mathcal{O}, \mathcal{L}, \mathcal{T}$ of a probabilistic instance are identical to those in a semistructured instance. However, in a probabilistic instance, there is uncertainty over:

- The number of sub-objects of an object o;
- The identity of the sub-objects.
- The values of the leaf objects.

This uncertainty is captured through the function $\wp(o)$. $\wp(o)$ may be defined extensionally, defining a probability for *each* potential child of every object, as we have done here. Or we may define

o	l	$\mathsf{lch}(o,l)$
R	book	{B1, B2, B3}
B1	title	{T1}
B1	author	{A1, A2}
B2	author	{ A1, A2, A3}
B3	title	{T2}
B3	author	{A3}
A1	institution	{I1}
A2	institution	{I1, I2}
A3	institution	{I2}

o	l	$\mathsf{card}(o,l)$
R	book	[2,3]
B1	author	[1,2]
B1	title	[0,1]
B2	author	[2,2]
B3	author	[1,1]
B3	title	[1,1]
A1	institution	[0,1]
A2	institution	[1,1]
A3	institution	[1,1]

$\mathsf{c} \in \mathsf{PC}(R)$	$\wp(R)(\mathsf{c})$
{B1, B2}	0.2
{B1, B3}	0.2
{B2, B3}	0.2
{B1, B2, B3}	0.4

$\mathsf{c} \in \mathsf{PC}(B1)$	$\wp(B1)(\mathsf{c})$
{A1}	0.3
{A1, T1}	0.35
{A2}	0.1
{A2, T1}	0.15
{A1, A2}	0.05
{A1, A2, T1}	0.05

$\mathsf{c} \in \mathsf{PC}(B2)$	$\wp(B2)(\mathsf{c})$
{A1, A2}	0.4
{A1, A3}	0.4
{A2, A3}	0.2

$\mathsf{c} \in \mathsf{PC}(B3)$	$\wp(B3)(\mathsf{c})$
{A3, T2}	1.0

$\mathsf{c} \in \mathsf{PC}(A1)$	$\wp(A1)(\mathsf{c})$
{}	0.2
{I1}	0.8

$\mathsf{c} \in \mathsf{PC}(A2)$	$\wp(A2)(\mathsf{c})$
{I1}	0.5
{I2}	0.5

$\mathsf{c} \in \mathsf{PC}(A3)$	$\wp(A3)(\mathsf{c})$
{I2}	1.0

Figure 2.3: A probabilistic instance for the bibliographic domain.

$\wp(o)$ more compactly if there are some symmetries or independence constraints that can be exploited in the representation. For example, if the occurrence of each category of labeled objects is independent, then we can simply specify a probability for each subset of objects with the same label and compute the joint probability as the product of the individual probabilities. For instance, if the existence of author and title objects is independent, then we only need to specify a distribution over authors and a distribution over titles. Furthermore, in some domains it may be the case that some objects are indistiguishable. For example in an object recognition system, we may not be able to distinguish between vehicles. Then if we have two vehicles, vehicle1 and vehicle2, and a bridge bridge1 in a scene S1, we may not be able to distinguish between a scene that has a bridge1 and vehicle1 in it from a scene that has bridge1 and vehicle2 in it. In this case, $\wp(S1)(\{bridge1, vehicle1\}) = \wp(S1)(\{bridge1, vehicle2\})$. The semantics of the model we have proposed is fully general, in that we can have arbitrary distributions over the sets of children of an object.

2.3 Semantics

In this section, I develop a semantics for probabilistic semistructured databases. We can use a PXML model to represent our uncertainty about the world as a distribution over possible semistructured instances. A probabilistic instance *implicitly* is shorthand for a set of (possible) semistructured instances—these are the only instances that are *compatible* with the information we do have about the actual world state that is defined by our weak instance. I begin by defining the notion of the set of semistructured instances that are compatible with a weak instance.

Definition 2.3.1 *Let $\mathcal{S} = (V_{\mathcal{S}}, E, \ell, \tau_{\mathcal{S}}, \mathsf{val}_{\mathcal{S}})$ be a semistructured instance over a set of objects $\mathcal{O}$, a set of labels $\mathcal{L}$ and a set of types $\mathcal{T}$ and let $\mathcal{W} = (V_{\mathcal{W}}, \mathsf{lch}_{\mathcal{W}}, \tau_{\mathcal{W}}, \mathsf{val}_{\mathcal{W}}, \mathsf{card})$ be a weak instance. $\mathcal{S}$ is* **compatible** *with $\mathcal{W}$ if the root of $\mathcal{S}$ is in $\mathcal{W}$ and for each o in $V_{\mathcal{S}}$:*

- *o is also in $V_{\mathcal{W}}$.*
- *If o is a leaf in $\mathcal{S}$ and also a leaf in $\mathcal{W}$, then $\tau_{\mathcal{S}}(o) = \tau_{\mathcal{W}}(o)$ and $\mathsf{val}_{\mathcal{S}}(o) \in \mathsf{dom}(\tau_{\mathcal{S}}(o))$.*
- *If o is not a leaf in $\mathcal{S}$ then*
  - *For each edge (o, o') with label l in $\mathcal{S}$, $o' \in \mathsf{lch}_{\mathcal{W}}(o, l)$,*
  - *For each label $l \in \mathcal{L}$, let $k = |\{o' | (o, o') \in E \wedge \ell(E) = l\}|$, then $card(o, l).min \leq k \leq card(o, l).max$.*

I use $Domain(\mathcal{W})$ to denote the set of all semistructured instances that are compatible with a weak instance $\mathcal{W}$. Similarly, for a probabilistic instance $\mathcal{I} = (V, \mathsf{lch}_{\mathcal{I}}, \tau_{\mathcal{I}}, \mathsf{val}_{\mathcal{I}}, \mathsf{card}, \wp)$, I use $Domain(\mathcal{I})$ to denote the set of all semistructured instances that are compatible with $\mathcal{I}$'s associated weak instance $\mathcal{W} = (V, \mathsf{lch}_{\mathcal{I}}, \tau_{\mathcal{I}}, \mathsf{val}_{\mathcal{I}}, \mathsf{card})$.

I now define a *global interpretation* based on the set of a compatible instances of a weak instance.

Definition 2.3.2 *Consider a weak instance* $\mathcal{W} = (V, \mathsf{lch}, \tau, \mathsf{val}, \mathsf{card})$*. A* **global interpretation** $\mathcal{P}$ *is a mapping from Domain*$(\mathcal{W})$ *to* $[0,1]$ *such that* $\Sigma_{S \in Domain(\mathcal{W})} \mathcal{P}(S) = 1$.

Intuitively, a global interpretation is a distribution over the set of semistructured instances compatible with a weak instance. In contrast, a local interpretation (Def. 2.2.11) defines semantics at a node by node level, rather than considering the space of all compatible semi-structured instances.

First I must impose an acyclicity requirement on the weak instance graph. This is required to ensure that my probabilistic model is coherent.

Definition 2.3.3 *Let* $\mathcal{W} = (V_{\mathcal{W}}, \mathsf{lch}_{\mathcal{W}}, \tau_{\mathcal{W}}, \mathsf{val}_{\mathcal{W}}, \mathsf{card})$ *be a weak instance.* $\mathcal{W}$ *is* **acyclic** *if its associated weak instance graph* $\mathcal{G}_{\mathcal{W}}$ *is acyclic.*

Note that I do not restrict a probabilistic instance to be trees; I allow graphs, I just do not allow cycles in the dependency structure. For example, the probabilistic instance in Figure 2.3 whose weak instance graph shown in Figure 2.1 is an acyclic graph.

Given a probabilistic instance $\mathcal{I}$ over an acyclic weak instance $\mathcal{W}$, the probability of any particular instance can be computed from the OPF and VPF entries corresponding to each object in the instance and its children.

For any probabilistic instance *under the local semantics*, we can construct an associated Bayesian network that encodes the *structure* of the semistructured instance. This Bayesian network allows us to construct a global interpretation from a local one. As we will see later, the Bayes nets only allows us to go the other way *in some cases.* This relationship with Bayesian network can be leveraged to some extent for query processing.

The Bayesian network is constructed as follows. For each object o_i in the PXML instance, there are two corresponding nodes in the Bayesian network, the node n_i and its child node c_i. Node n_i is a boolean random variable that is *true* if object o_i occurs in the semistructured instance, and *false* otherwise. For non-leaf objects o_i, node c_i is a discrete random variable whose values correspond to the *set* of children of object o_i that occurs in the semistructured instance. For leaf objects o_i, node c_i is a discrete random variable whose values correspond to the domain $\mathsf{dom}(\tau(o_i))$. The conditional probability distribution for each n_i is deterministic: n_i is *true* with probability 1 if o_i is the member of any child set that occurs; otherwise, it is *false* with probability 1. The value of the node n_i corresponding to the root is *true* with probability 1 because the root always exists. The conditional probability distribution for each c_i corresponds to the local probability models in the PXML instance (the OPF or VPF for o_i, as appropriate) if n_i is *true*; if n_i is *false*, then c_i has a special value $\emptyset$ with probability 1. Because of the acyclicity requirement for the weak instance graph, we will always be able to find an ordering of the random variables such that we have any c_j that potentially contains o_i before n_i.

Let $\mathcal{I} = (V, \mathsf{lch}_{\mathcal{I}}, \tau_{\mathcal{I}}, \mathsf{val}_{\mathcal{I}}, \mathsf{card}, \wp)$, be a probabilistic instance. Let $|V| = m$. The associated Bayesian network defines the following distribution:

$$\mathcal{P}_{BN}(n_1, \ldots, n_m, c_1, \ldots, c_m) = \prod_{i=1}^{m} P(n_i | c_1, \ldots, c_{i-1}) P(c_i | n_i)$$

For any complete assignment, the probability is 0 if any of the n_i are inconsistent with the structure encoded by the $c_1, \ldots, c_{i-1}$. For any complete assignment consistent with structure encoded by the c_i's and n_i's, the probability is simply the product of the $P(c_i|n_i)$.

With this construction in mind, I am now going to define the relationship between the local interpretation and the global interpretation for PXML.

Definition 2.3.4 *Let $\wp$ be local interpretation for a weak instance $\mathcal{W} = (V, \mathsf{lch}, \tau, \mathsf{val}, \mathsf{card})$. We define the function $\mathcal{P}_\wp$ as follows: for any instance $S \in Domain(\mathcal{W})$, $\mathcal{P}_\wp(S) = \prod_{o \in S} \wp(o)(\mathsf{c}_S(o))$, where if o is not a leaf in $\mathcal{W}$, then $\mathsf{c}_{\mathcal{S}}(o) = \{o' | (o, o') \in E\}$, i.e., the set of children of o in instance $\mathcal{S}$; otherwise, $\mathsf{c}_{\mathcal{S}}(o) = \mathsf{val}_{\mathcal{S}}(o)$, i.e., the value of o in instance $\mathcal{S}$.*

In order to use this definition of $\mathcal{P}_\wp$ for the semantics of our model, I must first show that the above function is in fact a legal global interpretation.

Theorem 2.1 *Suppose $\wp$ is a local interpretation for a weak instance $\mathcal{W} = (V, \mathsf{lch}, \tau, \mathsf{val}, \mathsf{card})$. Then $\mathcal{P}_\wp$ is a global interpretation for $\mathcal{W}$.*

Proof: The Bayesian network construction above defines a legal probability distribution denoted $\mathcal{P}_{BN}$. I show the correspondence between $\mathcal{P}_{BN}$ and $\mathcal{P}_{\wp}$.

An instance S defines an assignment to the random variables $n_1, \ldots, n_m, c_1, \ldots, c_m$ of the BN. By construction, for any assignment of $n_1, \ldots, n_m, c_1, \ldots, c_m$ that is not consistent with $S \in \mathit{Domain}(\mathcal{W})$, $\mathcal{P}_{BN}(S) = 0$. By definition, for any instance $S \in \mathit{Domain}(\mathcal{W})$,

$$\mathcal{P}_{BN}(n_1, \ldots, n_m, c_1, \ldots, c_m) = \prod_{i=1}^{m} P(n_i | c_1, \ldots, c_{i-1}) P(c_i | n_i)$$

I can collect together the terms corresponding to the $o_i \in S$ and the $o_i \notin S$ as follows:

$$\mathcal{P}_{BN}(n_1, \ldots, n_m, c_1, \ldots, c_m) = \prod_{o_i \in S} P(n_i | c_1, \ldots, c_{i-1}) P(c_i | n_i) \prod_{o_i \notin S} P(n_i | c_1, \ldots, c_{i-1}) P(c_i | n_i)$$

For each $o_i \notin S$, $n_i = \mathit{False}$ and $c_i = \emptyset$, and by the BN construction:

$$P(n_i = \mathit{False} | c_1, \ldots, c_{i-1}) P(c_i = \emptyset | n_i = \mathit{False}) = 1,$$

so I can ignore this term. For each $o_i \in S$, $n_i = \mathit{True}$ and $c_i = \mathsf{c}_{\mathcal{S}}(o)$, and by the BN construction:

$$P(n_i = \mathit{True} | c_1, \ldots, c_{i-1}) P(c_i = \mathsf{c}_{\mathcal{S}}(o) | n_i = \mathit{True}) = \wp(o)(\mathsf{c}_S(o)),$$

because the first term is 1, and the second term corresponds to the OPF or VPF for o. Thus, for the assignment of the n_i and c_i associated with S,

$$\mathcal{P}_{BN}(n_1, \ldots, n_m, c_1, \ldots, c_m) = \prod_{o \in S} \wp(o)(\mathsf{c}_S(o)) = \mathcal{P}_{\wp}(S),$$

and $\mathcal{P}_{\wp}(S)$ defines a global interpretation for $\mathcal{W}$. ∎

Example 2.3.1 *Consider S_1 in Figure 2.4 and the probabilistic semistructured instance from Figure 2.3.*

$$\begin{aligned} P(S_1) &= P(B1, B2 \mid R)\ P(A1, T1 \mid B1) P(A1, A2 \mid B2)\ P(I1 \mid A1)\ P(I1 \mid A2) \\ &= 0.2 \cdot 0.35 \cdot 0.4 \cdot 0.8 \cdot 0.5 = 0.00448 \end{aligned}$$

An important question is whether we can go the other way: from a global interpretation, can we find a local interpretation for a weak instance $\mathcal{W}(V, \mathsf{lch}, \tau, \mathsf{val}, \mathsf{card})$? It turns out that we can **if** the global interpretation can be factored in a manner consistent with the structure constraints

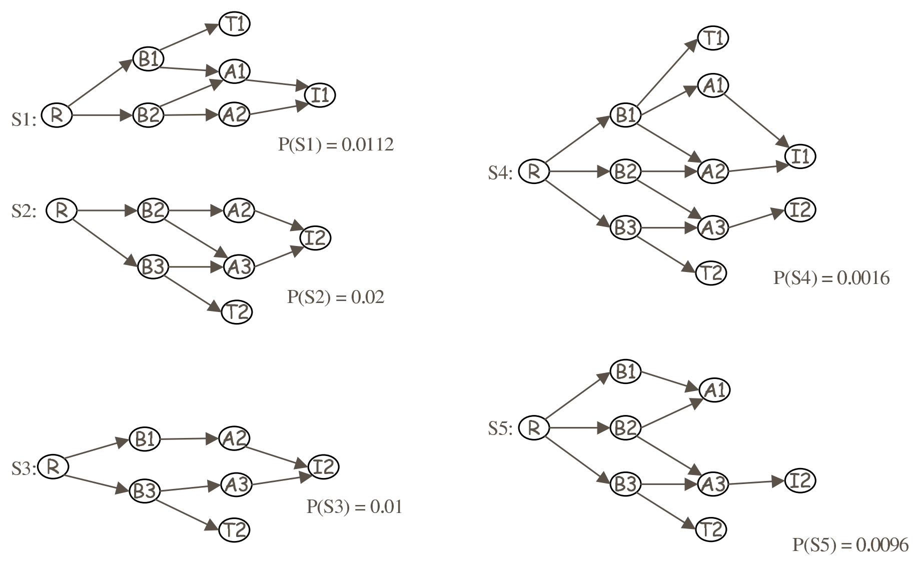


Figure 2.4: Some of the semistructured instances compatible with the probabilistic instance in Figure 2.3.

imposed by $\mathcal{W}(V, \mathsf{lch}, \tau, \mathsf{val}, \mathsf{card})$. One way to ensure this is to impose a set of independence constraints on the distribution $\mathcal{P}$.

Given $\mathcal{P}$ is a global interpretation and $\mathcal{W} = (V, \mathsf{lch}, \tau, \mathsf{val}, \mathsf{card})$ is a weak instance, we can compute the probability of any object o's children, $\mathsf{C}(o)$ given o exists, which we will denote $\mathcal{P}(\mathsf{C}(o) = c)$, directly from the global interpretation as follows:

$$\mathcal{P}(\mathsf{C}(o) = c) = \frac{\sum_{S \in Domain(\mathcal{W}) \wedge o \in S \wedge \mathsf{C}(o)=c} \mathcal{P}(S)}{\sum_{S \in Domain(\mathcal{W}) \wedge o \in S} \mathcal{P}(S)}.$$

Similarly, we can compute the probability of a set of objects. The objects that we will be interested in are the non-descendants of o in the weak instance graph for $\mathcal{W}$, which we will denote $\mathsf{non\text{-}des}_{\mathcal{W}}(o)$. The probability we will be interested in is the probability of the nondescendents given the object exists, $\mathcal{P}(\mathsf{non\text{-}des}_{\mathcal{W}}(o))$, and this is simply:

$$\mathcal{P}(\mathsf{non\text{-}des}_{\mathcal{W}}(o)) = \frac{\sum_{S \in Domain(\mathcal{W}) \wedge o \in S \wedge \mathsf{non\text{-}des}_{\mathcal{W}}(o)} \mathcal{P}(S)}{\sum_{S \in Domain(\mathcal{W}) \wedge o \in S} \mathcal{P}(S)}.$$

We can compute the probability of the children and the descendents, $\mathcal{P}(\mathsf{C}(o) = c, \mathsf{non\text{-}des}_{\mathcal{W}}(o))$, analogously.

Definition 2.3.5 *Suppose $\mathcal{P}$ is a global interpretation and $\mathcal{W} = (V, \mathsf{lch}, \tau, \mathsf{val}, \mathsf{card})$ is a weak instance. $\mathcal{P}$* **satisfies** *$\mathcal{W}$ iff for every non-leaf object $o \in V$ and each $c \in \mathsf{PC}(o)$ (and for every leaf object $o \in V$ and each $c \in \mathsf{dom}(\tau(o))$), it is the case that*

$$\mathcal{P}(\mathsf{C}(o) = c, \mathsf{non\text{-}des}_{\mathcal{W}}(o)) = \mathcal{P}(\mathsf{C}(o) = c).$$

In other words, given that o occurs in the instance, the probability of any potential children c of o is independent of the nondescendants of o in the instance. *Under this assumption*, it is possible to use the Bayesian net to construct a local interpretation from a global one. Future sections of this chapter, however, will *not* make this assumption unless explicitly stated.

Furthermore, given a global interpretation that satisfies a weak instance, we can find a local interpretation associated with it in the following manner:

Definition 2.3.6 ($\tilde{D}$ operator) *Suppose $\mathsf{c} \in \mathsf{PC}(o)$ for some non-leaf object o[5] and suppose $\mathcal{P}$ is a global interpretation. $\omega_{\mathcal{P},o}$, is defined as follows.*

$$\omega_{\mathcal{P},o}(\mathsf{c}) \quad = \quad \frac{\Sigma_{S \in Domain(\mathcal{W}) \,\wedge o \in S \,\wedge\, \mathsf{c}_S(o) = \mathsf{c}} \mathcal{P}(S)}{\Sigma_{S \in Domain(\mathcal{W}) \,\wedge o \in S} \mathcal{P}(S)}.$$

Then, $\tilde{D}(\mathcal{P})$ returns a function defined as follows: for any non-leaf object o, $\tilde{D}(\mathcal{P})(o) = \omega_{\mathcal{P},o}$.

Intuitively, we construct $\omega_{\mathcal{P},o}(\mathsf{c})$ as follows. Find all semistructured instances S that are compatible with $\mathcal{W}$ and eliminate those for which o's set of children is not c. The sum of the (normalized) probabilities assigned to the remaining semistructured instances by $\mathcal{P}$ is assigned to c by the OPF[6] $\omega_{\mathcal{P},o}(\mathsf{c})$. By doing this for each object o and each of its potential child sets, we get a local interpretation.

Theorem 2.2 *Suppose $\mathcal{P}$ is a global interpretation for a weak instance $\mathcal{W} = (V, \mathsf{lch}, \tau, \mathsf{val}, \mathsf{card})$ and $\mathcal{P}$ satisfies $\mathcal{W}$. Then $\tilde{D}(\mathcal{P})$ is a local interpretation for $\mathcal{W}$.*

Proof: From Definition 2.3.6, $\tilde{D}(\mathcal{P})(o) = \omega_{\mathcal{P},o}$ is an OPF (or VPF) for o because $\Sigma_{\mathsf{c}} \omega_{\mathcal{P},o}(\mathsf{c}) = 1$. By Definition 2.2.11, $\tilde{D}(\mathcal{P})$ is a local interpretation because for every non-leaf object o, $\tilde{D}(\mathcal{P})(o)$

[5]For leaf objects, $\mathsf{c} \in \mathsf{dom}(\tau(o))$ and $\mathsf{c}_S(o) = \mathsf{val}(o)$ in the formula.

[6]VPF for leaf objects; note that for the rest of this section, when I mention OPF, it is also true for the case of VPF.

returns an OPF for o and for every leaf object o, $\tilde{D}(\mathcal{P})(o)$ returns a VPF for o. ∎

2.4 Probabilistic Point Queries

In the next section, we turn to more complex algebraic operations over PXML that do not map directly to simple Bayesian network queries. But first, now that we have the semantics for PXML, there are a number of simple queries that are straightforward to compute and in the case where the local semantics hold, some of these can be computed directly using the Bayesian network construction from the previous section.

To begin, the simplest probability that we may wish to compute is the probability of an object o_i existing, which I will denote simply $P(o_i)$. For a given probabilistic instance $\mathcal{W}$ with global interpretation $\mathcal{P}$,

$$P(o_i) = \sum_{S \in Domain(\mathcal{W}) \wedge o \in S} \mathcal{P}(S)$$

In the case where we can represent $\mathcal{P}(S)$ using the local representation, this is equivalent to computing the query $\mathcal{P}_{BN}(n_i = 1)$. By definition, this is equal to:

$$\mathcal{P}_{BN}(n_i = 1) = \sum_{n_1,\ldots,n_{i-1},n_i,\ldots,n_m,c_1,\ldots,c_m)} \prod_{i=1}^{m} P(n_i|c_1,\ldots,c_{i-1})P(c_i|n_i).$$

We can compute this query efficiently using any of a number of Bayesian network inference techniques [66, 53, 15]. In general, if the network is tree structured, the inference will be linear in the number of nodes in the network. If the network is not a tree, the complexity depends on the connectivity of the graph and the induced tree width of the graph. In practice, if the graph is not highly connected, as in our example, the inference is quite efficient. And, regardless of the structure, the inference algorithms are significantly more efficient than naively computing the probability by marginalizing over all of the compatible instances.

It is also straight-forward to compute the probability of the existence of some collection of objects, simply by computing the marginal probability of their associated n_i's.

2.5 Probabilistic Semistructured Algebra

This section describes several more complex manipulations of PXML using algebraic operations on probabilistic instances. For convenience, I use the term *instance* to refer to *a probabilistic instance* when there is no ambiguity.

Relational algebra is based on relation names and attribute names while my algebra is based on probabilistic instance names and *path expressions*. The definition of path expressions is a variation of the standard definition [3].

Definition 2.5.1 *An* **edge sequence** *is a sequence $l_1.\dots.l_n$, where the l_i's are labels of edges. A* **path expression** $p = r.l_1.\dots.l_n$ *is an object (oid) r, followed by a (possibly empty) edge sequence $l_1.\dots.l_n$; p denotes the set of objects that can be reached from r via the sequence of edges with labels $l_1.\dots.l_n$.*

A path expression is used to locate objects in an instance. We say $o \in p$ iff there is a path p to reach o. For example, in the instance in Figure 2.1, $A2 \in R.book.author$ because there is a path from R to reach $A2$ through a path that is labeled $book.author$.

In this section I will define the following operators: projection, selection, and cross product (join can be defined in terms of these operations in the standard way). For each operator, I will first describe how it works on an ordinary semistructured instance, then I will describe how it works on a probabilistic instance.

2.5.1 Projection

I propose several projection operators including ancestor projection, descendant projection and single projection as follows. The *ancestor projection* operation extracts subgraphs composed of objects located by a path expression and those objects' ancestors up to the root. Note that only those ancestors and edges on the paths to those objects are extracted. The *descendant projection* operation extracts subgraphs composed of objects located by a given set of path expressions and those objects' descendants. The objects located by path expressions are connected to the root.

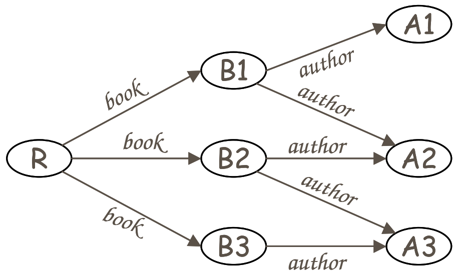


Figure 2.5: The result of the ancestor projection on the semistructured instance in Figure 2.1 with the path expressions R.*book.author*.

Finally, the *single projection* operation extracts all objects by a given path expression and then connects them to the root. I only discuss ancestor projection - the other two notions of projection can be similarly constructed.

Example 2.5.1 *Consider the semistructured instance shown earlier in Figure 2.1. Suppose we have a path expression R.book.author. Ancestor projection will first locate the set $V' = \{A1, A2, A3\}$ of objects that satisfy the path expression. V' is expanded by adding objects on the path from the root to the objects in V' (the added objects are B1, B2 and B3), as well as the root of the instance (R). V' is the set of the objects in our new instance. If there was an edge between two nodes n_1, n_2 in the semistructured instance of Figure 2.1 and $n_1, n_2 \in V'$ then we draw an edge from n_1 to n_2 with the same label. The resulting instance is shown in Figure 2.5.*

Definition 2.5.2 *[ancestor projection (Λ)] Suppose $G = (V, E, \ell)$ is an instance, r is the root of G and p is a path expression. The* ancestor-projection *of G on p, denoted $\Lambda_p(G) = (V', E')$, is defined as follows:*

- $V' = \{o \mid o \in V \wedge (o \in p \vee \exists o' \in V,$ *edge sequences* s, s' $(p = r.s.s' \wedge o \in r.s \wedge o' \in o.s'))\} \cup \{r\}$
- $E' = \{(o, o') \mid (o, o') \in E \wedge o, o' \in V' \wedge \exists$ *edge sequences* s, s' *a label* l *and an object* $o'' \in V'$ $(p = r.s.l.s' \wedge o \in r.s \wedge o' \in o.l \wedge o'' \in o'.s')\}$
- $\forall (a, b) \in E'$ $(\ell'(a, b) = \ell(a, b))$

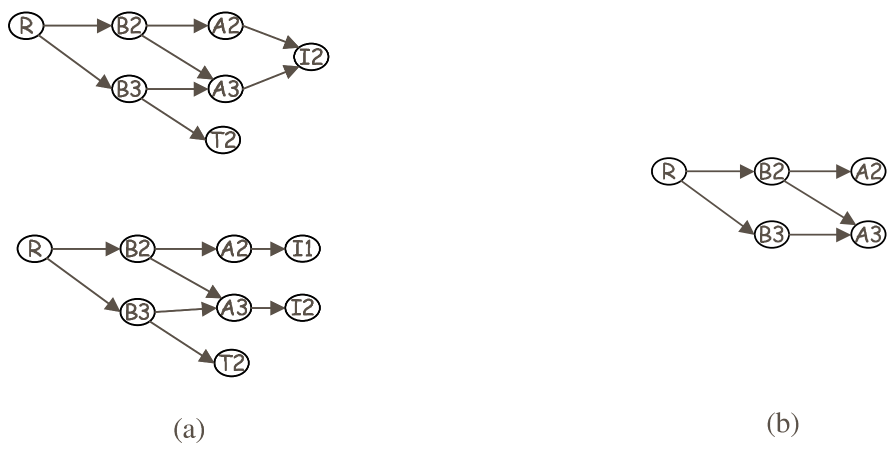


Figure 2.6: Given the two instances S1 and S2 on the left, the ancestor projection of R.*book.author* gives the same resulting semistructured instance shown as S3 on the right. Because these are the only two compatible instances that produce this result, the probability of the result is simply the sum of the two probabilities, $P(S3) = P(S1) + P(S2)$.

- *For any newly created leaf* o', $\mathsf{val}(o') = NULL$.

We have seen how ancestor projection works on a semistructured instance. Now, we are going to see what it means for a probabilistic instance. For example, recall the first question we wanted to answer in Section 2.1.1. We can use an ancestor projection with a path expression *R.book.author* on the probabilistic instance. The result keeps the authors and their ancestors, which can be used to deduce the global probabilities of compatible instances or the probability of a particular author in the future. Recall that from a probabilistic instance, we can obtain a set of compatible instances and a distribution over the probability of each of the compatible instances. We can perform the ancestor projection on each of the compatible instances to obtain a resulting set of semistructured instances. We then combine the probabilities of identical instances by summing them up.

For example, after the ancestor projection with a path expression *R.book.author* on the set of instances S1 and S2 in Figure 2.6(a), we will have $S3$ as the result, shown in Figure 2.6(b).

We can combine the probabilities of $S1$, $S2$, i.e., $P(S1) + P(S2)$, which is the probability of the resulting instance.

Definition 2.5.3 *Suppose $\mathcal{I} = (V, \mathsf{lch}, \tau, \mathsf{val}, \mathsf{card}, \wp)$ is a probabilistic instance, $\mathcal{P} = \tilde{W}(\wp)$ and p is a path expression. The probabilities of the result of the ancestor projection with path expression p on $\mathcal{I}$ are defined as follows: for every $\mathcal{S} \in Domain(\Lambda_p(\mathcal{I}))$, the probability of $\mathcal{S}$ is $\sum_{\mathcal{S}'' \in Domain(\mathcal{I}) \text{ s.t. } \Lambda_p(\mathcal{S}'')=\mathcal{S}} \mathcal{P}(\mathcal{S}'')$.*

I give an efficient algorithm when all compatible instances are tree-structured. Note that this is not a requirement, but it does simplify the algorithm.

Ancestor projection on a probabilistic instance results in a new probabilistic instance, with the probability of an instance S' in the projection computed as the sum of the probabilities of the instances that map to S' in the original probabilistic instance. We can treat the probabilistic instance as an ordinary semistructured instance and perform ancestor projection on it and update card and $\wp$. We begin at the leaves. The value of a newly created leaf is NULL, with probability 1. The update of $\wp$ and card is done starting from the immediate parents of leaves. The update is bottom up; it will be performed on an object only if the updates have been done on all of its children.

Let o_i denote the non-leaf object whose $\wp(o_i)$ and card are to be updated. I denote the original set of children before projection as $\mathsf{C}(o_i)$, the new set of children after projection as $\mathsf{C}'(o_i)$, and $C_d = \mathsf{C}(o_i) - \mathsf{C}'(o_i)$. Similarly, we use $\wp'$ and card' to denote the new local interpretation and cardinality.

- First, consider the immediate parent of a leaf.
  - **Marginalization.** Intuitively, for each $c' \subseteq \mathsf{C}'(o_i)$, we project all the children in the original, $c \subseteq \mathsf{C}(o_i)$, where c' is the result of projection of c (after removing the deleted children), to c':

$$\wp'(o_i)(c') = \sum_{d \subseteq C_d \text{ s.t. } c=(c' \cup d) \in \mathsf{PC}(o_i)} \wp(o_i)(c' \cup d).$$

- **Normalization.**

  A non-leaf object (except the root) in the result of ancestor projection should not exist in a compatible instance if none of its children exists in the compatible instance (by the definition of ancestor projection). We will compute ϵ_{o_i}, the probability that o_i has some child still existing in the result of the ancestor projection:

$$\epsilon_{o_i} = \sum_{c' \in \mathsf{PC}'(o_i) \wedge c' \neq \emptyset} \wp'(o_i)(c').$$

  We then renormalize the probabilities so that $\wp'(o_i)(c)$ will represent the conditional probability of o_i having children c given the condition that some of the children exist. We set $\wp'(o_i)(\{\}) = 0$ and do the normalization as follows: $\forall c \subseteq \mathsf{C}'(o_i)$,

$$\wp'(o_i)(c) = \frac{\wp'(o_i)(c)}{\epsilon_{o_i}}$$

- For other non-leaf object (except the root), for each $c' \in \mathsf{C}'(o_i)$, we project all the children in the original $c \in \mathsf{C}(o_i)$, where c' is a subset of c, to c', and multiply by the probability that each exists:

$$\wp'(o_i)(c') \quad = \sum_{c \in \mathsf{PC}(o_i) \wedge c' \subseteq c} \wp(o_i)(c) \prod_{o_j \in c'} \epsilon_{o_j} \prod_{o_j \in (c - c') \wedge o_j \in \mathsf{PC}'(o_i)} (1 - \epsilon_{o_j}).$$

  As, above, we will record the probability ϵ_{o_i}, set $\wp'(o_i)(\{\}) = 0$ and renormalize the probabilities by dividing them by ϵ_{o_i}.

- For the root r, we marginalize as above. However, we do not need to set $\wp'(r)(\{\})$ to 0 and do normalization. In essence, $\wp'(r)(\{\})$ is the probability that a compatible instance in the original has no object satisfying the path expression of the ancestor projection and, as a result, only the root object is returned.

The process of update of card is the same for all non-leaf objects: for an object o_0 and an edge label l_j,

$$\mathsf{card}'(o_0, l_j).min = \min_{C \subseteq \mathsf{C}'(o_0) \text{ s.t. } \wp'(o_0)(C) > 0} (\text{number of objects in } C \text{ that have edge label } l_j)$$

$$\mathsf{card}'(o_0, l_j).max = \max_{C \subseteq \mathsf{C}'(o_0) \text{ s.t. } \wp'(o_0)(C) > 0} (\text{number of objects in } C \text{ that have edge label } l_j)$$

Probabilistic Path Queries

Now let us consider computing the probability of a simple object chain. To compute the probability of a simple object chain $c = r.o_1.o_2.\ldots.o_i$, we consider all possible ways that the chain can be achieved:

$$P(c) \;=\; p(r) \sum_{c_1 \in \mathsf{PC}(r) \wedge o_1 \in c_1} p(c_1) \times \sum_{c_2 \in \mathsf{PC}(c_1) \wedge o_2 \in c_2} p(c_2) \times \cdots \times \sum_{c_i \in \mathsf{PC}(c_{i-1}) \wedge o_i \in c_i} p(c_i)$$

Next we consider *probabilistic path queries*, which allow us to compute the probability that an object satisfies a path expression. This kind of query can be used to answer the last situation in Section 2.1.1: we want to know the probability that a particular author exists.

Definition 2.5.4 *Given a path expression p and an object o in a probabilistic instance, a probabilistic path query returns the probability that $o \in p$ in a compatible instance.*

Here I assume that $o \in p$ in the probabilistic instance, otherwise it is obvious that the probability must be zero. First, I define the *path ancestors* of o as all o's ancestors such that for every such ancestor o_a, there exists a path identical to the path expression p from the root to o_a and then to o. Note that if I extract only the object o and its path ancestors from the probabilistic instance, and use the same method described in the previous section to calculate ϵ_r, then ϵ_r will be the answer to this problem. The reason is that the root of the result of the ancestor projection on a compatible instance will have a child if and only if there is some object in that compatible instance satisfying the path expression. Here, because I only keep o and its path ancestors, the root of the result of the ancestor projection on a compatible instance will have a child if and only if o in that compatible instance satisfies the path expression p. Recall the meaning of ϵ_r is that, given that r exists in a compatible instance (which is true always), r still has a child that should exist after the ancestor projection on that compatible instance, so ϵ_r also gives the probability that o satisfies p.

An extension to this problem is to find the probability that there exists some object satisfying a given path expression. We can solve it by keeping all objects satisfying the path expression in the probabilistic instance and their path ancestors and calculate ϵ_r as the answer.

2.5.2 Selection

In this section I will describe the selection operation. I define two types of selection conditions, an *object selection condition* and a *value selection condition.*

Definition 2.5.5 (object selection condition) *An* object selection condition *is of the form $p \in o$ or $p \notin o$ where p is a path expression starting from the root and o is an object id.*

Definition 2.5.6 (value selection condition) *A* value selection condition *is of the form* $\mathsf{val}(p)$ ϕ v *where p is a path expression starting from the root to some leaf, ϕ is a binary predicate from* $\{=, \neq, \leq, \geq, <, >\}$ *and v is a value.*

It is straightforward to add other kinds of selection conditions (e.g. those based on cardinality or OPFs/VPFs) - space constraints preclude us from doing so.

With a given selection condition sc and a probabilistic instance $\mathcal{I}$, the global approach will give a set of semistructured instances (with normalized probabilities) compatible with the result of selection operation. The global approach works as follows: among the set of instances compatible with the probabilistic instance $\mathcal{I}$, only those instances satisfying the selection condition sc will be selected; then their resulting probabilities will be obtained by normalizing their original probabilities using the formula in the following definition.

Definition 2.5.7 (selection (σ)) *Suppose $\mathcal{I} = (V, \mathsf{lch}, \tau, \mathsf{val}, \mathsf{card}, \wp)$ is a probabilistic instance, sc is a selection condition. Let $Domain(\sigma_{sc}(\mathcal{I})) = \{\mathcal{S} \in Domain(\mathcal{I}) \mid \mathcal{S}$ satisfies $sc\}$ be the set of compatible instances satisfying the selection condition. Then, $\forall \mathcal{S} \in Domain(\sigma_{sc}(\mathcal{I}))$, $\mathcal{P}'(\mathcal{S}) = \frac{\mathcal{P}(\mathcal{S})}{\sum_{\mathcal{S}' \in Domain(\sigma_{sc}(\mathcal{I}))} \mathcal{P}(\mathcal{S}')}$*

Example 2.5.2 *Suppose we have a (simplified) probabilistic instance with the $\mathcal{P}$ shown in Figure 2.7(a). Recall the second situation in Section 2.1.1 where the book $B1$ surely exists. So,*

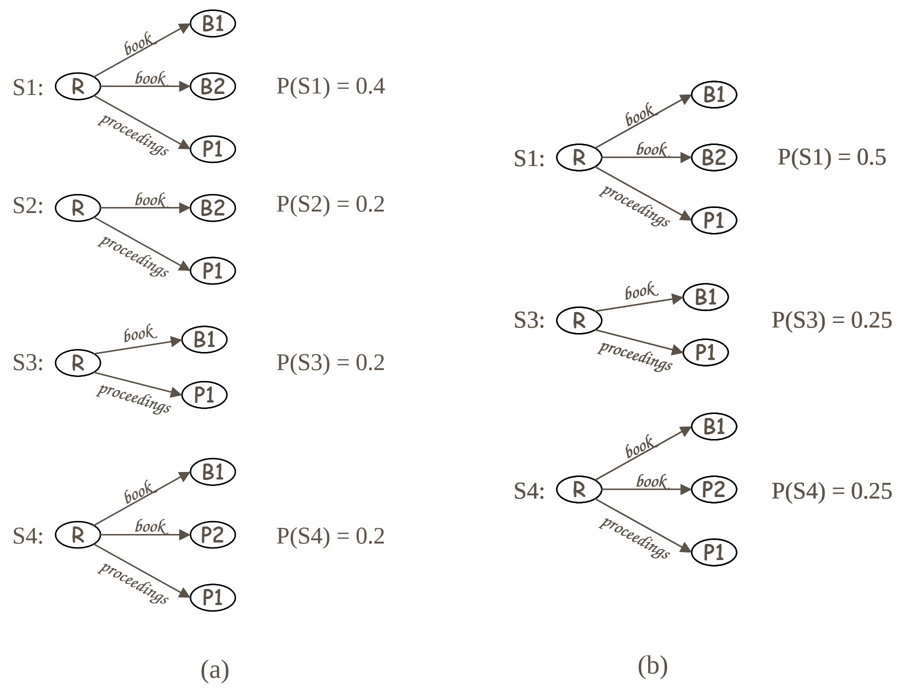


Figure 2.7: (a) The set of compatible instances of a probabilistic instance along with their probabilities. (b) The result of the selection R.*book* = *B1*.

how will the probabilities be affected? The result of the selection R.book = B1 is shown in Figure 2.7(b). The set of compatible instances are shown, along with their updated probabilitites. Among the four compatible instances shown in Figure 2.7(a), only $S1, S3, S4$ *satisfy the selection condition. Then we normalize the probabilities of the selected instances as follows. For example, consider* $S1$*:* $\mathcal{P}'(S1) = \frac{\mathcal{P}(S1)}{\mathcal{P}(S1)+\mathcal{P}(S3)+\mathcal{P}(S4)} = \frac{0.4}{0.4+0.2+0.2} = 0.5$

Here we are not making any assumptions about the factorization of the global interpretation.

2.5.3 Cartesian Product

In Section 2.1.1, I mentioned a situation where we want to combine two probabilistic instances into one. As in the case of Cartesian product in the relational algebra, I assume that the object ids are unique (after renaming, if necessary).

Definition 2.5.8 (Cartesian product ($\times$)) *Suppose* $\mathcal{I} = (V, \mathsf{lch}, \tau, \mathsf{val}, \mathsf{card}, \wp), \mathcal{I}' = (V', \mathsf{lch}', \tau', \mathsf{val}', \mathsf{card}', \wp')$ *are two probabilistic instances,* r, r' *are the roots of* $\mathcal{I}, \mathcal{I}'$*. The* Cartesian product *of*

$\mathcal{I}, \mathcal{I}'$, denoted $\mathcal{I} \times \mathcal{I}'$, results in a new probabilistic instance $\mathcal{I}'' = (V'', \mathsf{lch}'', \tau'', \mathsf{val}'', \mathsf{card}'', \wp'')$. $\mathcal{I}''$ is rooted at r'' and is defined as follows:

- $V'' = (V \cup V' \cup \{r''\}) - \{r\} - \{r'\}$, $\tau'' = \tau \cup \tau'$, $\mathsf{val}'' = \mathsf{val} \cup \mathsf{val}'$.
- $\mathsf{lch}'' = \mathsf{lch} \cup \mathsf{lch}'$ *and* $\mathsf{card}'' = \mathsf{card} \cup \mathsf{card}'$ *with the modification such that the two old roots are merged into r'' with all children of r, r' become the children of r''.*
- *for every label l,* $\forall b \in \mathsf{lch}''(a, l)$
  - *if $a = r''$, then if $b \in V$, then $\ell''(a, b) = \ell(r, b)$; otherwise, $\ell''(a, b) = \ell'(r', b)$;*
  - *if $a \neq r''$, then if $b \in V$, then $\ell''(a, b) = \ell(a, b)$; otherwise, $(\ell''(a, b) \leftarrow \ell'(a, b))$.*
- *$\wp''$ is defined such that $\forall o \in V$, $\wp''(o) = \wp(o)$; $\forall o \in V'$, $\wp''(o) = \wp'(o)$. The root r'' requires a special treatment: $\forall \mathsf{c}'' \in \mathsf{PC}(r'')$ such that $c'' = c \cup c'$ where $\mathsf{c} \in \mathsf{PC}(r), \mathsf{c}' \in \mathsf{PC}(r')$, $\wp''(r'')(\mathsf{c}'') = \wp(r)(\mathsf{c}) \times \wp'(r')(\mathsf{c}')$.*

Note that the last item in the above definition uses an independence assumption to multiply probabilities. Lakshmanan et. al. [50] introduced the concept of a *conjunction strategy* to compute the probability of a conjunction of events from the probabilities of the individual events. The multiplication in the last step of the above definition can be replaced by any of their conjunctive strategies if the person posing the query believes that strategy to be appropriate (e.g. if he knows that the events in question are positively correlated).

The standard condition join operator is defined in the usual way as a cartesian product followed by a selection.

2.6 Probabilistic Aggregate Operators

In this section, I consider another useful class of PXML operations, operations that use aggregates. I will use as a running example the surveillance application introduced earlier. Consider the probabilistic instance and its associated weak instance graph shown in Figure 2.8. The primary goal of this section is to define the declarative semantics of aggregate queries. Answering aggregate queries in PXML raises three important issues:

o	l	$\mathsf{lch}(o, l)$
I1	convoy	{ convoy1, convoy2 }
convoy1	tank	{ tank1, tank2 }
convoy2	truck	{ truck1 }

o	$\tau(o)$	$\mathsf{val}(o)$
tank1	tank-type	T-80
tank2	tank-type	T-72
truck1	truck-type	rover

o	l	$\mathsf{card}(o, l)$
I1	convoy	[1,2]
convoy1	tank	[1,1]
convoy2	truck	[1,1]

$\mathsf{c} \in \mathsf{PC}(I1)$	$\wp(I1)(\mathsf{c})$
{ convoy1}	0.3
{ convoy2}	0.2
{ convoy1, convoy2}	0.5

$\mathsf{c} \in \mathsf{PC}(convoy1)$	$\wp(convoy1)(\mathsf{c})$
{ tank1}	0.4
{ tank2}	0.6

$\mathsf{c} \in \mathsf{PC}(convoy2)$	$\wp(convoy2)(\mathsf{c})$
{ truck1}	1

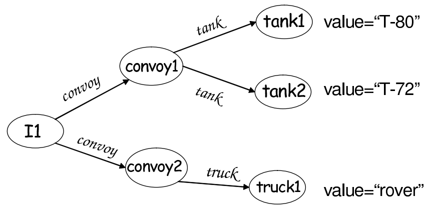


Figure 2.8: A probabilistic instance for the surveillance domain.

- **Possible-worlds answer:** Consider a query that wishes to count the number of objects in all convoys in probabilistic instance $I1$. This probabilistic instance has five compatible semi-structured instances marked as $S1, \ldots, S5$ in Figure 2.9. Each of these instances has between 1 and 2 objects - we could return the set $\{1, 2\}$ indicating that the answer to the count query is not known precisely, but is either 1 or 2 (each with probability 0.5 in this example).

- **Expected answer:** Alternatively, we could use the statistical notion of *expected value*. In this case, we always return one count for any count query. We multiply the number of objects in $S1$ (i.e. 2) by the probability of $S1$ (i.e. $0.5 \times 0.4 \times 1 = 0.2$) and add this to the number of objects in $S2$ (i.e. 2) by the probability of $S2$ (i.ee. $0.5 \times 0.6 \times 1 = 0.3$) and so on. In the above example, we would return 1.5 as the expected value.

- **Form of the answer:** Instead of just giving a "direct" answer to an aggregate query such as 1.5, we may want to return a probabilistic instance. The advantage of returning a probabilistic instance as output is that this can be the subject of further querying.

In the rest of this section, I proceed as follows. As the answers in both semantics above depend upon finding the answer to an aggregate query in a semistructured instance, I first give a formal definition of aggregates for semistructured instances (Section 2.6.1), and then extend it to the case of the possible world aggregates (Section 2.6.2) and then to the case of expected aggregates (Section 2.6.3).

2.6.1 Aggregates on semistructured instances

The standard aggregate functions such as sum, count, avg, min, max take a set of values and return a single value. In addition, as we will see shortly, we can define a more general notion of aggregate function that takes a multiset of values and returns a set of values.

In classical relational databases, we may count all the tuples that satisfy a given selection condition (e.g. find the number of people making over 100K in an employee database) or sum up values in a given attribute (e.g. find the sum of monthly salaries of employees in the Sales department). Such aggregates have two parts: (i) a condition such as the 100K condition or the Sales department condition mentioned above, and (ii) an aggregate function to be applied. In PXML, the analog of (i) is a path condition (or path expression), while (ii) is unchanged. In addition, PXML supports a higher level of aggregates (e.g. return the average number of tanks per convoy). To achieve this, we need to specify a path expression ($I.convoy$) specifying that we are interested in each convoy and $I.convoy.tank$. In this case a *set* of path expressions must be specified.

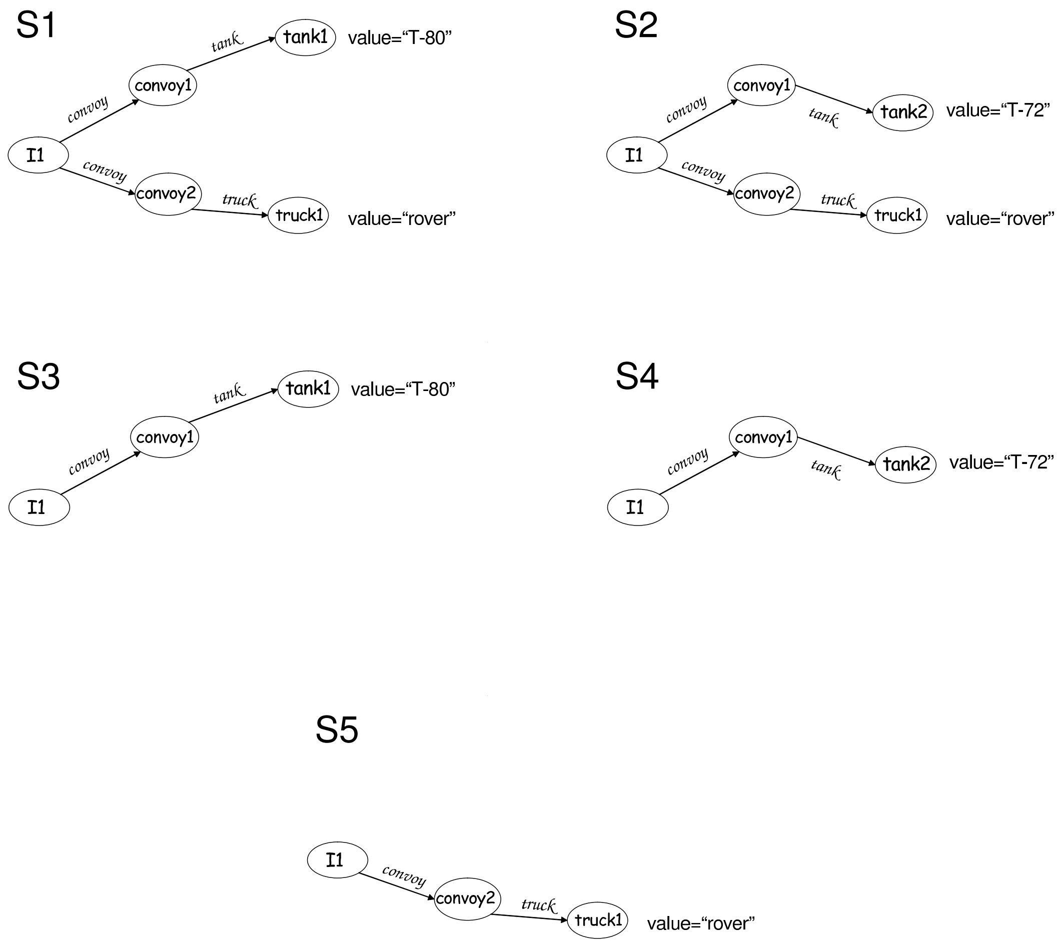


Figure 2.9: The set of semistructured instances compatible with the probabilistic instance in Figure 2.8.

Definition 2.6.1 *Suppose $\mathcal{S}$ is a semistructured instance, f is an aggregate function and α is a set of paths. Let A be the multiset of values of objects in $\mathcal{S}$ that are selected by a path in α. The aggregate operator f° returns the semistructured instance $\mathcal{S}' = (V', E', \ell', \tau', \mathsf{val}')$ where*

- *$V' = \{r', agg_{f(A)}\}$ where r' is the root of $\mathcal{S}'$ and $agg_{f(A)}$ is a new object with value $f(A)$,*
- *$E = \{(r', agg_{f(A)})\}$ with a label f.*

In this context, the simple aggregates mentioned earlier (sum, count, avg, min, max) select a multiset of objects using the path conditions. These values are then mapped to another value by f. The resulting semistructured instance is a graph with an artificial root node r'. The root has a child

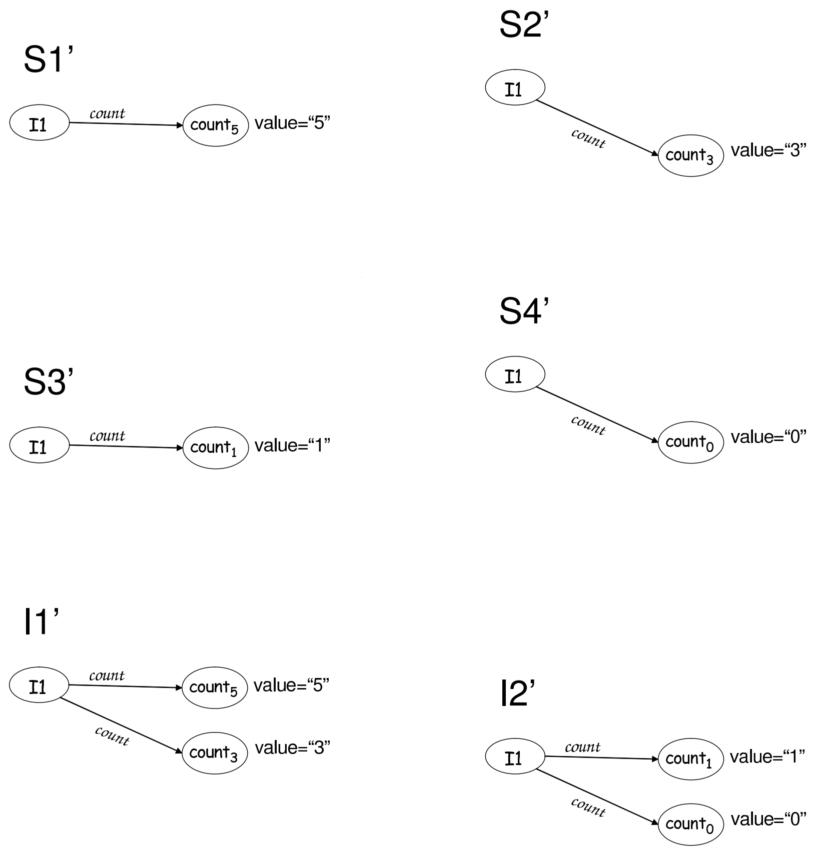


Figure 2.10: Consider the probabilistic instance $I1$ and its five compatible instances in Figure 2.9. $S1' = \mathsf{count}^{\circ}(S, \emptyset)$ where $S = S1, S2$. $S2' = \mathsf{count}^{\circ}(S, \emptyset)$ where $S = S3, S4, S5$. $I1' = \mathsf{count}^{\mathsf{P}}(I1, \emptyset)$. $S3' = \mathsf{count}^{\circ}(S, \{I1.convoy.tank\})$ where $S = S1, S2, S3, S4$. $S4' = \mathsf{count}^{\circ}(S5, \{I1.convoy.tank\})$. $I2' = \mathsf{count}^{\mathsf{P}}(I1, \{I1.convoy.tank\})$.

node labeled $agg_{f(A')}$. The edge is labeled with the name of the function f.

Example 2.6.1 *Consider the semistructured instance $S1$ in Figure 2.9 and consider the aggregate function* count. *The aggregate query* $\mathsf{count}^{\circ}(S1, \emptyset)$ *may be used to determine the number of objects in $S1$. This query returns a semistructured instance with the root connected to an object $count_5$ with the value* 5 *(shown as $S1'$ in Figure 2.10). The aggregate query* $\mathsf{count}^{\circ}(S1, \{I1.convoy.tank\})$ *can be used to determine the total number of tanks in $S1$. This returns the answer with object $count_1$ with the value* 1 *(shown as $S3'$ in Figure 2.10).*

2.6.2 Possible-worlds aggregates on probabilistic instances

Given a probabilistic instance $\mathcal{I}$, the set of semi-structured instances that are compatible with $\mathcal{I}$ represents "possible worlds." An aggregate operator has a given value in each such compatible instance. Suppose the values of a given aggregate in compatible instances $w_1, \ldots, w_k$ are $v_1, \ldots, v_k$ and suppose the probability of each compatible instance w_i is p_i. Then the probability of the aggregate value being v is given by $\Sigma_{v_j=v} p_j$. In other words, given v, we find all compatible instances w_j where $v_j = v$ and add up the probabilities of all such compatible instances.

At this point, we are still left with the problem of how to present the answer - we want the answer to be a probabilistic instance. The following definition describes how to accomplish this.

Definition 2.6.2 *Suppose* f° *is an aggregate operator. The* probabilistic aggregate operator $\mathsf{f}^{\mathsf{P}}(\mathcal{I}, \alpha)$ *based on* f° *takes a probabilistic semistructured instance* $\mathcal{I} = (V, \mathsf{lch}, \tau, \mathsf{val}, \mathsf{card}, \wp)$ *and a set* α *of paths as input, and returns as output, a probabilistic instance* $\mathcal{I}' = (V', \mathsf{lch}', \tau', \mathsf{val}', \mathsf{card}', \wp')$ *where:*

1. $V' = V'' \cup \{r'\}$ *where* r' *is the root of* $\mathcal{I}'$ *and* $V'' = \cup_{\mathcal{S} \in Domain(\mathcal{I})} (\mathcal{S}'.f)$ *where* $\mathcal{S}' = \mathsf{f}^{\circ}(\mathcal{S}, \alpha)$.
2. lch' *is given by* $\mathsf{lch}'(r', f) = V''$.
3. $\forall o \in V''$, $\tau(o)$ *is the type returned by* f°.
4. $\mathsf{card}'(r', f) = [1, 1]$.
5. *It follows immediately from the above that* $Domain(\mathcal{I}') \equiv \{\mathcal{S}' \mid \exists \mathcal{S} \in Domain(\mathcal{I})(\mathcal{S}' = \mathsf{f}^{\circ}(\mathcal{S}, \alpha))\}$. *I define the local interpretation as follows:*

   $\forall o \in V''$, $\wp'(r')(\{o\}) = \sum_{\mathcal{S} \in Domain(\mathcal{I}) \wedge \mathsf{val}(\mathcal{S}'.f) = \mathsf{val}(o)} \mathcal{P}_{\wp}(\mathcal{S})$

   $= \sum_{\mathcal{S} \in Domain(\mathcal{I}) \wedge o \in \mathsf{f}^{\circ}(\mathcal{S}, \alpha)} \mathcal{P}_{\wp}(\mathcal{S})$, *where* $\mathcal{S}' = \mathsf{f}^{\circ}(\mathcal{S}, \alpha)$.

   Equivalently, $\forall \mathcal{S}' \in Domain(\mathcal{I}')$, $\mathcal{P}'_{\wp}(\mathcal{S}') = \sum_{\mathcal{S} \in Domain(\mathcal{I}) \wedge \mathsf{f}^{\circ}(\mathcal{S}, \alpha) = \mathcal{S}'} \mathcal{P}_{\wp}(\mathcal{S})$. *Obviously,* $\forall o \in V''$, $\wp'(r')(\{o\}) \equiv \mathcal{P}_{\wp}(\mathcal{S}')$ *where* $\mathcal{S}' \in Domain(\mathcal{I}')$ *and* $o \in \mathcal{S}'$. *Similarly,* $\forall \mathcal{S}' \in Domain(\mathcal{I}')$, $\mathcal{P}_{\wp}(\mathcal{S}') \equiv \wp'(r')(\{o\})$ *where* r' *is the root of* $\mathcal{I}'$ *as well as* $\mathcal{S}'$ *and* o *is the child of* r' *in* $\mathcal{S}'$.

The following proposition provides an important commutativity result. Suppose $\mathcal{I}$ is a probabilistic instance and $\mathcal{I}'$ is the probabilistic instance that results by computing a probabilistic aggregate. If we were to perform the same aggregate on the set of compatible instances of $\mathcal{I}$, then we would get the set of compatible instances of $\mathcal{I}'$. This very neat result is a kind of "correctness" theorem that says that $\mathcal{I}'$ succinctly represents the answer.

Proposition 1 *Suppose* f^{o} *is an aggregate operator,* $\mathcal{I}$ *is a probabilistic semistructured instance,* α *is a set of path expressions, and* $\mathcal{I}' = \mathsf{f}^{\mathsf{P}}(\mathcal{I}, \alpha)$*. Then,* $Domain(\mathcal{I}') \equiv \{\mathcal{S}' \mid \exists \mathcal{S} \in Domain(\mathcal{I})(\mathcal{S}' = \mathsf{f}^{\mathsf{o}}(\mathcal{S}, \alpha))\}$*.*

Proof: Immediately follows from the construction in Definition 2.6.2. ∎

Example 2.6.2 *Consider the two aggregate operators defined in Example 2.6.1, the probabilistic instance* $I1$ *in Figure 2.8 and the compatible instances shown in Figure 2.9. The corresponding probabilistic aggregate operators work as follows.* $\mathsf{count}^{\mathsf{P}}(I1, \emptyset)$ *returns* $I1'$ *in Figure 2.10 where* $\wp'(I1)(\{count_5\}) = \mathcal{P}_\wp(S1) + \mathcal{P}_\wp(S2) = 0.2 + 0.3 = 0.5$ *and* $\wp'(I1)(\{count_3\}) = \mathcal{P}_\wp(S3) + \mathcal{P}_\wp(S4) + \mathcal{P}_\wp(S5) = 0.12 + 0.18 + 0.2 = 0.5$ *(because object* $count_5$ *results from* $S1, S2$ *and object* $count_3$ *results from* $S3, S4, S5$*). Note that* $Domain(I1') = \{S1', S2'\}$*. Similarly,* $\mathsf{count}^{\mathsf{P}}(I1, \{I1.convoy.tank\})$ *returns* $I2'$ *in Figure 2.10 where* $\wp'(I1)(\{count_1\}) = \mathcal{P}_\wp(S1) + \mathcal{P}_\wp(S2) + \mathcal{P}_\wp(S3) + \mathcal{P}_\wp(S4) = 0.2 + 0.3 + 0.12 + 0.18 = 0.8$ *and* $\wp'(I1)(\{count_0\}) = \mathcal{P}_\wp(S5) = 0.2$ *(because object* $count_1$ *results from* $S1, S2$*,* $S3$*,* $S4$ *and object* $count_0$ *results from* $S5$*). Note that* $Domain(I2') = \{S3', S4'\}$*.*

2.6.3 Expected aggregates on probabilistic instances

I now move on to the next kind of aggregate: expected value aggregates. Such an aggregate has no uncertainty. It just returns a single value.

Suppose $\mathcal{I}$ is a probabilistic instance, and suppose the values V of a given aggregate f in compatible instances $w_1, \ldots, w_k$ are $v_1, \ldots, v_k$ and suppose the probability of each compatible instance w_i is p_i. Then, following the classical statistical notion of expected value, the answer returned by our "expected value" aggregate semantics is

$$E(V) = \Sigma_{i=1}^{k} p_i \times v_i.$$

Formally, I define this as follows.

Definition 2.6.3 *Suppose* f° *is an* aggregate operator. *The expected aggregate function* $\mathsf{f}^{\mathsf{E}}(\mathcal{I}, \alpha)$ *takes a probabilistic semistructured instance* $\mathcal{I} = (V, \mathsf{lch}, \tau, \mathsf{val}, \mathsf{card}, \wp)$ *and a path input set* α*, and returns the expected value of the aggregate:* $\sum_{\mathcal{S} \in Domain(\mathcal{I})} \mathcal{P}_{\wp}(\mathcal{S}) \times v$.

2.7 Probabilistic Aggregate Algorithms

In the previous section, I gave very general definitions of aggregate operators, probabilistic aggregate operators and expected aggregate functions. The last two operations examine the entire set of compatible instances. However, this is very expensive in practice as the set of compatible instances may be huge. This section shows that for a large collection of aggregates that have the properties of *idempotence* and *distributivity*, we can avoid this.

Definition 2.7.1 *Let* X *be any set. A mapping* $f : M(X) \to M(X)$ *is idempotent-distributive (ID for short) iff*

- $\forall$ *multiset* $A \in M(X)$, $f(f(A)) \equiv f(A)$.
- $\forall$ *multisets* $A, B \in M(X)$, $f(A \cup B) \equiv f(f(A) \cup f(b))$.

Examples of ID aggregates include $\mathsf{sum}, \mathsf{count}, \mathsf{avg}, \mathsf{max}, \mathsf{min}$.[7]

Examples of operators that are not ID include medians and modes.

2.7.1 SP Algorithm

We now describe the SP algorithm that can be used to compute aggregates efficiently as long as the aggregate function has the ID property. Figure 2.11 shows the details of the algorithm.

The basic algorithm can be summarized as follows.

- $\mathsf{agg}^{\mathsf{P}}(\mathcal{I}, \alpha)$
  1. select set Y of objects and their ancestors from $\mathcal{I}$ according to the definition of f^{P} and input α;

[7]Strictly speaking, count is not ID. However, count can be computed easily by a trick in which we proceed as though we are computing sum, but replace the values being summed by 1.

```
probabilistic aggregate operator: $\mathsf{f}^{\mathsf{P}}$
        /* input: $\mathcal{I}, \alpha$; output: $\mathcal{I}'$ */
1)      locate a set $Y$ of objects from $\mathcal{I}$ according to the definition of $\mathsf{f}^{\mathsf{P}}$ and input $\alpha$;
2)      $K \leftarrow$ all elements in $Y$ and their ancestors;
3)      $T \leftarrow$ bottom-up($r$) where $r$ is the root;
4)      if $\mathsf{f}^{\mathsf{P}} = \mathsf{count}^{\mathsf{P}}$, then add a tuple $(\{0\}, u)$ into $T$ where $u = 1 - \sum_{p \in T.P} p$;
5)      else, add a tuple $(\text{undefined}, u)$ into $T$ where $u = 1 - \sum_{p \in T.P} p$;
6)      construct a probabilistic instance $\mathcal{I}'$ from $T$ as follows:
            for every $a \in T.A$, connect the root $r'$ to a new object with object id $f_a$ and value $a$
                with an edge with label $f$; $\wp(r')(\{f_a\}) = \sigma_{(T.A=a)}T.P$;
7)      return $\mathcal{I}'$;

function bottom-up
        /* input: object $o$; output: relational table $T = (A, P)$ */
1)      if $o$ is a leaf, then
2)          if $\tau(o)$ is compatible with f then $T \leftarrow \{(\{\mathsf{val}(o)\}, 1)\}$;
3)          else $T \leftarrow$ undefined;
4)          return $T$;
5)      let $C \leftarrow \{o_1, \ldots, o_n\}$ denote the children of $o$ such that $T_i \leftarrow$ bottom-up($o_i$) is not undefined;
6)      let $D \leftarrow \mathsf{C}(o) - C$;
7)      for every non-empty subset $E = \{o_{e_1}, \ldots, o_{e_m}\}$ of $C$,
8)          $p \leftarrow \sum_{F \subseteq D} \wp(o)(E \cup F)$;
9)          if $p > 0$, then
10              for each combination of values $a_{e_1}, \ldots, a_{e_m}$ where $a_{e_i} \in T_{e_i}.A$,
11)                 $a \leftarrow \mathsf{f}(\cup_{i=1}^{m} a_{e_i})$;
12)                 if $o \in Y$, then $a \leftarrow \mathsf{f}(a \cup a_{\mathsf{default}})$;
13)                 $p \leftarrow p \times \prod_{i=1}^{m} \sigma_{(T_{e_i}.A=a_{e_i})}(T_{e_i}.P)$;
14)                 $T \leftarrow$update($T, a, p$);
15)     return $T$;

function update
        /* input: relational table $T$, value set $a$, probability $p$; output: relational table $T$ */
1)      if $a \notin T.A$ then $T \leftarrow T \cup \{(a, p)\}$;
2)      else replace $(a, p') \in T$ by $(a, p + p')$;
3)      return $T$;
```

Figure 2.11: Algorithm SP for ID probabilistic aggregate operators

```
    2. probability table $T \leftarrow$ bottom-up($r$) where $r$ is the root;
    3. add a tuple (undefined, $u$) into $T$ where $u = 1 - \sum_{p \in T.P} p$;
    4. construct the resulting probabilistic instance $\mathcal{I}'$ from $T$;
• bottom-up
    1. if $o$ is a leaf, then
        (a) if $o \in Y$, then $a \leftarrow \{\mathsf{val}(o)\}$;
        (b) $T \leftarrow \{(a, 1)\}$, return $T$;
    2. let $C \leftarrow \{o_1, \ldots, o_n\}$ denote the children of $o$ such that $T_i \leftarrow$ bottom-up($o_i$) is not
       undefined;
    3. for every non-empty subset $c_i$ of of $C$,
        (a) calculate the marginal probability of existence of $c_i$ given $o$;
        (b) for each combination of values, each from an element in $c_i$, we aggregate them,
            multiple their probabilities $p$ from the tables $T_i$, which is finally multiplied by the
            marginal probability above; $T$ is updated with the aggregate result and probability;
    4. return $T$;
```

Consider the probabilistic instance $I1$ in Figure 2.8 and a probabilistic aggregate operator $\mathsf{count}^{\mathsf{P}}(I1, I1.convoy.*)$ which returns the number of vehicles in a convoy. SP starts by locating the set $Y = \{tank1, tank2, truck1\}$ of objects. As count is not ID, we set the values of objects in Y to $\{1\}$ and use the aggregate function sum instead.

After obtaining Y, we know K (K consists of the Y's and their ancestors) contains all objects. SP calls the function *bottom-up* recursively. At the leaves ($tank1, tank2, truck1$), *bottom-up* passes up the stored value (1) with probability 1. For example, at $convoy1$, it obtains $T_1 = \{(\{1\}, 1)\}$ from $tank1$ where the first attribute store the aggregate value of objects under (and including) $tank1$ which belong to Y. The second attribute is the corresponding conditional probability (given $tank1$ exists). It then considers each subset of children of $convoy1$. For $\{tank1\}$, the only possible combination of the values from $tank1$ is $\{(\{1\}, 1)\}$. The results of aggregate of the first attribute is: $\mathsf{sum}(\{1\}) = \{1\}$ with probability $p = \wp(convoy1)(\{tank1\}) \times 1 = 0.4$. We then update the table of *convoy*1 by adding a tuple $(\{1\}, 0.4)$. Similarly, the remaining subsets of children $\{tank2\}$, $\{\}$ and $\{tank1, tank2\}$ are considered and the table at *convoy*1 becomes $T_1 = \{(\{1\}, 1)\}$, which means that *convoy*1 must have exactly one vehicle. SP works similarly on the branch of *convoy*2 and obtains the table $T_2 = \{(\{1\}, 1)\}$.

At the root, we consider subset of children of $I1$. For example, for $\{convoy1, convoy2\}$, the only possible combination of the values from $tank1, tank2$ is $\{(\{1\}, 1), (\{1\}, 1)\}$. The results of aggregating the first attribute is: $\mathsf{sum}(\{1, 1\}) = \{2\}$ with probability $p = \wp(I1)(\{convoy1, convoy2\}) \times 1 \times 1 = 0.5$. We then update the table T of $I1$ by adding a tuple $(\{2\}, 0.5)$. Similarly, we update the table with the tuple $(\{1\}, 0.3)$ for $\{convoy1\}$ and $(\{1\}, 0.2)$ for $\{convoy2\}$. The table now becomes $T = \{(\{1\}, 0.5), (\{2\}, 0.5)\}$, which means that there are one to two vehicles, each with the equal probability. SP finally creates a probabilistic instance with this result.

2.7.2 Complexity of SP Algorithm

As the data in each object is required at most once in SP, only one disk scan is needed.

Space complexity: The worst case space complexity is $O(2^{|Y|})$, where Y is the set of objects whose values we want to aggregate. The worst case upper bound can be reduced, depending on f. For example, the worst case upper bound of space required by $\mathsf{count}^{\mathsf{P}}$ is just $|Y|$.

Time complexity: Consider a non-leaf object o. The time complexity of *bottom-up* is bounded by the product of the size of each child's probability table and 2^n where n is the number of children of o in line 5. Since the size of each child o_i's probability table is bounded by 2^{j_i} where j_i is the number of selected objects under it, the product is bounded by $2^{\sum j_i} = 2^j$ where j is the number of selected objects under o. Thus, the complexity now becomes $O(2^{n+j})$. However, if $|\wp(o)| < 2^n$, we can modify the algorithm slightly so that, instead of trying all possible subsets of children (in E), we try all the entries (with elements in E) of o's OPF. In this way, we can reduce the complexity to $O(|\wp(o)|2^j)$.

It follows that the total time complexity is $O(|\mathcal{I}|2^{|Y|})$. For most aggregates (such as $\mathsf{count}^{\mathsf{P}}, \mathsf{sum}^{\mathsf{P}}, \mathsf{avg}^{\mathsf{P}}$, and even $\mathsf{max}^{\mathsf{P}}, \mathsf{min}^{\mathsf{P}}$) where the table size is non-decreasing while going from the leaves to the root, we can use the following upper bound which is more precise: $O(|\mathcal{I}| \cdot |T_{root}|)$ where $\mathcal{I}$ is the size of the probabilistic instance including the size of all OPF entries, and T_{root} is the probability table at the root.

2.7.3 Pruning

Time complexity increases with the actual size of the final probability table passed up from the root, i.e., the total number of possible aggregate values. If this size is not unreasonably large, we can effectively prune our computation. For example, when a probability table T is passed up, all values with a probability below a preset threshold can be eliminated. Another method is to keep the h most probable values in the probability table. In addition, a hybrid of the above two and other pruning methods can be used. Since the effects of pruning techniques depends on the application domain, the instance, the aggregate operator used and the thresholds used, users are advised to fine tune their pruning methods according to the desired performance of their applications. In Section 2.8.3, I will investigate the effectiveness of pruning by keeping the size of probability table of every non-root object (finalized at the end of the function *bottom-up*) not exceeding a threshold ranging from 50 to 300 probability entries.

2.7.4 SE Algorithm

The SE algorithm (shown in Figure 2.12) modifies SP for the expected aggregate computations. Unlike SP, instead of keeping the whole probability table, SE only keeps the expected aggregate value and propagates it up.

To see how SE works, consider the probabilistic instance $I1$ in Figure 2.8. Consider the probabilistic aggregate operator $\mathsf{count}^{\mathsf{E}}(I1, \{I1.convoy.*\})$ which returns the total number of vehicles. SE first locates $Y = \{tank1, tank2, truck1\}$. The aggregate function to be applied on them is count. However, as count is not ID, we will set the values of selected objects (in Y) to be $\{1\}$ and use the aggregate function sum instead. After computing Y, we know that K contains all objects. SE calls the function *bottom-up* recursively. At the leaves ($tank1, tank2$, $truck1$), *bottom-up* passes up the stored value with probability 1. For example, at $convoy1$, it obtains $T_1 = \{(\{1\}, 1)\}$

```
expected aggregate function: $\mathsf{f}^{\mathsf{E}}$ /* input: $\mathcal{I}, \alpha$; output: real number $e$ */
1)  locate set $Y$ of objects from $\mathcal{I}$ according to the definition of $\mathsf{agg}^{\mathsf{E}}$ and input $\alpha$;
2)  $K \leftarrow$ all elements in $Y$ and their ancestors; $T \leftarrow$ bottom-up($r$) where $r$ is the root;
3)  $\{e\} \leftarrow T.A$; $p \leftarrow T.P$;
4)  return $e/p$;

function bottom-up /* input: object $o$; output: relational table $T = (A, P)$ */
1)  if $o$ is a leaf, then
2)      if $\tau(o)$ is compatible with agg then $T \leftarrow \{(\{\mathsf{val}(o)\}, 1)\}$; else $T \leftarrow$ undefined;
3)      return $T$;
4)  let $C \leftarrow \{o_1, \ldots, o_n\}$ denote the children of $o$ which are also in $K$
        such that $T_i \leftarrow$ bottom-up($o_i$) is not undefined;
5)  for every OPF entry $\wp(o)(F)$,
6)      let $E = \{o_{e_1}, \ldots, o_{e_m}\} = F \cap C$;
7)      if $|E| > 0$, then
8)          $a \leftarrow \mathsf{agg}(\cup_{i=1}^{m} a_{e_i})$ where $a_{e_i} \in T_{e_i}.A$,
9)          if $o \in Y$, then $a \leftarrow \mathsf{agg}(a \cup a_{\mathsf{default}})$;
10)         $p \leftarrow \wp(o)(F) \times \prod_{i=1}^{m}(T_{e_i}.P)$; $T \leftarrow$update$(T, a, p)$;
11) return $T$;

function update
/* input: relational table $T$, value set $a = \{e\}$, probability $p$; output: relational table $T$ */
1)  if $|T| \neq 0$ then $T \leftarrow T \cup \{(\{e \times p\}, p)\}$; else replace $(\{e'\}, p') \in T$ by $(\{e' + e \times p\}, p + p')$;
2)  return $T$;
```

Figure 2.12: Algorithm SE for ID expected aggregate functions

from $tank1$ and $T_2 = \{(\{1\}, 1)\}$ from $tank2$. It then considers each OPF entry. For $\{tank1\}$, the only possible combination of the values from $tank1$ is $\{(\{1\}, 1)\}$. The result of the aggregate computation is: $\mathsf{sum}(\{1\}) = \{1\}$ with probability $p = \wp(convoy1)(\{tank1\}) \times 1 = 0.4$. We then update the table of *convoy*1 by adding a tuple $(\{1 \times 0.4\}, 0.4)$. Similarly, the remaining OPF entry $\{tank2\}$ is considered and the table at *convoy*1 becomes $T_1 = \{(\{1\}, 1)\}$, which means that *convoy*1 must have exactly one vehicle. SE works similarly on the branch of *convoy*2 and obtain a table $T_2 = \{(\{1\}, 1)\}$.

At the root, we consider the OPF entries of $I1$. For example, for the subset $\{convoy1$, $convoy2\}$, the only possible combination of the values from $tank1, tank2$ is $\{(\{1\}, 1), (\{1\}, 1)\}$. The result is $\mathsf{sum}(\{1, 1\}) = \{2\}$, with probability $p = \wp(I1)(\{convoy1, convoy2\}) \times 1 \times 1 = 0.5$. We then update the table T of *convoy*1 by adding a tuple $(\{2 \times 0.5\}, 0.5)$. Similarly, we update the table with the tuple $(\{1 \times 0.3\}, 0.3)$ for $\{convoy1\}$ and $(\{1 \times 0.2\}, 0.2)$ for $\{convoy2\}$. The final table becomes $T = \{(\{1.5\}, 1)\}$, which means that the expected number of vehicles is 1.5.

2.7.5 Complexity of SE Algorithm

As the table only contains a single value, SE's space complexity is $O(|Y|)$ where Y is the set of objects selected. SE's time complexity is $O(b|\mathcal{I}|)$ where b is the branch factor (the number of children), and $\mathcal{I}$ is the size of the probabilistic instance including the size of all OPF entries. SE is therefore much faster (and more space efficient) than SP.

2.8 PXML Experiments

2.8.1 Experimental Design

I have implemented a prototype system in C on a Dell PowerEdge with 1.13 Ghz PIII processors, 4GB RAM running Linux. I generated probabilistic instances as balanced trees with the depth (of the tree) ranging from 3 to 9 and with every non-leaf object having b children (b is the branch factor, from 2 to 14) and every leaf object having a value in $[0, r-1]$ (r is the range of the value, from 2 to 32). I assume that there is no cardinality constraint, so the total number of OPF entries in a local interpretation for each non-leaf object is 2^b. There are two kinds of distributions of OPF entries within a given object. In the uniform distribution, all potential child sets have the same probability. In the exponential distribution, exponentially decreasing probabilities are assigned randomly to the potential child sets. There are two kinds of random edge labelings. In the *same label* (or SL) labeling, all children of the same parent have the same labels (shown as SL in the figures). The *fully random* (or FR) labeling assigns random labels to all children of the same parent. I evaluated the performance of ancestor projection, selection, Cartesian product and aggregate operations.

2.8.2 Performance results of algebra experiments

In this set of results, I include graphs of total query time and the time required to update the local interpretation ($\wp$). The total query time is the sum of the time to make a copy of the input instance, the time to locate objects satisfying a path expression (and the object id of the object to be selected in the case of selection operation), the time to update the structure of the instance (for ancestor projection only), the time to update the local interpretation, and the time to write the resulting instance onto a disk. For each depth, each branching factor and each operation, I generated 10 instances. For each instance, I kept track of labels used by edges of objects in each depth and generated 10 random queries that returned results consisting of the root and at least one more object. For example, in an instance of depth 2 where the edges connecting objects of depth 1 to their parents have labels from the set $\{a, b\}$ and the edges connecting objects of depth 2 to their parents have labels from the set $\{c, d\}$, the path expression of an ancestor projection query generated has the form $r.x_1.x_2$ where r is the root id, $x_1 \in \{a, b\}$ and $x_2 \in \{c, d\}$. I accepted this query in the performance measurement in our experiment only if there were objects satisfying the path expression of this query. For each selection query, I generated a path expression p (in the same way) and similarly found a set $SelObj$ of objects satisfying the path expression. The selection queries used have the form $p = o$ where o is an object id selected randomly from $SelObj$. For each Cartesian product query, I generated two instances of the same depth and the same branch factor. In my experiments, I only consider single path expressions as defined in Definition 2.5.1. In addition, I set the length of the query (the length of the path expression) equal to the depth of the instance because, according to the definition of ancestor projection and selection, the objects whose depth exceeds the length of the query will not be considered and will not affect the query results and the local interpretation of such objects does not need updating. For each combination of depth and branching factor, I took the average of 100 such queries.

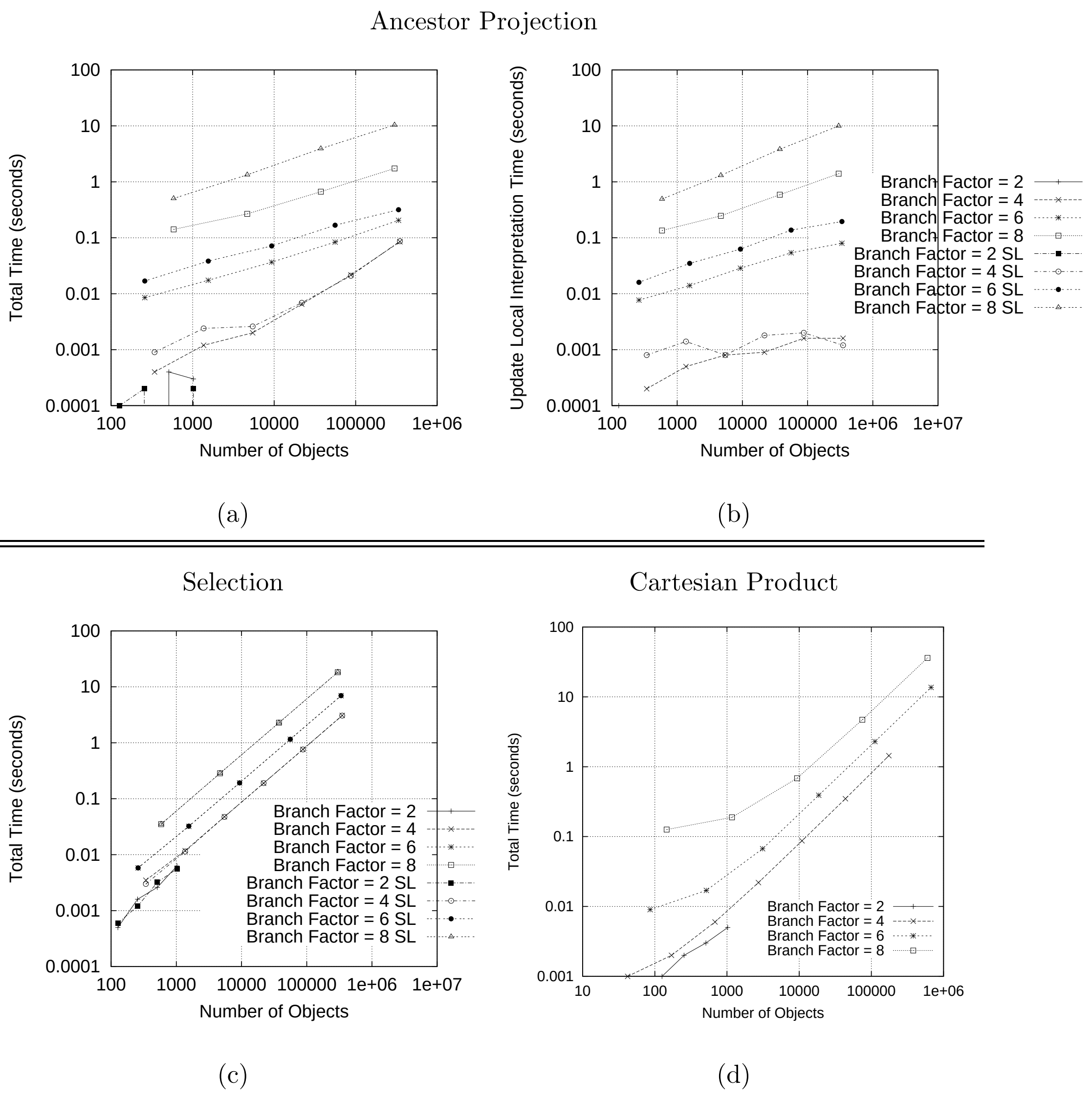


Figure 2.13: (a) Total query time of ancestor projection, (b) update local interpretation time of ancestor projection, (c) total query time of selection and (d) total query time of Cartesian product for instances of sizes ranging from 10 to 1000000, branching factors ranging from 2 to 8 and two different labeling schemes (SL = same labels for children of the same parent; otherswise, all random labels).

Figures 2.13 (a) and (b) show the total query processing time and the time of updating

$\wp$ in an ancestor projection. By comparing the two graphs, we see that the time to update $\wp$ dominates the total query processing time. Figure 2.13 (b) shows that the time to update $\wp$ is linear in the number of objects, which can be explained by the fact that $\wp(o)$ is updated only once for each object o. Recall that the time to propagate probabilities from children of an object o to o is quadratic in the size of $\wp(o)$. When we fix the number of objects, we see from Figure 2.13 (b) that the time increases by a multiple less than 16 when the branching factor increases by 2, i.e. the number of entries in $\wp(o)$ is multiplied by 4. In addition, under the setting of having the same labels for all children of the same parent, the time is longer than the other setting. One possible reason is that in the former setting, there is a higher chance that more objects are located by the path expression, and so there are more objects to be kept whose local interpretations are to be updated. The final note is that the updating time for 299593 objects and branch factor 8 SL (the top rightmost point) is 10.4s, which seems to be long. However, it is reasonable when we consider the fact that about 700 - 5000 objects are kept and about 28000 - 200000 $\wp(o)$ entries are processed.

Figures 2.13 (c) and (d) show the total query processing time for selection and Cartesian product. Their results are different from that of ancestor projection as the time to write the result onto the disk dominates the total query processing time (the time of updating $\wp$ in selection only involves less than 0.001 second; the time of updating the root data in Cartesian product only takes less than 0.01 second). The reason is that the remaining structure of the resulting instance does not change after selection. Hence, the amount of data to be written is much larger than the number of objects whose $\wp(o)$ needs to be updated (the number is the same as the depth) in selection, or the data of the root in Cartesian product. The total time is linear in the number of objects and linear in the number of entries in $\wp(o)$ of each object o. The quantity of data to be written is independent of whether SL or FR labelings are used.

2.8.3 Performance results of aggregate experiments

In this section, I only evaluate the performance of the core part of aggregate operations, so I assume that the probabilistic instances generated are the results after selecting objects of interest and their ancestors. Thus, the instances used here can be treated as just small subsets of a much larger instance.

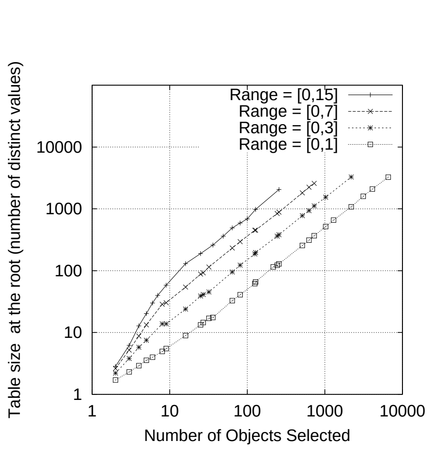
I evaluated two aggregate operators which are typical and useful. One is $\mathsf{avgsum}^{\mathsf{P}}(\mathcal{I}, \{p_1, p_2\})$ which selects object sets Y_1, Y_2 satisfying path expressions p_1, p_2 from probabilistic instance $\mathcal{I}$, and then finds all the possible answers (with probabilities) of the quotient from dividing the sum of values of objects in Y_2 by the number of objects in Y_1. I use an extended version of SP program to handle this operator. This extended SP works similarly to SP, but it keeps aggregate results and probabilities for both Y_1 and Y_2 and processes an extra calculation on them in the final stage, which is division for $\mathsf{avgsum}^{\mathsf{P}}$ to get the average. The other operator is $\mathsf{sum}^{\mathsf{P}}(\mathcal{I}, \{p\})$.

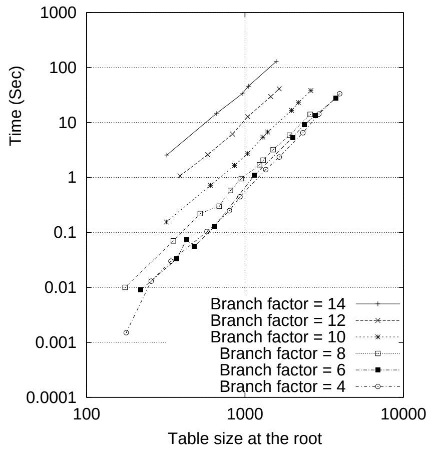
The experimental results show that the second operator's corresponding expected probabilistic function $\mathsf{sum}^{\mathsf{E}}$ can be computed by the SE algorithm within 0.5 second even when there are 10^5 selected objects. I also implemented the pruning technique by keeping the size of probability table of every non-root object (finalized at the end of the function *bottom-up*) not exceeding a threshold ranging from 50 to 300.

I now show results (averaged over 10 to 100 runs for each setting) of the SP algorithm about (1) the relationship of the number of selected objects and the size of table at the root, (2) the running time of SP (without pruning), and (3) the performance (running time and relative error) of SP with pruning.

Root Table Size and Running Time of SP

Figure 2.14(a) shows that the size of the probability table T_{root} at the root (in SP program for $\mathsf{sum}^{\mathsf{P}}(\mathcal{I}, \{p\})$) is approximately linear in the number of selected objects because the slope is approximately 1. The table size ($|T_{root}|$) increases with the value range (r) to the power of about 1.4, i.e., $|T_{root}| \propto r^{1.4}$.





(a) (b)

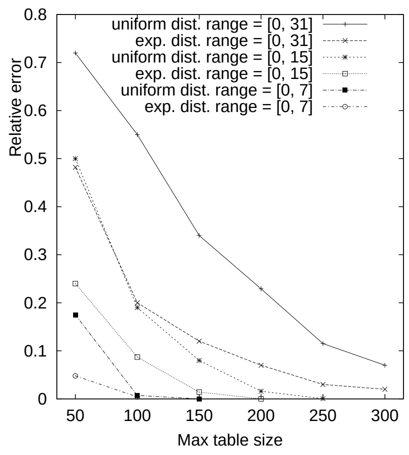
Figure 2.14: (a) In SP for sum^P, the linear relationship between the size of the table T at the root (number of distinct values) and the number of objects selected (in log scale), (b) running time of extended SP for $\mathsf{avgsum}^\mathsf{P}$ against the size of the root table (in log scale).

Figure 2.14(b) shows that the running time of extended SP to compute $\mathsf{avgsum}^\mathsf{P}$ (including the time for dividing the sum in Y_2 by $|Y_1|$) is sub-cubic relative to the root table size because the slope is between 2 and 3 The time complexity of this extended SP is $O(\beta b|\mathcal{I}| \cdot |T_{root}|^3)$. When the branch factor increases by 2, the number of OPF entries increases by a factor of four. We see from the graph that after counting the effect of b on the complexity, the time increases approximately linearly with the probabilistic instance size (the total number of OPF entries). Note that the curves of branch factor $4, 6, 8$ are close to each other. A possible reason is the common overhead which dominates over the effect of branch factor when the branch factor is small.

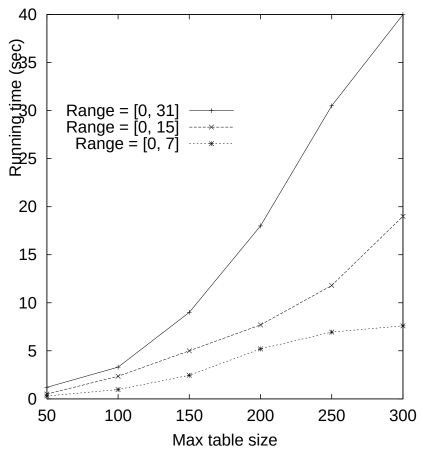
Performance of Pruning

Here I generate instances with depth 4 and branch factor 4 (i.e., 256 selected objects) and evaluate the operator sum^P. When the range of values is $8, 16, 32$, the mean root probability table size is $918, 1872, 4043$, and the running time is $6.84, 55.84, 566.5$ seconds respectively.

Figure 2.15(a) shows that the relative error of the result with pruning compared with the case where there is no pruning decreases with the increase in the range of values and the maximum



(a)



(b)

Figure 2.15: (a) Error of SP with pruning (relative to without pruning) for $\mathsf{sum}^{\mathsf{P}}$ against the maximum table size allowed, (b) Running time of SP with pruning for $\mathsf{sum}^{\mathsf{P}}$ against the maximum table size allowed.

probability table size allowed at non-root objects. Instances with an exponential distribution give less error than those with uniform distribution because the former produces aggregate values with a larger difference in probabilities, and hence the pruning of aggregate values with lower probabilities gives less effect to the final answer.

Figure 2.15(b) shows that the running time of SP with pruning increases with the range values and maximum table size, but the time are all much less than the time without pruning (except the curve of range 8 which flattens when the maximum table size allowed is even larger than the original non-root table size).

2.9 Summary

In this chapter, I have introduced the probabilistic semistructured data model (PXML). I have given two semantics of this model, namely the local semantics and the global semantics. The global semantics gives an intuition of possible-worlds interpretation of the semantics of the data model while the local semantics allows efficient direct manipulation to avoid the expensive handling of exponentially large number of compatible instances (possible worlds). I have presented an algebra to manipulate PXML data instances and showed how to evaluate such queries efficiently. Furthermore, I introduced two semantics (possible-worlds and expectation) for aggregations on PXML instances. The second semantics can be computed efficiently while the first semantics can be computed with acceptable performance with pruning. Experimental results have verified the feasiblity of PXML algebra and aggregations.

The PXML model presented uses point probability to model uncertainty of information. We will see how this can be extended to use interval probability in the next chapter.

Chapter 3

Probabilistic Interval XML Model

In this chapter, I extend the PXML model developed in the previous chapter so that interval probabilities are used instead of point probabilities to represent uncertainty. In Section 3.1, I will first provide some necessary definitions on which the PIXML model will be built. In Section 3.2, I then develop the PIXML probabilistic interval data model. I then provide two alternative formal semantics for the PIXML model. Section 3.3 presents the first semantics which is a declarative (model-theoretic) semantics. The remaining sections present the second semantics which is an operational semantics that can be used for computation. In the W3C formal specification of XML, an instance is considered as an ordered rooted tree in which cycles can possibly appear[80]. In this chapter, I will assume that an instance is an acyclic graph - this assumption will be needed to provide a coherent semantics to PIXML databases. However I do not restrict attention to tree-structures. I also provide an operational semantics that is provably correct for a queries over a large class of probabilistic instances called tree-structured instances.

3.1 Interval Probabilities

An extension to handle interval probabilities is useful because almost all statistical evidence involves margins of error. For instance, when a statistical estimate says that something is true with probability 95% with a $\pm 2\%$ margin of error, then this really corresponds to saying the event's probability lies in the interval $[0.93, 0.97]$. Likewise, using intervals is valuable when one does not know the relationship between different events. For example, if we know the probabilities of events e_1, e_2 and want to know the probability of both of them holding, then we can, in general, only infer an interval for the conjunction of e_1, e_2 ([9, 26]) *unless* we know something more about the dependencies or lack thereof between the events. Furthermore, it is also natural for a human judgement to be expressed as an interval probability rather than an exact point probability. For example, a human expert may say that the vehicle in a picture is *likely* a tank. If he or she is asked to indicate a probability, he or she may feel difficulty to give a point probability (say, 60%), but he or she may feel more natural to give an interval probability (say, 40% to 70%), which also reflects the nature of uncertainty. An extreme case is $[0, 1]$ (i.e., "0% to 100%") which indicates that we have no information about the probability or likeliness of an event.

Below I quickly review definitions and give some important theorems for interval probabilities. Given an interval $I = [x, y]$ I will often use the notation $I.lb$ to denote x and $I.ub$ to denote y.

An ***interval function*** ι w.r.t. a set S associates, with each $s \in S$, a closed subinterval $[lb(s), ub(s)] \subseteq [0, 1]$. ι is called an ***interval probability function*** if $\sum_{s \in S} lb(s) \leq 1$ and $\sum_{s \in S} ub(s) \geq 1$. A ***probability distribution*** w.r.t. a set S over an interval probability function ι is a mapping $\mathcal{P} : S \to [0, 1]$ where

1. $\forall s \in S, lb(s) \leq \mathcal{P}(s) \leq ub(s)$, and
2. $\Sigma_{s \in S} \mathcal{P}(s) = 1$.

Lemma 1 *For any set S and any interval probability function ι w.r.t. S, there exists a probability distribution P(S) which is compatible with ι.*

Proof: There are potentially many possible distributions that are compatible with ι. One solution is the distribution that is as close to "the middle" of each interval as possible. Let $\sum_{s \in S} lb(s) = L$ and $\sum_{s \in S} ub(s) = U$. By the definition of interval probability function, we know that $L \leq 1$ and $U \geq 1$. A probability function that is consistent with the interval constraints is: $p(s_i) = lb(s_i) + (ub(s_i) - lb(s_i)) * \frac{1-L}{U-L}$. It is easy to check that $\sum_i p(s_i) = 1$ and that $lb(s_i) \leq p(s_i) \leq ub(s_i)$. ∎

It may be noted that among the possible distributions, there has been work such as [34] to find the one with maximum entropy. An interval probability function ι w.r.t. S is ***tight*** iff for any interval probability function ι' w.r.t. S such that every probability distribution $\mathcal{P}$ over ι is also a probability distribution over ι', $\iota(s).lb \geq \iota'(s).lb$ and $\iota(s).ub \leq \iota'(s).ub$ where $s \in S$. If every probability distribution P over ι' is also a probability distribution over ι, then we say that ι is the **tight equivalent** of ι'. A ***tightening operator***, tight, is a mapping from interval probability functions to interval probability functions such that $\mathsf{tight}(\iota)$ produces a tight equivalent of ι. The following result (Theorem 2 of [16]) tells us that we can always tighten an interval probability function.

Theorem 3.1 *[16, Theorem 2] Suppose ι, ι' are interval probability functions over S and $tight(\iota') = \iota$. Let $s \in S$. Then:*

$$\iota(s) = \left[max\left(\iota'(s).lb, 1 - \sum_{s' \in S \wedge s' \neq s} \iota'(s').ub \right), min\left(\iota'(s).ub, 1 - \sum_{s' \in S \wedge s' \neq s} \iota'(s').lb \right) \right].$$

For example, we can use the above formula to check that the interval probability functions in Figure 2.3 are tight. Throughout the rest of this chapter, unless explicitly specified otherwise, I will assume that all interval probability functions are tight.

3.2 The PIXML Data Model

In probabilistic XML, we have uncertainty because we do not know which of various possible semistructured instances is "correct." Rather than defining a point probability for each instance, we will use interval probabilities to give bounds on the probabilities for structure. In this section, I will first define a probabilistic *interval* semistructured instance. The following section will describe its model theoretic semantics.

Recall the definitions of a weak instance (Definition 2.2.5), a potential l-child set (Definition 2.2.6), a potential child set (Definition 2.2.7), a weak instance graph (Definition 2.2.8), an object probability function (OPF) (Definition 2.2.9) and a local interpretation (Definition 2.2.11). A probabilistic semistructured instance defined in Section 2.2 uses a local interpretation to map a set of OPFs to non-leaf objects for the *point* probabilities of children sets. Here, a probabilistic interval semistructured instance uses ipf for a similar purpose; however, instead of point probabilities, *interval* probabilities are used in ipf.

Definition 3.2.1 *A **probabilistic instance** $\mathcal{I}$ is a 6-tuple $\mathcal{I} = (V, \mathsf{lch}, \tau, \mathsf{val}, \mathsf{card}, \mathsf{ipf})$ where:*

o	l	$\mathsf{lch}(o, l)$
I1	convoy	{ convoy1, convoy2 }
convoy1	tank	{ tank1, tank2 }
convoy2	truck	{ truck1 }

o	$\tau(o)$	$\mathsf{val}(o)$
tank1	tank-type	T-80
tank2	tank-type	T-72
truck1	truck-type	rover

o	l	$\mathsf{card}(o, l)$
I1	convoy	[1,2]
convoy1	tank	[1,1]
convoy2	truck	[1,1]

$\mathsf{c} \in \mathsf{PC}(I1)$	$\mathsf{ipf}(I1, \mathsf{c})$
{ convoy1}	[0.2, 0.4]
{ convoy2}	[0.1, 0.4]
{ convoy1, convoy2}	[0.4, 0.7]

$\mathsf{c} \in \mathsf{PC}(convoy1)$	$\mathsf{ipf}(convoy1, \mathsf{c})$
{ tank1}	[0.2, 0.7]
{ tank2}	[0.3, 0.8]

$\mathsf{c} \in \mathsf{PC}(convoy2)$	$\mathsf{ipf}(convoy2, \mathsf{c})$
{ truck1}	[1, 1]

Figure 3.1: A probabilistic instance for the surveillance domain.

1. $\mathcal{W} = (V, \mathsf{lch}, \tau, \mathsf{val}, \mathsf{card})$ *is a weak instance and*

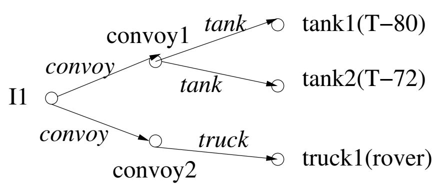
2. ipf *is a mapping which associates with each non-leaf object* $o \in V$*, an interval probability function* ipf *w.r.t.* $\mathsf{PC}(o)$*, where* $\mathsf{c} \in \mathsf{PC}(o)$ *and* $\mathsf{ipf}(o, \mathsf{c}) = [lb, ub]$.

Intuitively, a probabilistic instance consists of a weak instance, together with probability intervals associated with each potential child set of each object in the weak instance. Similarly, given a probabilistic instance, we can obtain its weak instance graph from its corresponding weak instance.

Example 3.2.1 *Figure 2.3 shows a very simple probabilistic instance.*[1] *The set* $\mathcal{O}$ *of objects is* $\{I1, convoy1, convoy2, tank1, tank2, truck1\}$*. The first table shows the legal children of each of the objects, along with their labels. The cardinality constraints are shown in the third table; for example object I1 can have from one to two convoy-children. The tables on the right of Figure 2.3 shows the* ipf *of each potential child of I1, convoy1 and convoy2. Intuitively,* $\mathsf{ipf}(I1, \{convoy1\}) = [0.2, 0.4]$ *says that the probability of having only convoy1 is between* 0.2 *and* 0.4*.*

[1]Here we only show objects with non-empty set of children.

(a)



(b)

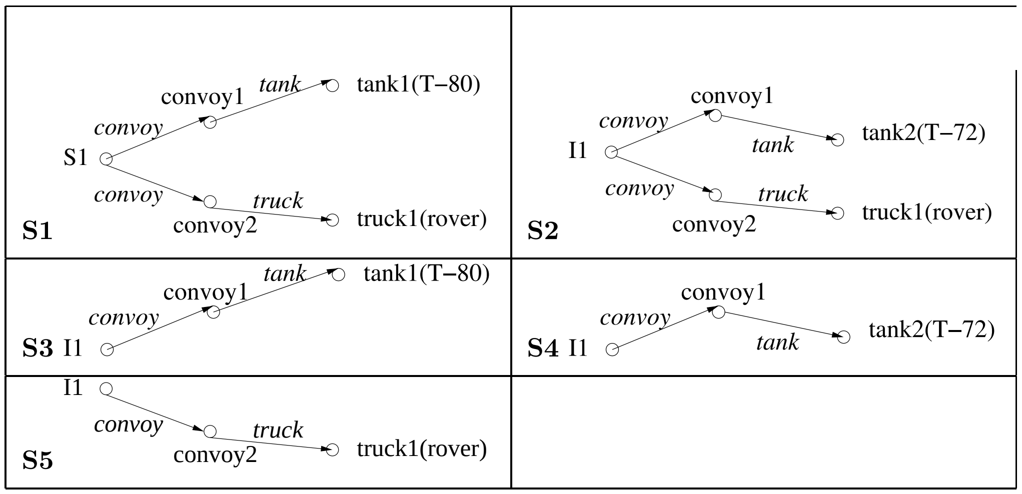


Figure 3.2: (a) The graph structure of the probabilistic instance in Figure 2.3. (b) The set of semistructured instances compatible with the probabilistic instance in Figure 2.3.

In the rest of this chapter, I will omit the term *interval* in "probabilistic interval semistructured instance" and write it as "probabilistic semistructured instance" or simply "probabilistic instance".

3.3 PIXML : Declarative Semantics

I am now ready to describe the declarative semantics of a probabilistic instance. Recall Definition 2.3.1 of an ordinary semistructured instance to be compatible with a weak instance. Intuitively, this means that the graph structure of the semistructured instance is consistent with the graph structure and cardinality constraints of the weak instance. If a given object o occurs in the weak instance $\mathcal{W}$ and o occurs also in a compatible semistructured instance $\mathcal{S}$, then the children of o in $\mathcal{S}$ must be a set of potential children of o in $\mathcal{W}$.

I use $Domain(\mathcal{W})$ to denote the set of all semistructured instances that are compatible with a weak instance $\mathcal{W}$. Similarly, for a probabilistic instance $\mathcal{I} = (V, \mathsf{lch}_{\mathcal{I}}, \tau_{\mathcal{I}}, \mathsf{val}_{\mathcal{I}}, \mathsf{card}, \mathsf{ipf})$, I use $Domain(\mathcal{I})$ to denote the set of all semistructured instances that are compatible with $\mathcal{I}$'s associated weak instance $\mathcal{W} = (V, \mathsf{lch}_{\mathcal{I}}, \tau_{\mathcal{I}}, \mathsf{val}_{\mathcal{I}}, \mathsf{card})$.

Figure 3.2 shows the set of all semistructured instances compatible with the weak instance corresponding to the probabilistic instance defined in Example 3.2.1.

I will now define global interpretations that associate probability distributions over the set of all semistructured instances that are compatible with a probabilistic instance.

Definition 3.3.1 *Suppose we have a weak instance* $\mathcal{W} = (V, \mathsf{lch}, \tau, \mathsf{val}, \mathsf{card}, \mathsf{ipf})$. *A* ***global interpretation*** $\mathcal{P}$ *is a mapping from Domain*$(\mathcal{W})$ *to* $[0, 1]$ *such that* $\Sigma_{S \in Domain(\mathcal{W})} \mathcal{P}(S) = 1$.

Intuitively, a global interpretation is a distribution over the semistructured instances compatible with a weak instance. On the other hand, local interpretations assign semantics on an object by object basis. To define local interpretations, I will use the definition of OPFs in Definition 2.2.9.

Definition 3.3.2 *Suppose* $\mathcal{W} = (V, \mathsf{lch}, \tau, \mathsf{val}, \mathsf{card})$ *is a weak instance. A* ***local interpretation*** *is a mapping* $\wp$ *from the set of non-leaf objects* $o \in V$ *to object probability functions, i.e.* $\wp(o)$ *returns an OPF for* o *w.r.t.* $\mathcal{W}$.

Intuitively, a local interpretation specifies, for each non-leaf object in the weak instance, an object probability function.

3.3.1 Connections between Local and Global Semantics

In this section, I show that there is a transformation to construct a global interpretation from a local one, and vice versa, and that these transformations exhibit various nice properties.

Definition 3.3.3 ($\tilde{W}$ operator) *Let* $\wp$ *be a local interpretation for a weak instance* $\mathcal{W} = (V, \mathsf{lch}, \tau, \mathsf{val}, \mathsf{card})$. *Then* $\tilde{W}(\wp)$ *is a function which takes as input, any* $S \in Domain(\mathcal{W})$ *and is defined as follows:* $\tilde{W}(\wp)(S) = \prod_{o \in S} \wp(o)(\mathsf{C}_S(o))$ *where* $\mathsf{C}_S(o)$ *are the actual children of* o *in* S.

The following theorem says that $\tilde{W}(\wp)$ is a valid global interpretation for $\mathcal{W}$ with an acyclic weak instance graph.

Theorem 3.2 *Suppose $\wp$ is a local interpretation for a weak instance $\mathcal{W} = (V, \mathsf{lch}, \tau, \mathsf{val}, \mathsf{card})$ with an acyclic weak instance graph. Then $\tilde{W}(\wp)$ is a global interpretation for $\mathcal{W}$.*

Proof:

The proof is by induction.

The length of the path from the root r of a rooted directed acyclic graph G to an object o is the depth of o in G. The largest depth of any object in G is the height of G. I will call the root of a graph of height k as o_k. Now, I define k sets, namely $O_{G,k}, O_{G,k-1}, \ldots, O_{G,0}$, which contain objects of depth $0, 1, \ldots, k$.[2] A set $O_{G,k}$ is defined to contain o_k only. $O_{G,j-1}$ is defined as the union of the sets of children of objects in $O_{G,j}$, minus $O_{G,j} \cup \ldots \cup O_{G,k}$, i.e., $O_{G,j-1} = \bigcup_{o \in O_{G,j}} \mathsf{C}(o) - \bigcup_{m=j}^{k} O_{G,m}$. Intuitively, the depth of objects in $O_{G,j}$ is $k - j$. Suppose $\wp(o)$ returns an OPF ω_o for o w.r.t. $\mathcal{W}$.

Consider the case that the height of $\mathcal{W}$ is 1. The root o_1 is the only object in any $S \in \mathit{Domain}(\mathcal{W})$ that can have children. Thus,

$$\tilde{W}(\wp)(S) = \prod_{o \in S} \wp(o)(\mathsf{C}_S(o)) = \wp(o_1)(\mathsf{C}_S(o_1)) = \omega_{o_1}(\mathsf{C}_S(o_1))$$

In order for $\tilde{W}(\wp)$ to be a global interpretation for $\mathcal{W}$, the sum of $\tilde{W}(\wp)(S)$ over all S compatible with $\mathcal{W}$ should be equal to one. In this case, each distinct S has the object o_1 to contain a distinct potential child set. By Definition 2.2.9, the sum always gives one.

$$\sum_{S \in \mathit{Domain}(\mathcal{W})} \tilde{W}(\wp)(S) = \sum_{\mathsf{c}_0 \in \mathsf{PC}(o_1)} \omega_{o_1}(\mathsf{c}_0) = 1.$$

Consider the case that the height of $\mathcal{W}$ is 2. The root is denoted as o_2.

$$\begin{aligned}
\tilde{W}(\wp)(S) &= \prod_{o \in S} \wp(o)(\mathsf{C}_S(o)) \\
&= \wp(o_2)(\mathsf{C}_S(o_2)) \prod_{o_1 \in \mathsf{C}_S(o_2)} \wp(o_1)(\mathsf{C}_S(o_1)) \\
&= \omega_{o_2}(\mathsf{C}_S(o_2)) \prod_{o_1 \in \mathsf{C}_S(o_2)} \omega_{o_1}(\mathsf{C}_S(o_1))
\end{aligned}$$

Since $\mathit{Domain}(\mathcal{W})$ contains all possible compatible instances and the set of all potential child sets of an object is independent of other objects, $\mathit{Domain}(\mathcal{W})$ will then contains all possible

[2]I intentionally make the subscript of O as opposite to the depth it is corresponding so that the remaining parts of the proof is simpler and easier to understand when I am using induction.

combinations of every potential child set of every object.

$$\begin{aligned}\sum_{S\in Domain(\mathcal{W})} \tilde{W}(\wp)(S) &= \sum_{\mathsf{c}_1\in \mathsf{PC}(o_2)} \omega_{o_2}(\mathsf{c}_1) \prod_{o_1\in \mathsf{c}_1} \sum_{\mathsf{c}_0\in \mathsf{PC}(o_1)} \omega_{o_1}(\mathsf{c}_0)\\ &= \sum_{\mathsf{c}_1\in \mathsf{PC}(o_2)} \omega_{o_2}(\mathsf{c}_1) \prod_{o_1\in \mathsf{c}_1} 1\\ &= \sum_{\mathsf{c}_1\in \mathsf{PC}(o_2)} \omega_{o_2}(\mathsf{c}_1)\\ &= 1.\end{aligned}$$

Assume that the theorem is true for the cases that the height of $\mathcal{W}$ is $1, \ldots, k+1$. Now, let us consider the case that the height of $\mathcal{W}$ is $k+2$.

$$\begin{aligned}\tilde{W}(\wp)(S) &= \prod_{o\in S} \wp(o)(\mathsf{C}_S(o))\\ &= \prod_{j=1}^{k+2} \prod_{o_j\in O_{G,j}} \wp(o_j)(\mathsf{C}_S(o_j))\\ &= \prod_{j=1}^{k+2} \prod_{o_j\in O_{G,j}} \omega_{o_j}(\mathsf{C}_S(o_j))\end{aligned}$$

While summing up over all compatible instances, I can use the assumption that the subgraph (height $k+1$) of $\mathcal{W}$ without the root o_{k+2} has the product of sum equal to one.[3]

$$\begin{aligned}\sum_{S\in Domain(\mathcal{W})} \tilde{W}(\wp)(S) &= \sum_{\mathsf{c}_{k+1}\in \mathsf{PC}(o_{k+2})} \omega_{o_{k+2}}(\mathsf{c}_{k+1}) \times \prod_{j=1}^{k+1} \left(\prod_{o_j\in O_{\mathcal{W},j}} \left(\sum_{\mathsf{c}_{j-1}\in \mathsf{PC}(o_j)} \omega_{o_j}(\mathsf{c}_{j-1}) \right)\right)\\ &= \sum_{\mathsf{c}_{k+1}\in \mathsf{PC}(o_{k+2})} \omega_{o_{k+2}}(\mathsf{c}_{k+1}) \times 1\\ &= 1.\end{aligned}$$

∎

Example 3.3.1 *Consider the probabilistic instance in Example 3.2.1. Suppose we are given a local interpretation* $\wp$ *such that* $\wp(I1) = \omega_{I1}$, $\wp(convoy1) = \omega_{convoy1}$, $\wp(convoy2) = \omega_{convoy2}$, *where* $\omega_{I1}(\{convoy1\}) = 0.3$. $\omega_{I1}(\{convoy2\}) = 0.2$, $\omega_{I1}(\{convoy1, convoy2\}) = 0.5$, $\omega_{convoy1}(\{tank1\}) =$

[3]Although now the subgraph may have more than one root, it can be proved in a similar way that the product of sum equal to one.

0.4, $\omega_{convoy1}(\{tank2\}) = 0.6$, $\omega_{convoy2}(\ \{truck1\}) = 1$. *Then the probability of a compatible instance $S1$ shown in Figure 3.2 will be:*
$\tilde{W}(\wp)(S1) = \wp(I1)(\{convoy1,\ convoy2\})\ \times\wp(convoy1)(\{tank1\}) \times \wp(convoy2)(\{truck1\}) = 0.5 \times 0.4 \times 1 = 0.2$.

An important question is whether we can go the other way: from a global interpretation, can we find a local interpretation for a weak instance $\mathcal{W}$? It turns out that we can **if** the global interpretation can be factored in a manner consistent with the structure constraints imposed by $\mathcal{W}$. One way to ensure this is to impose a set of independence constraints relating every non-leaf object and its non-descendants in the weak instance graph $\mathcal{G}_{\mathcal{W}}$. The independence constraints are defined below.

Definition 3.3.4 *Suppose $\mathcal{P}$ is a global interpretation and $\mathcal{W} = (V, \mathsf{lch}, \tau, \mathsf{val}, \mathsf{card})$ is a weak instance.* $\mathcal{P}$ **satisfies** $\mathcal{W}$ *iff for every non-leaf object $o \in V$ and each $c \in \mathsf{PC}(o)$, it is the case that*[4]*: $\mathcal{P}(c|o, \mathsf{non\text{-}des}(o)) = \mathcal{P}(c|o)$. Here, $\mathsf{non\text{-}des}(o)$ are the non-descendants of o in $\mathcal{G}_{\mathcal{W}}$.*

In other words, given that o occurs in the instance, the probability of any potential children c of o is independent of any *possible* set of nondescendants. From now on, given a weak instance $\mathcal{W}$, I will only consider $\mathcal{P}$ that satisfies $\mathcal{W}$. The definition below tells us how to associate a local interpretation with any global interpretation.

Definition 3.3.5 ($\tilde{D}$ operator) *Suppose $\mathsf{c} \in \mathsf{PC}(o)$ for some non-leaf object o and suppose $\mathcal{P}$ is a global interpretation. $\omega_{\mathcal{P},o}$, is defined as follows.*

$$\omega_{\mathcal{P},o}(\mathsf{c}) \quad = \quad \frac{\Sigma_{S \in Domain(\mathcal{W})\, \wedge o \in S\, \wedge\, \mathsf{C}_S(o) = \mathsf{c}} \mathcal{P}(S)}{\Sigma_{S \in Domain(\mathcal{W})\, \wedge o \in S} \mathcal{P}(S)}.$$

Then, $\tilde{D}(\mathcal{P})$ returns a function defined as follows: for any non-leaf object o, $\tilde{D}(\mathcal{P})(o) = \omega_{\mathcal{P},o}$.

Intuitively, I construct $\omega_{\mathcal{P},o}(\mathsf{c})$ as follows. Find all semistructured instances S that are compatible with $\mathcal{W}$ and given that o occurs, find the proportion for which o's set of children is c. The sum of

[4]Here, $\mathcal{P}(c|o)$ is the probability of c being children of o given that o exists. The notation of $\mathcal{P}(c|o, A)$ means the probability of c being children of o given that o and A exists, where A is a set of objects.

the (normalized) probabilities assigned to the remaining semistructured instances by $\mathcal{P}$ is assigned to c by the OPF $\omega_{\mathcal{P},o}(\mathsf{c})$. By doing this for each non-leaf object o and each of its potential child sets, we get a local interpretation. The following theorem establishes this claim formally.

Proposition 2 *Suppose $\mathcal{P}$ is a global interpretation for a weak instance $\mathcal{W} = (V, \mathsf{lch}, \tau, \mathsf{val}, \mathsf{card})$. Then $\tilde{D}(\mathcal{P})$ is a local interpretation for $\mathcal{W}$.*

Proof:

From Definition 2.3.6, $\tilde{D}(\mathcal{P})(o) = \omega_{\mathcal{P},o}$ is an OPF for o because $\Sigma_{\mathsf{c}}\omega_{\mathcal{P},o}(\mathsf{c}) = 1$. By Definition 2.2.11, $\tilde{D}(\mathcal{P})$ is a local interpretation because for every non-leaf object o, $\tilde{D}(\mathcal{P})(o)$ returns an OPF for o. ∎

Example 3.3.2 *Consider the probabilistic instance in Example 3.3.1 and the set of compatible instances in Figure 3.2. Suppose we are given a global interpretation $\mathcal{P}$ such that $\mathcal{P}(S1) = 0.2$, $\mathcal{P}(S2) = 0.3$, $\mathcal{P}(S3) = 0.12$, $\mathcal{P}(S4) = 0.18$, $\mathcal{P}(S5) = 0.2$. Then a local probability can be obtained by calculating the probability of each potential child of every non-leaf object. For example, when we calculate the probability of $\{tank1\}$ as the actual child of $convoy1$, we notice that $S1, S2, S3, S4$ contain $convoy1$, but only the child of $convoy1$ in $S1, S3$ is $\{tank1\}$. Hence, $\tilde{D}(\mathcal{P})(convoy1)(\{tank1\})$ $= \frac{\mathcal{P}(S1)+\mathcal{P}(S3)}{\mathcal{P}(S1)+\mathcal{P}(S2)+\mathcal{P}(S3)+\mathcal{P}(S4)} = \frac{0.2+0.12}{0.2+0.3+0.12+0.18} = \frac{0.32}{0.8} = 0.4$*

The following theorems tell us that applying the two operators $\tilde{D}$ and $\tilde{W}$ one after the other (on appropriate arguments) yields no change.

Theorem 3.3 *Suppose $\wp$ is a local interpretation for a weak instance $\mathcal{W} = (V, \mathsf{lch}, \tau, \mathsf{val}, \mathsf{card})$. Then, $\tilde{D}(\tilde{W}(\wp)) = \wp$.*

Proof:

Given a graph G and a subset A of objects in G, $G - A$ is a subgraph of G after removing the set A of objects and all edges connected to A. Define $G_{ndes(o)} = G - (\{o\} \cup \mathsf{des}_G(o))$ for any acyclic directed graph G, where $\mathsf{des}_G(o)$ is the set of descendants of o in G, i.e., $G_{ndes(o)}$ is a subgraph of G without an object o and its descendants. Define $G_{des(o)} = G - (\{o\} \cup \mathsf{non\text{-}des}_G(o))$

for any acyclic directed graph G, where $\textsf{non-des}_G(o)$ is the set of non-descendants of o in G, i.e., $G_{des(o)}$ is a subgraph of G containing descendants of an object o only. In this proof, I will treat instances S and weak instances $\mathcal{W}$ as graphs and the above notations will be used on them.

Define $Domain_{ndes(o)}$ as function of $\mathcal{W}$ which returns a set of "compatible" subgraphs of $\mathcal{W}$ only containing nondescendants of an object o, i.e., $Domain_{ndes(o)}(\mathcal{W}) = \{S - o - \textsf{des}_{\mathcal{W}}(o) | S \in Domain(\mathcal{W})\}$ where $\textsf{des}_{\mathcal{W}}(o)$ are the descendants of o in $\mathcal{W}$. Similarly, define $Domain_{des(o)}$ as function of $\mathcal{W}$ which returns a set of "compatible" subgraphs of $\mathcal{W}$ only containing an object o and its descendants, i.e., $Domain_{des(o)}(\mathcal{W}) = \{S - \textsf{non-des}_{\mathcal{W}}(o) | S \in Domain(\mathcal{W})\}$ where $\textsf{non-des}_{\mathcal{W}}(o)$ are the nondescendants of o in $\mathcal{W}$.

I need to prove that for any non-leaf object o and any of its potential child sets c, $\tilde{D}(\tilde{W}(\wp))(o)(\mathsf{c}) = \wp(o)(\mathsf{c})$. The formula in Definition 3.3.3 can be rewritten as the following:

$$\tilde{W}(\wp)(S) = \alpha_S \times \wp(o)(\mathsf{C}_S(o)) \times \beta_S$$

where

$$\alpha_S = \prod_{o' \in S \wedge o' \in \mathcal{W}_{ndes(o)}} \wp(o')(\mathsf{C}_S(o')), \qquad \beta_S = \prod_{o' \in S \wedge o' \notin \mathcal{W}_{ndes(o)}} \wp(o')(\mathsf{C}_S(o')).$$

Intuitively, α_S is the product of the probabilities of children of all objects excluding o and its descendants. β_S is the product of the probabilities of children of all objects including only descendants of o. Here, descendants and nondescendants are those of the object o in $\mathcal{W}$.

It is obvious that for any two instances $S, S' \in Domain(\mathcal{W})$, if $S_{ndes(o)} = S'_{ndes(o)}$, then $\alpha_S = \alpha_{S'}$. Similarly, if $S_{des(o)} = S'_{des(o)}$, then $\beta_S = \beta_{S'}$. For any $X \in Domain_{ndes(o)}(\mathcal{W})$, I define new term $\alpha_{(X)} = \alpha_S$ if $X = S_{ndes(o)}$. Similarly, for any $Y \in Domain_{des(o)}(\mathcal{W})$, I define a new term $\beta_{(Y)} = \beta_S$ if $Y = S_{des(o)}$.

The formula in Definition 2.3.6 to compute $\omega_{\tilde{W}(\wp),o}(\mathsf{c})$ can be rewritten as:

$$\tilde{D}(\tilde{W}(\wp))(o)(\mathsf{c}) = \omega_{\tilde{W}(\wp),o}(\mathsf{c}) = \frac{a}{a+b}$$

where

$$a = \sum_{S \in Domain(\mathcal{W}) \wedge o \in S \wedge \mathsf{C}_S(o) = \mathsf{c}} \alpha_S \times \wp(o)(\mathsf{C}_S(o)) \times \beta_S$$

$$= \sum_{X \in Domain_{ndes(o)}(\mathcal{W})} \alpha_{(X)} \times \wp(o)(\mathsf{c}) \times \sum_{Y \in Domain_{des(o)}(\mathcal{W})} \beta_{(Y)}$$

(the reasoning in this step is similar to Theorem 3.2)

$$= \sum_{X \in Domain_{ndes(o)}(\mathcal{W})} \alpha_{(X)} \times \wp(o)(\mathsf{c}) \times 1,$$

(the last term can be proved similarly to Theorem 3.2)

$$b = \sum_{S \in Domain(\mathcal{W}) \wedge o \in S \wedge \mathsf{C}_S(o) \neq \mathsf{c}} \alpha_S \times \wp(o)(\mathsf{C}_S(o)) \times \beta_S$$

$$= \sum_{X \in Domain_{ndes(o)}(\mathcal{W})} \alpha_{(X)} \times \sum_{\mathsf{c}_i \in \mathsf{PC}(o) \text{ where } \mathsf{c}_i \neq \mathsf{c}} \wp(o)(c_i) \times \sum_{Y \in Domain_{des(o)}(\mathcal{W})} \beta_{(Y)}$$

(more precisely, for the last term, I require $\exists S \in Domain(\mathcal{W})$

such that $S_{\textsf{non-des}(o)} = X$ and $S_{\textsf{des}(o)} = Y$.)

$$= \sum_{X \in Domain_{ndes(o)}(\mathcal{W})} \alpha_{(X)} \times \sum_{\mathsf{c}_i \in \mathsf{PC}(o) \text{ where } \mathsf{c}_i \neq \mathsf{c}} \wp(o)(\mathsf{C}_S(o)) \times 1$$

$$= \sum_{X \in Domain_{ndes(o)}(\mathcal{W})} \alpha_{(X)} \times (1 - \wp(o)(\mathsf{c})).$$

Thus,

$$\omega_{\tilde{W}(\wp),o}(\mathsf{c}) = \frac{\wp(o)(\mathsf{c})}{\wp(o)(\mathsf{c}) + 1 - \wp(o)(\mathsf{c})} = \wp(o)(\mathsf{c}).$$

∎

The following theorem tells us that if we first apply the $\tilde{D}$ operator and then apply the $\tilde{W}$ operator to a global interpretation, then we get the global interpretation back.

Theorem 3.4 *Suppose $\mathcal{P}$ is a global interpretation for a weak instance $\mathcal{W} = (V, \mathsf{lch}, \tau, \mathsf{val}, \mathsf{card})$ and $\mathcal{P}$ satisfies $\mathcal{W}$. Then, $\tilde{W}(\tilde{D}(\mathcal{P})) = \mathcal{P}$.*

Proof:

As $\mathcal{P}$ satisfies $\mathcal{W}$ and as the probability of a child set of a given object is independent of other objects, so we can factorize $\mathcal{P}(S)$ to the product of the conditional probabilities of child sets of all non-leaf objects and there exists an OPF ω_o for every non-leaf object o such that for every $S \in Domain(\mathcal{W})$, $\mathcal{P}(S) = \prod_{o \in S} \omega_o(\mathsf{C}(o))$.

If I can prove that for any non-leaf object o and its any potential child set c, $\tilde{D}(\mathcal{P})(o)(\mathsf{c}) = \omega_o(\mathsf{c})$, then I can show that for every S, $\tilde{W}(\tilde{D}(\mathcal{P}))(S) = \prod_{o \in S} \tilde{D}(\mathcal{P})(o)(\mathsf{C}(o)) = \prod_{o \in S} \omega_o(\mathsf{C}(o))$

$= \mathcal{P}(S)$.

Now I must show that $\tilde{D}(\mathcal{P})(o)(\mathsf{c}) = \omega_o(\mathsf{c})$. I can define a local interpretation $\wp$ such that $\wp(o) = \omega_o$ for any non-leaf object o. Then, from Definition 3.3.3, $\tilde{W}(\wp)(S) = \prod_{o \in S} \wp(o)(\mathsf{C}_S(o)) = \prod_{o \in S} \omega_o(\mathsf{C}(o)) = \mathcal{P}(S)$. Thus, $\tilde{W}(\wp) = \mathcal{P}$. As a result, I can substitute $\mathcal{P}$ by $\tilde{W}(\wp)$ in $\tilde{D}(\mathcal{P})(o)(\mathsf{c}) = \omega_o(\mathsf{c})$, then what I now need to prove has become: for any non-leaf object o and its any potential child set c, $\tilde{D}(\tilde{W}(\wp))(o)(\mathsf{c}) = \wp(o)(\mathsf{c})$. This is exactly the same as that in the proof of Theorem 3.3.

■

3.3.2 Satisfaction

I am now ready to address the important question of when a local (resp. global) interpretation satisfies a probabilistic instance.

A probabilistic instance imposes constraints on the probability specifications for objects. I associate a set of object constraints with each non-leaf object as follows.

Definition 3.3.6 (object constraints) *Suppose $\mathcal{I} = (V, \mathsf{lch}, \tau, \mathsf{val}, \mathsf{card}, \mathsf{ipf})$ is a probabilistic instance, $o \in V$ is a non-leaf object. We associate with o, a set of constraints called **object constraints**, denoted $\mathsf{OC}(o)$, as follows. For each $\mathsf{c} \in \mathsf{PC}(o)$, $\mathsf{OC}(o)$ contains the constraint $\mathsf{ipf}(o, \mathsf{c}).lb \leq p(\mathsf{c}) \leq \mathsf{ipf}(o, \mathsf{c}).ub$ where $p(\mathsf{c})$ is a real-valued variable denoting the probability that c is the actual set of children of o. $\mathsf{OC}(o)$ also includes the following constraint $\Sigma_{\mathsf{c} \in \mathsf{PC}(o)} p(\mathsf{c}) = 1$.*

Example 3.3.3 *Consider the probabilistic instance defined in Example 3.2.1. $\mathsf{OC}(I1)$ is defined as follows: $0.2 \leq p(\{convoy1\}) \leq 0.4$, $0.1 \leq p(\{convoy2\}) \leq 0.4$, $0.4 \leq p(\{convoy1, convoy2\}) \leq 0.7$, and $p(\{convoy1\}) + p(\{convoy2\}) + p(\{convoy1,\ convoy2\}) = 1$.*

Intuitively, an OPF satisfies a non-leaf object iff the assignment made to the potential children by the OPF is a solution to the constraints associated with that object. Obviously, a probability distribution w.r.t. $\mathsf{PC}(o)$ over ipf is a solution to $\mathsf{OC}(o)$.

Definition 3.3.7 (object satisfaction) *Suppose $\mathcal{I} = (V, \mathsf{lch}, \tau, \mathsf{val}, \mathsf{card}, \mathsf{ipf})$ is a probabilistic instance, $o \in V$ is a non-leaf object, ω is an OPF for o, and $\wp$ is a local interpretation. ω **satisfies** o iff ω is a probability distribution w.r.t. $\mathsf{PC}(o)$ over ipf. $\wp$ satisfies o iff $\wp(o)$ satisfies o.*

Example 3.3.4 *Consider the probabilistic instance defined in Example 3.2.1, the probability interpretation defined in Example 3.3.1 and the* $\mathsf{OC}(I1)$ *defined in Example 3.3.3. Since the assignment made to the potential children of* $I1$ *by the OPF* $\wp(I1) = \omega_{I1}$ *is a solution to the constraints* $\mathsf{OC}(I1)$ *associated with* $I1$*,* ω_{I1} *is a probability distribution w.r.t.* $\mathsf{PC}(I1)$ *over* ipf*. Thus,* ω *satisfies* $I1$ *and the local interpretation* $\wp$ *satisfies* $convoy$*. Similarly,* $convoy1$ *and* $convoy2$ *are satisfied.*

I am now ready to extend the above definition to the case of satisfaction of a probabilistic instance by a local interpretation.

Definition 3.3.8 (local satisfaction of a prob. inst.) *Suppose* $\mathcal{I} = (V, \mathsf{lch}, \tau, \mathsf{val}, \mathsf{card}, \mathsf{ipf})$ *is a probabilistic instance, and* $\wp$ *is a local interpretation.* $\wp$ ***satisfies*** $\mathcal{I}$ *iff for every non-leaf object* $o \in V$*,* $\wp(o)$ *satisfies* o*.*

Example 3.3.5 *Consider the probabilistic instance defined in Example 3.2.1, the local interpretation* $\wp$ *defined in Example 3.3.1. In view of the fact that* $\wp$ *satisfies all three non-leaf objects,* $I1$*,* $convoy1$ *and* $convoy2$*, it follows that* $\wp$ *satisfies the example probabilistic instance.*

Similarly, a global interpretation $\mathcal{P}$ satisfies a probabilistic instance if the OPF computed by using $\mathcal{P}$ can satisfy the object constraints of each non-leaf object.

Definition 3.3.9 (global satisfaction of a prob. inst.) *Suppose* $\mathcal{I} = (V, \mathsf{lch}, \tau, \mathsf{val}, \mathsf{card}, \mathsf{ipf})$ *is a probabilistic instance, and* $\mathcal{P}$ *is a global interpretation.* $\mathcal{P}$ *satisfies* $\mathcal{I}$ *iff for every non-leaf object* $o \in V$*,* $\tilde{D}(\mathcal{P})(o)$ *satisfies* o*, i.e.,* $\tilde{D}(\mathcal{P})$ *satisfies* $\mathcal{I}$*.*

Corollary 1 (equivalence of local and global sat.) *Suppose* $\mathcal{I} = (V, \mathsf{lch}, \tau, \mathsf{val}, \mathsf{card}, \mathsf{ipf})$ *is a probabilistic instance, and* $\wp$ *is a local interpretation. Then* $\wp$ *satisfies* $\mathcal{I}$ *iff* $\tilde{W}(\wp)$ *satisfies* $\mathcal{I}$*.*

Proof:

By Definition 3.3.9, $\tilde{W}(\wp)$ satisfies $\mathcal{I}$ iff $\tilde{D}(\tilde{W}(\wp))$ satisfies $\mathcal{I}$. By Theorem 3.3, $\tilde{D}(\tilde{W}(\wp)) = \wp$. Thus, it is trivial that the corollary is true. ∎

We say a probabilistic instance is ***globally (resp. locally) consistent*** iff there is at least one global (resp. local) interpretation that satisfies it. Using Lemma 1, I can prove the following theorem saying that according to my definitions, all probabilistic instances are guaranteed to be globally (and locally) consistent.

Theorem 3.5 *Every probabilistic instance is both globally and locally consistent.*

Proof:

By Definition 3.3.9 and Corollary 1, a probabilistic instance is globally consistent iff it is locally consistent. Thus, I only need to prove that every probabilistic instance is locally consistent. Suppose $\mathcal{I} = (V, \mathsf{lch}, \tau, \mathsf{val}, \mathsf{card}, \mathsf{ipf})$ is a probabilistic instance. For every non-leaf object $o \in V$, $\mathsf{OC}(o)$ are exactly the same constraints of the definition of a probability distribution w.r.t. $\mathsf{PC}(o)$ over $\mathsf{ipf}(o, \mathsf{c})$. By Lemma 1, there exists such a probability distribution P, so I can define an OPF ω_o for o such that $\forall \mathsf{c} \in \mathsf{PC}(o)$, $\omega_o(\mathsf{c}) = P(\mathsf{c})$. ω_o is a probability distribution w.r.t. $\mathsf{PC}(o)$ over ipf w.r.t. $\mathsf{PC}(o)$, so it satisfies o. Thus, for each non-leaf object $o \in V$, I can define an OPF ω_o that satisfies o. Then I can define a local interpretation $\wp$ such that for every non-leaf object $o \in V$, $\wp(o) = \omega_o$. Therefore, for every non-leaf object $o \in V$, $\wp(o)$ satisfies o, so $\wp$ satisfies $\mathcal{I}$. ∎

3.4 PIXML Queries: Syntax

In this section, I define the formal syntax of a PIXML query. The important concept of a *path expression* defined below (a bit different from Definition 2.5.1) plays the role of an attribute name in the relational algebra.

Definition 3.4.1 (path expression) *Suppose $\mathcal{I}$ is a probabilistic instance, o_{pr} is an object id, and $l_1, l_2, \ldots, l_n$ are labels of edges. Then $\mathcal{I} : o_{pr}$ and $\mathcal{I} : o_{pr}.l_1 \ldots l_n$ are called* path expressions. *When $\mathcal{I}$ is clear from context, I will often just write o_{pr} and $o_{pr}.l_1 \ldots l_n$.*

I write $o \in p$ whenever object o can be located by path p. A path expression is ***instance-specific*** iff the path expression is specified with an instance name of the form $\mathcal{I} : p$ where $\mathcal{I}$ is an instance name and p is a path expression. When a path expression is not instance-specific, it refers to a default instance.

3.4.1 Single-Instance Queries

A single instance query is one that only accesses one probabilistic instance.

Given an attribute name A in the relational algebra, we can write queries such as $A = v, A \geq v$, etc. We can do the same with path expressions.

Definition 3.4.2 (atomic query) *An **atomic query** has one of the following forms:*

1. *$p\ \phi\ o$, where p is a path expression, ϕ is a binary predicate from $\{=, \neq\}$ and o is an oid.*
2. *$\mathsf{val}(p)\ \phi\ v$, where p is a path expression, ϕ is a binary predicate from $\{=, \neq, \leq, \geq, <, >\}$ and v is a value.*
3. *$\mathsf{card}(p, x)\ \phi\ I$, where p is a path expression, x is either $l \in \mathcal{L}$ or $\star$ (a wildcard matches any label), ϕ is a binary predicate from $\{=, \neq, \leq, \geq, <, >, \subset, \subseteq, \supset, \supseteq, \}$ and I is an interval. $I_1\ \phi\ I_2$ has the intended interpretation. For example, $I_1 > I_2$ means $I_1.lb > I_2.lb \wedge I_1.ub > I_2.ub$.*
4. *$\mathsf{ipf}(p, x)\ \phi\ I$, where p is a path expression, x is either $\mathsf{c} \in \mathsf{PC}(p)$ or the wildcard $\star$ (which means that it matches any potential child), ϕ is a binary predicate from $\{=, \neq, \leq, \geq, <, >, \subset, \subseteq, \supset, \supseteq, \}$ and I is an interval $\subseteq [0, 1]$.*
5. *$operand_1\ \phi\ operand_2$, where both $operand_1$ and $operand_2$ should be of the same form among $p, \mathsf{val}(p)$, $\mathsf{card}(p,\ x)$ and $\mathsf{ipf}(p, x)$ defined above; ϕ is a corresponding binary predicate defined above.*

I assume that an order is defined on the elements in the domain of a type, but some types such as strings only allow operations in $\{=, \neq\}$. An atomic selection expression of the form $\mathsf{val}(p)\ \phi\ v$ or $\mathsf{val}(p_1)\ \phi\ \mathsf{val}(p_2)$ is satisfied only if both sides of the binary predicate are type-compatible and compatible with ϕ (i.e., ϕ is defined on their types).

A compound query is a boolean combination of atomic queries.

Definition 3.4.3 *$(q_1 \wedge q_2)$ and $(q_1 \vee q_2)$ are compound queries if q_1, q_2 are atomic queries or compound queries.*

The simplest kind of query is one where all the conditions in the query can be satisfied (I will formally define satisfaction later) by one object in one instance. This kind of query can be either an atomic query with just one path expression (like a free variable for only one object as the answer) or a compound query with all path expressions referring to the same object (this can be done by using an object variable, e.g., $val(w = p_1) > v_1 \wedge val(w) < v_2$ where w is an object variable, p_1 is a path expression, v_1, v_2 are values).

Definition 3.4.4 *A* ***single-instance-single-object (SISO) query*** *is either (i) an atomic query (form (1)–(4) in Definition 3.4.2), or (ii) a compound query as a result of a boolean combination of atomic queries (form (1)–(4) in Definition 3.4.2) where all the path expressions in the query refer to the same object by using an object variable.*

In contrast, the following defines a query that can be satisfied by more than one object in one instance.

Definition 3.4.5 *A* ***single-instance-multiple-object (SIMO) query*** *is either (i) an atomic query (form (5) in Definition 3.4.2), or (ii) a compound query.*

3.4.2 Multiple-Instance Queries

In the previous section, only one probabilistic instance is queried at a time. It is straight forward to extend the above syntax to handle multiple probabilistic instances.

When the path expressions are specified with instance names, we can query multiple instances in a way similar to SIMO queries.

Definition 3.4.6 *A* ***multiple-instance-multiple-object (MIMO) query*** *is either (i) an atomic query (form (5) in Definition 3.4.2), or (ii) a compound query where all the path expressions are instance-specific and involve more than one instance. Furthermore, it is called an* ***independent MIMO*** *(IMIMO) if and only if every atomic query in an MIMO query involves only one instance AND the atomic queries can be rearranged into the form* $(q_{1,1}\ \gamma\ q_{1,2}\ \gamma\ \ldots\ \gamma\ q_{1,n_1})\ \gamma\ (q_{2,1}\ \gamma\ q_{2,2}$

*$\gamma \ \ldots \ \gamma \ q_{2,n_2}) \ \gamma \ \ldots \ \gamma \ (q_{m,1} \ \gamma \ q_{m,2} \ \gamma \ \ldots \ \gamma \ q_{m,n_m})$, where γ^5 is either $\wedge$ or $\vee$, and $q_{i,j}$ are atomic queries involving only instance $\mathcal{I}_i$. Otherwise, it is called a **dependent MIMO** (DMIMO).*

Suppose $\mathcal{I}_1, \mathcal{I}_2$ are instance names, p_1, p_2, p_3, p_4 are path expressions. Then $(val(\mathcal{I}_1 : p_1) > val(\mathcal{I}_1 : p_2) \vee val(\mathcal{I}_1 : p_1) < 100) \wedge val(\mathcal{I}_2 : p_3) = val(\mathcal{I}_2 : p_4)$ is IMIMO while $val(\mathcal{I}_1 : p_1) > val(\mathcal{I}_2 : p_2)$ is DMIMO and $(val(\mathcal{I}_1 : p_1) > val(\mathcal{I}_1 : p_2) \vee val(\mathcal{I}_2 : p_3) = val(\mathcal{I}_2 : p_4)) \wedge val(\mathcal{I}_1 : p_1) < 100$ is also DMIMO

A query is reduced to SISO or SIMO if only one instance is involved. For example, $val(\mathcal{I}_1 : p_1) = val(\mathcal{I}_1 : p_2)$ is reduced to SIMO query $val(p_1) = val(p_2)$ on $\mathcal{I}_1$.

Note that it is possible to rearrange the atomic queries in a DMIMO query to become a boolean combination of groups of atomic queries. Each group is a boolean combination of atomic queries and the instances involved in this group are not involved in any other groups. In this case, we can solve each group by algorithms for IMIMO or DMIMO (depending on whether the group is IMIMO or DMIMO) and then combine the result by the strategies described in the algorithm for IMIMO. In this way, the complexity and actual running time can be reduced significantly. The algorithms and strategies for IMIMO and DMIMO queries will be described in later sections (Section 3.6.3 and Section 3.6.4).

3.5 PIXML Queries: r-Answers

3.5.1 r-Answers to SISO queries

First, let us consider the answer to an SISO query.

In order to define the ***answer*** to an SISO query w.r.t. a probabilistic instance, I proceed in two steps. I first define what it means for an object to satisfy an SISO query. I then define what the answer to an SISO query is w.r.t. a probabilistic instance.

Definition 3.5.1 (satisfaction of an SISO query by an object) *An object o_1 **satisfies** an atomic SISO query Q via substitution $\theta(p/o_1)$ (denoted $o_1 \models Q$) if and only if $o_1 \in p$ where p is*

[5]Here, I assume that the operator γ has a specified order of precedence, so I do not put parentheses to specify the order of computation of the query.

the path expression in Q.

An object o_1 ***satisfies*** *a compound SISO query if and only if $o_1 \in p_i$ for every path expression p_i in Q and Q is true under substitution $\theta = \{p_1/o_1, p_2/o_1, \ldots\}$.*

In order to define the *answer* to a query, I must account for the fact that a given probabilistic instance is compatible with many semistructured instances. The *probability* that an object is an answer to the query is determined by the probabilities of all the compatible semistructured instances that it occurs in.

Definition 3.5.2 (satisfaction of an SISO query by an object with prob r) *Suppose $\mathcal{I}$ is a probabilistic instance, Q is an SISO query, and $\mathcal{S} \in Domain(\mathcal{I})$. I say that object o of $\mathcal{I}$* ***satisfies query Q with probability r or more****, denoted $o \models^r Q$ iff*

$$r \leq \mathbf{INF}\{\Sigma_{\mathcal{S} \in Domain(\mathcal{I}) \,\wedge\, o \in \mathcal{S} \,\wedge\, o \models Q} \mathcal{P}(\mathcal{S}) \mid \mathcal{P} \models \mathcal{I}\}.$$

Intuitively, any global interpretation[6] $\mathcal{P}$ assigns a probability to each semistructured instance $\mathcal{S}$ compatible with a probabilistic instance $\mathcal{I}$. By summing up the probabilities of the semistructured instances in which object o occurs, we obtain an ***occurrence probability*** for o in $\mathcal{I}$ w.r.t. global interpretation $\mathcal{P}$. However, different global interpretations may assign different probabilities to compatible semistructured instances. Hence, if we examine all global interpretations that satisfy $\mathcal{I}$ and take the **INF** (infimum) of the occurrence probability of o w.r.t. each such satisfying global interpretation, then we are guaranteed that for all global interpretations, the probability of o's occurrence is at least the **INF** obtained in this way. This provides the rationale for the above definition. The answer to a query may be defined in many ways. My standard norm will be to assume that the user sets a real number $0 < r \leq 1$ as a threshold and that only objects satisfying a query with probability r or more will be returned.

Definition 3.5.3 (r-answer) *Suppose $0 < r \leq 1$. The* ***r-answer*** *to SISO query Q is the set of all objects o such that $o \models^r Q$.*

[6]Note that given a probabilistic instance $\mathcal{I}$, I will only consider global interpretations that satisfies the weak instance corresponding to $\mathcal{I}$ (by Definition 3.3.4).

Example 3.5.1 *Consider the probabilistic instance in Example 2.2.3 and a query Q which is* $\mathsf{val}($ *I1.convoy.tank* $) = T80$*. Suppose we want to find a* 0.4*-answer to* Q*. Obviously, the only possible object to satisfy* Q *is* $tank1$*. However, there exists a global interpretation (for example, the one in Example 3.3.2) such that the sum of the probability of compatible instances containing* $tank1$ *satisfying* Q *is less than* 0.4*. Thus, the* 0.4*-answer to* Q *is empty.*

3.5.2 r-Answers to SIMO/MIMO queries

For multiple-object queries (including SIMO and MIMO queries), it is possible for an atomic query to have form (5) of Definition 3.4.2, i.e., two objects are necessary to satisfy it. For example, in $\mathsf{val}(p_1) = \mathsf{val}(p_2)$ where p_1, p_2 are path expressions, only a pair of objects satisfying each path expression and having the same value can satisfy this query.

Definition 3.5.4 *Objects* o_1 *and* o_2 *satisfy an atomic query* $(p_1\ \phi\ p_2)$ *via substitution* $\theta = \{p_1/o_1, p_2/o_2\}$ *if and only if* $(o_1 \in p_1 \wedge o_2 \in p_2 \wedge o_1\ \phi\ o_2)$ *holds. Similarly for other forms of the two operands.*

Instead of talking about *an object* satisfying an SISO query, I will need to define when *a set of objects* satisfies an SIMO or MIMO query. I say a set of objects $T = \{o_1, \ldots, o_n\}$ satisfies the query with a substitution $\theta_T = \{p_1/o_1, \ldots, p_n/o_n\}$ if all the path expressions are satisfied and the result of the boolean combination of the atomic queries returns true. For example, we are given a query Q defined as $val(w = p_1) > val(p_2) \wedge val(w) < val(p_3)$ (where p_1, p_2, p_3 are path expressions and w is an object variable) that is used to find object sets such that an object satisfying p_1 has a value lying between the values of some object satisfying p_2 and some object satisfying p_3. Here I say that a set of objects $T = \{o_1, o_2, o_3\}$ satisfies Q by a substitution $\theta_T = \{p_1/o_1, p_2/o_2, p_3/o_3\}$ if and only if $o_1 \in p_1$, $o_2 \in p_2$, $o_3 \in p_3$ and $((val(o_1) > val(o_2)) \wedge (val(o_1) < val(o_3)))$ is true.

Definition 3.5.5 (satisfaction of an SIMO/MIMO query by a set of objects) *A set* $T = \{o_1, \ldots, o_n\}$ *of objects* ***satisfies*** *an SIMO/MIMO query* Q *(denoted* $T \models Q$*) via substitution* $\theta_T = \{p_1/o_1, \ldots, p_n/o_n\}$ *if and only if any one of the following conditions is satisfied:*

- *Q is an atomic query (form (5) in Definition 3.4.2), $T = \{o_1, o_2\}$, o_1 and o_2 satisfy Q via θ_T;*
- *Q is a compound query in the form of $q_1 \wedge q_2$ (where q_1, q_2 are atomic queries or compound queries), and T satisfies q_1 and q_2 via θ_T;*
- *Q is a compound query in the form of $q_1 \vee q_2$ (where q_1, q_2 are atomic queries or compound queries), and (i) T satisfies q_1 via θ_T or (ii) T satisfies q_2 via θ_T.*

The definitions of satisfaction with probability r and r-answer for SIMO/MIMO queries are similar to those for SISO queries.

Definition 3.5.6 (satisfaction of an SIMO query by a set with prob r) *Suppose $\mathcal{I}$ is a probabilistic instance, Q is an SIMO query, and $\mathcal{S} \in Domain(\mathcal{I})$. I say that the set $T = \{o_1, \ldots, o_n\}$ of objects in $\mathcal{I}$* ***satisfies query Q with probability*** *r* ***or more under substitution*** *$\theta_T = \{p_1/o_1, \ldots, p_n/o_n\}$, denoted $T \models^r Q$, iff*

$$r \leq \mathbf{INF}\{\Sigma_{\mathcal{S} \in Domain(\mathcal{I}) \,\wedge\, T \subseteq \mathcal{S} \,\wedge\, T \models Q} \mathcal{P}(\mathcal{S}) \mid \mathcal{P} \models \mathcal{I}\}.$$

The major difference of MIMO queries from SIMO queries is that we have multiple probabilistic instances $\mathcal{I}_1, \ldots, \mathcal{I}_m$. Suppose T is the set of objects satisfying a given MIMO query over probabilistic instances $\mathcal{I}_1, \ldots, \mathcal{I}_m$, and suppose T_i consists of all objects in T that are in $\mathcal{I}_i$. The occurrence probability for T w.r.t. $\mathcal{I}_1, \ldots, \mathcal{I}_m$ and a sequence $\mathcal{P}_1, \ldots, \mathcal{P}_m$ of global interpretations (where $\mathcal{P}_i$ satisfies $\mathcal{I}_i$) is the combination of the probabilities assigned by the $\mathcal{P}_i$'s to the T_i's. Hence, I need to examine all combinations of global interpretations and take the **INF** of T's occurrence probabilities w.r.t. each such combination.

[50] proposed the concept of a conjunctive strategy used for conjunction of interval probabilities. Intuitively, a conjunction strategy is a function satisfying several axioms (shown in Figure 3.3) related to conjunction which takes probability intervals associated with the conjuncts in a conjunction and finds a probability for the conjunction. Based on the user's knowledge of the dependencies between the conjunctions, an appropriate conjunction strategy can be chosen –

[50] suggest several (cf. Figure 3.4). The strategies for interval probabilities will be used again when we discuss how to solve MIMO queries in later sections. Here in the definition for r-answer of MIMO queries, as I need to obtain the **INF** of the conjunction of point probabilities only, I can transform each point probability p to an interval probability $[p, p]$ and take the lower bound of a specific conjunctive strategy as the value of the conjunction.

Generic Postulates ($\star \in \{\otimes, \oplus\}$)	
1. Commutativity	$([l_1, u_1] \star [l_2, u_2]) = ([l_2, u_2] \star [l_1, u_1])$
2. Associativity	$(([l_1, u_1] \star [l_2, u_2]) \star [l_3, u_3]) = ([l_1, u_1] \star ([l_2, u_2] \star [l_3, u_3]))$
3. Monotonicity	$([l_1, u_1] \star [l_2, u_2]) \leq ([l_1, u_1] \star [l_3, u_3])$ if $[l_2, u_2] \leq [l_3, u_3]$
Probabilistic Conjunction Postulates	
4. Bottomline	$([l_1, u_1] \otimes [l_2, u_2]) \leq [min(l_1, l_2), min(u_1, u_2)]$
5. Identity	$([l_1, u_1] \otimes [1, 1]) = [l_1, u_1]$
6. Annihilator	$([l_1, u_1] \otimes [0, 0]) = [0, 0]$
7. Ignorance	$([l_1, u_1] \otimes [l_2, u_2]) \subseteq [max(0, l_1 + l_2 - 1), min(u_1, u_2)]$

Figure 3.3: Postulates of a probabilistic conjunctive strategy

Probabilistic Conjunctive Strategies	
Ignorance	$([l_1, u_1] \otimes_{ig} [l_2, u_2]) = [max(0, l_1 + l_2 - 1), min(u_1, u_2)]$
Positive Correlation	$([l_1, u_1] \otimes_{pc} [l_2, u_2]) = [min(l_1, l_2), min(u_1, u_2)]$
Negative Correlation	$([l_1, u_1] \otimes_{nc} [l_2, u_2]) = [max(0, l_1 + l_2 - 1), max(0, u_1 + u_2 - 1)]$
Independence	$([l_1, u_1] \otimes_{in} [l_2, u_2]) = [l_1 \cdot l_2, u_1 \cdot u_2]$

Figure 3.4: Examples of probabilistic conjunctive strategies

Definition 3.5.7 (satisfaction of an MIMO query by a set with prob r) *Suppose $\mathcal{I}_1, \ldots \mathcal{I}_m$ are probabilistic instances, Q is a MIMO query, and $\mathcal{S}_i \in Domain(\mathcal{I}_i)$. I say that the object set $T = \{o_1, \ldots, o_n\} \subseteq \mathcal{I}_1 \cup \ldots \cup \mathcal{I}_m$* ***satisfies query Q with probability r or more under sub-***

stitution *$\theta_T = \{p_1/o_1, \ldots, p_n/o_n\}$, denoted $T \models^r Q$, iff*

$$r \leq \mathbf{INF}\{Conj_i(\Sigma_{\mathcal{S}_i \in Domain(\mathcal{I}_i) \wedge (T \cap \mathcal{I}_i) \in \mathcal{S}_I \wedge T \models Q} \mathcal{P}_i(\mathcal{S})) \mid \mathcal{P}_i \models \mathcal{I}_i\}.$$

where $Conj_i(x_i)$ returns the conjunction of values $x_1, \ldots, x_m$ using a specified conjunctive strategy.[7]

Definition 3.5.8 (r-answer) *Suppose $0 < r \leq 1$. The r-**answer** to SIMO/MIMO query Q is the set of all object sets $T = \{o_1, \ldots, o_n\}$ and their corresponding substitutions $\theta_T = \{p_1/o_1, \ldots, p_n/o_n\}$ such that $T \models^r Q$.*

3.6 PIXML Queries: Operational Semantics

3.6.1 Algorithm to solve SISO queries

In this section, I study the problem: ***Given a probabilistic instance $\mathcal{I}$, a real number $0 < r \leq 1$, and an SISO query Q, how do we find all objects o in the probabilistic instance that satisfy query Q with probability r or more?***

Clearly, we do not wish to solve this problem by explicitly finding all global interpretations that satisfy $\mathcal{I}$ and explicitly computing the sum on the right side of the inequality in Definition 3.5.2. This approach is problematic for many reasons, the first of which is that there may be infinitely many global interpretations that satisfy $\mathcal{I}$. I present a more practical solution.

Recall that the weak instance graph $\mathcal{G}_{\mathcal{W}}$ describes all the potential children of an object. A probabilistic instance is said to be ***tree-structured*** iff its corresponding weak instance graph is a tree (i.e. the vertices and edges of $\mathcal{G}_{\mathcal{W}}(\mathcal{I})$ constitute a tree). Throughout the rest of this section, I assume that we are only dealing with tree structured probabilistic instances.

My algorithm to solve this problem involves two core steps.

- **Step 1:** The first step is to identify all objects o in the weak instance graph $\mathcal{G}_{\mathcal{W}}$ of $\mathcal{I}$ that

[7]Here, for each probabilistic instance $\mathcal{I}_i$, $\Sigma_{\mathcal{S}_i \in Domain(\mathcal{I}_i) \wedge (T \cap \mathcal{I}_i) \in \mathcal{S}_i \wedge T \models Q} \mathcal{P}_i(\mathcal{S})$ gives the probability that $\mathcal{I}_i$ contains the objects in T intersecting $\mathcal{I}_i$, for a particular global interpretation $\mathcal{P}_i$. Because I am considering a combination (hitting set) (of a possible global interpretation of each probabilistic instance), I need a conjunction of the above probabilities from all probabilistic instances (details in Section 3.6.3).

satisfy the query Q. This can be done using any available implementation of semistructured databases[59], hence I do not present the details here.

- **Step 2:** The second step is to check (using the original probabilistic instance $\mathcal{I}$ rather than its weak instance graph) which of the objects returned in the preceding step have an occurrence probability that exceeds the threshold w.r.t. all global interpretations that satisfy the original probabilistic instance.

In this section, I focus on Step 2 as Step 1 can be readily solved using existing techniques for pure, non-probabilistic semistructured databases. To solve step 2, we must have the ability to find the minimal occurrence probability of the type described earlier for o (given an o that passes the test of step 1 above).

Suppose $\mathcal{I} = (V, \mathsf{lch}, \tau, \mathsf{val}, \mathsf{card}, \mathsf{ipf})$ and suppose $o \in V$. I define a quantity $\mathsf{cex}(o)$ as follows. Intuitively, $\mathsf{cex}(o)$ finds the ***conditional probability*** of occurrence of o, given that o's parent in the tree is known to occur.

1. If o is the root of $\mathcal{G}_{\mathcal{W}}$, then $\mathsf{cex}(o) = 1$.
2. Otherwise, $o \in \mathsf{lch}(o', l)$ for some o' and some l. Recall the object constraints $\mathsf{OC}(o')$ defined in Definition 3.3.6. Set $\mathsf{cex}(o)$ to be the result of solving the linear programming problem:

   minimize $\Sigma_{\mathsf{c} \in \mathsf{PC}(o') \wedge o \in \mathsf{c}} p(\mathsf{c})$

   subject to $\mathsf{OC}(o')$,

   where $p(\mathsf{c})$ is a real-valued variable denoting the probability that c is the actual set of children of o'. As usual I assume that all the variables $p(\mathsf{c}) \geq 0$.

Intuitively, the quantity $\mathsf{cex}(o)$ specifies the smallest occurrence probability of object o *given that* its parent is already known to occur in a semistructured instance.

Note: though $\mathsf{OC}(o')$ appears to contain exponentially many variables, this is in reality linear in the number of potential children of the object o' and hence linear in the size of ipf (and hence linear in the size of the probabilistic instance).

Proposition 3 *For any probabilistic instance $\mathcal{I}$ and any object o, $\mathsf{cex}(o)$ can be computed in time polynomial in the size of $\mathcal{I}$.*

Suppose the path expression to object o in $\mathcal{I}$ is of the form $r.l_1 \ldots l_n$ where r is the root, and $l_1, \ldots, l_n$ are labels. Let us now use the object id o_n to denote o, and object id o_{i-1} to denote the parent of o_i where $1 \leq i \leq n$. Then the ***computed occurrence probability,*** $\mathsf{cop}(o)$ ***of*** o ***in*** $\mathcal{I}$ is given by $\mathsf{cex}(o_0) \cdots \mathsf{cex}(o_n)$.

Note: it is not necessary for us to always compute all $\mathsf{cex}(o_i)$ before finding that $\mathsf{cop}(n)$ is less than the threshold. Since $\mathsf{cop}(n)$ is computed by multiplying all $\mathsf{cex}(o_i)$, when we are computing $\mathsf{cex}(o_i)$ one by one, we can stop as soon as their product is already less than the threshold. This pruning technique applies to all algorithms in this chapter.

The following theorem says that the occurrence probability of o (which is defined declaratively in Definition 3.5.2) corresponds exactly to the computed occurrence probability of o according to the above procedure.

Theorem 3.6 (correctness theorem) *Suppose $\mathcal{I} = (V, \mathsf{lch}, \tau, \mathsf{val}, \mathsf{card}, \mathsf{ipf})$ is a tree structured probabilistic instance and $o \in V$. Then,* $\mathsf{cop}(o) = \mathbf{INF}\{\Sigma_{\mathcal{S} \in Domain(\mathcal{I}) \wedge o \in \mathcal{S}} \mathcal{P}(S) \mid \mathcal{P} \models \mathcal{I}\}$.

Proof

I provide the proof of this theorem. Suppose the path expression to object o in $\mathcal{I}$ is of the form $r.l_1 \ldots l_n$ where r is the root's object id, and $l_1, \ldots, l_n$ are labels. Let us now use the object id o_n to denote o, and object id o_{i-1} to denote the parent of o_i where $1 \leq i \leq n$. The proof proceeds by induction on n.

If $n = 0$ then o is the root of $\mathcal{I}$. Hence, $\mathsf{cop}(o) = 1$ by definition. As every compatible semistructured instance contains the root, it follows by the definition of probabilistic interpretation that every probabilistic interpretation assigns 1 to $\Sigma_{\mathcal{S} \in Domain(\mathcal{I}) \wedge o \in \mathcal{S}} \mathcal{P}(S)$. This completes the base case.

Suppose $n = m + 1$ for some integer m. Consider the path $o_1.l_2 \ldots l_m l_{m+1}$. Clearly, this is a path of length m from o_1 to $o_{m+1} = o$. Let $\mathcal{I}'$ be the probabilistic instance which is just like the subtree (of $\mathcal{I}$) rooted at o_1. By the induction hypothesis, I know that $\mathsf{cop}_{\mathcal{I}'}(o) = \mathbf{INF}\{\Sigma_{\mathcal{S}' \in Domain(\mathcal{I}') \wedge o \in \mathcal{S}' \wedge \mathcal{P}' \models \mathcal{I}'} \mathcal{P}'(S')\}$ where $\mathcal{P}'$ is defined as follows. If $\mathcal{S}'$ is compatible with $\mathcal{I}'$ then $\mathcal{P}'(\mathcal{S}')$ is the sum of $\mathcal{P}(\mathcal{S})$ for all $\mathcal{S}$ compatible with $\mathcal{I}$ such that $\mathcal{S}'$ is the subtree of $\mathcal{S}$ rooted at o_1.

Note that $\mathsf{cop}_{\mathcal{I}}(o) = \mathsf{cex}(o_0) \times \mathsf{cop}_{\mathcal{I}'}(o_1)$ by definition of cop. From the above, I may infer that $\mathsf{cop}(o) = \mathsf{cex}(o_0) \times \mathbf{INF}\{\Sigma_{\mathcal{S}' \in Domain(\mathcal{I}') \wedge o \in \mathcal{S}' \wedge \mathcal{P}' \models \mathcal{I}'} \mathcal{P}'(S')\}$. I therefore need to show that

$$\mathsf{cex}(o_0) \times \mathbf{INF}\{\Sigma_{\mathcal{S}' \in Domain(\mathcal{I}') \wedge o \in \mathcal{S}' \wedge \mathcal{P}' \models \mathcal{I}'} \mathcal{P}'(S')\} = \mathbf{INF}\{\Sigma_{\mathcal{S} \in Domain(\mathcal{I}) \wedge o \in \mathcal{S} \wedge \mathcal{P} \models \mathcal{I}} \mathcal{P}(S)\}.$$

But $\mathsf{cex}(o_0) = 1$ by definition, so I need to show that

$$\mathbf{INF}\{\Sigma_{\mathcal{S}' \in Domain(\mathcal{I}') \wedge o \in \mathcal{S}' \wedge \mathcal{P}' \models \mathcal{I}'} \mathcal{P}'(S')\} = \mathbf{INF}\{\Sigma_{\mathcal{S} \in Domain(\mathcal{I}) \wedge o \in \mathcal{S} \wedge \mathcal{P} \models \mathcal{I}} \mathcal{P}(S)\}.$$

The reason this equality holds is because of the construction of $\mathcal{P}'$. Consider each compatible instance $\mathcal{S} \in Domain(\mathcal{I})$ that contains o. The interpretation $\mathcal{P}'$ looks at each $\mathcal{S}'$ obtained by restricting $\mathcal{S}$ to the subtree rooted at o_1. A single $\mathcal{S}$ generates only one $\mathcal{S}'$ in this way, but the same $\mathcal{S}'$ may be obtained from different semistructured instances $\mathcal{S}$ compatible with $\mathcal{I}$. The sum of all such $\mathcal{P}'(\mathcal{S}')$ clearly equals the sum of all such $\mathcal{P}(\mathcal{S})$ containing o by construction. Furthermore, $\mathcal{P}$ satisfies $\mathcal{I}$ iff its corresponding $\mathcal{P}$" satisfies $\mathcal{I}'$. Since both sums are equal to each other for every $\mathcal{P}$ satisfying $\mathcal{I}$, the infimum of one sum is also equal to the infimum of the other sum. This completes the proof. ∎

It is important to note that the above condition is very similar to the condition on the right side of the inequality of Definition 3.5.2 (it is exactly the same if we only consider objects o that satisfy the query). ***An immediate consequence of the above theorem is that the two-step procedure outlined in this section is correct, as step 2 is only applied to objects that satisfy the SISO query condition Q.***

Example 3.6.1 *Consider the probabilistic instance in Example 3.2.1 and a compound SISO query Q which is* ($\mathsf{val}(\ w = I1.convoy.tank\ ) = T80 \vee \mathsf{val}(\ w\ ) = T72$). *This query indicates that we want only one object whose value is either $T80$ or $T72$. Suppose we want to find a 0.1-answer to Q.*

- ***Step 1:*** *The first step is to identify all objects that satisfy* Q*. The two possibilities are objects* $tank1$ *and* $tank2$*.*

- ***Step 2:*** *The second step is to compute the occurrence probability of object* $tank1$ *and* $tank2$ *respectively. First we identify that the ancestors of* $tank1$ *consists of* $convoy1$ *and the root* $I1$*. On the other hand, the ancestors of* $tank2$ *also consists of* $convoy1$ *and the root* $I1$*. Then we compute the (minimum) conditional probability of occurrence of* $I1$*,* $convoy1$*,* $tank1$ *and* $tank2$*.* $\mathsf{cex}(I1) = 1$ *because* $I1$ *is the root. As* $I1$ *is the parent of* $convoy1$*, we can obtain* $\mathsf{cex}(convoy1)$ *by solving the following linear programming problem:*

  minimize $\Sigma_{\mathsf{c} \in \mathsf{PC}(I1) \wedge convoy1 \in \mathsf{c}} p(\mathsf{c})$

  subject to $\mathsf{OC}(I1)$*,*

  where $p(\mathsf{c}) \geq 0$ *is a real-valued variable denoting the probability that* c *is the actual set of children of* $I1$*. The object constraints* $\mathsf{OC}(I1)$ *consist of the following equations:*

  $0.2 \leq p(\{convoy1\}) \leq 0.4,$

  $0.1 \leq p(\{convoy2\}) \leq 0.4,$

  $0.4 \leq p(\{convoy1, convoy2\}) \leq 0.7$ *and*

  $p(\{convoy1\}) + p(\{convoy2\}) + p(\{convoy1, convoy2\}) = 1.$

  The result is: $\mathsf{cex}(convoy1) = 0.6$*. Similarly, as* $convoy1$ *is the parent of* $tank1$*, we can obtain* $\mathsf{cex}(tank1)$ *by solving the following linear programming problem:*

  minimize $\Sigma_{\mathsf{c} \in \mathsf{PC}(convoy1) \wedge tank1 \in \mathsf{c}} p(\mathsf{c})$

  subject to $\mathsf{OC}(convoy1)$*,*

  where $p(\mathsf{c}) \geq 0$ *is a real-valued variable denoting the probability that* c *is the actual set of children of* $convoy1$*. The object constraints* $\mathsf{OC}(convoy1)$ *consist of the following equations:*

  $0.2 \leq p(\{tank1\}) \leq 0.7,$

  $0.3 \leq p(\{tank2\}) \leq 0.8,$

  $p(\{tank1\}) + p(\{tank2\}) = 1.$

  The result is: $\mathsf{cex}(tank1) = 0.2$*. Finally, as* $convoy1$ *is the parent of* $tank2$*, we can obtain* $\mathsf{cex}(tank2)$ *by solving the following linear programming problem:*

minimize $\Sigma_{\mathsf{c} \in \mathsf{PC}(convoy1) \wedge tank2 \in \mathsf{c}} p(\mathsf{c})$

subject to $\mathsf{OC}(convoy1)$,

where $p(\mathsf{c}) \geq 0$ *is a real-valued variable denoting the probability that* c *is the actual set of children of* $convoy1$*. The result is:* $\mathsf{cex}(tank2) = 0.3$*. Therefore, the (minimum) computed occurrence probability of* $tank1$ *is* $\mathsf{cop}(tank1) = \mathsf{cex}(I1) \cdot \mathsf{cex}(convoy1) \cdot \mathsf{cex}(tank1) = 1 \cdot 0.6 \cdot 0.2 = 0.12$ *which exceeds the threshold. The (minimum) computed occurrence probability of* $tank2$ *is* $\mathsf{cop}(tank2) = \mathsf{cex}(I1) \cdot \mathsf{cex}(convoy1) \cdot \mathsf{cex}(tank2) = 1 \cdot 0.6 \cdot 0.3 = 0.18$ *which exceeds the threshold. As a result, the* 0.1*-answer to* Q *is* $\{tank1, tank2\}$*.*

3.6.2 Algorithm to solve SIMO queries

The preceding section provides a sound and complete algorithm to answer SISO queries. In this section, I extend the algorithm to solve SIMO queries. Unlike SISO queries, to answer a SIMO query, I need to find a set of objects that satisfy the query. As a consequence, an r-answer to an SIMO query is defined as a set of *sets of objects* (Definition 3.5.6, Definition 3.5.8). I extend the previous algorithm in Section 3.6.1 as follows.

The extended algorithm to solve this problem also involves two core steps:

- **Step 1:** The first step is to identify all *sets* T *of objects* $\{o_1, \ldots, o_n\}$ (and substitutions $\theta_T = \{p_1/o_1, \ldots, p_n/o_n\}$) in the weak instance graph $\mathcal{G}_{\mathcal{W}}$ of $\mathcal{I}$ that satisfy the SIMO query Q. As in the SISO case, this can be done using any available implementation of semistructured databases. Hence I do not present the details here.

- **Step 2:** The second step is to check (using the original probabilistic instance $\mathcal{I}$ rather than its weak instance graph) which of the sets of objects returned in the preceding step have an occurrence probability that exceeds the threshold w.r.t. all global interpretations that satisfy the original probabilistic instance.

To solve step 2, I can modify the procedure used to find the minimal occurrence probability of T (given a T that passes the test of step 1 above).

Suppose $\mathcal{I} = (V, \mathsf{lch}, \tau, \mathsf{val}, \mathsf{card}, \mathsf{ipf})$ and suppose $T = \{o_1, \ldots, o_n\}$ where $o_1, \ldots, o_n \in V$. Furthermore, define set A to contain objects in T and all their ancestors (in the weak instance graph $\mathcal{G}_{\mathcal{W}}$). The subgraph of $\mathcal{G}_{\mathcal{W}}$ consisting of objects in A and edges among them is denoted by $\mathcal{G}_{\mathcal{W}}^{A}$. For every non-leaf object o in $\mathcal{G}_{\mathcal{W}}^{A}$, I define a quantity $\mathsf{ccp}_{\mathsf{A}}(o)$ as follows. Intuitively, $\mathsf{ccp}_{\mathsf{A}}(o)$ finds the ***conditional probability*** of occurrence of the set A_o of all o's *children* (that are also in A, i.e., in $\mathcal{G}_{\mathcal{W}}^{A}$), given that o is known to occur. For those leaf objects o in $\mathcal{G}_{\mathcal{W}}^{A}$, $\mathsf{ccp}_{\mathsf{A}}(o)$ is set to 1 by default.

1. If o is a leaf in $\mathcal{G}_{\mathcal{W}}^{A}$, then $\mathsf{ccp}_{\mathsf{A}}(o) = 1$.

2. Otherwise, let A_o be the set of o's children in $\mathcal{G}_{\mathcal{W}}^{A}$. Recall the object constraints $\mathsf{OC}(o)$ defined in Definition 3.3.6. Set $\mathsf{ccp}_{\mathsf{A}}(o)$ to be the result of solving the linear programming problem:

   minimize $\Sigma_{\mathsf{c} \in \mathsf{PC}(o) \wedge A_o \subseteq \mathsf{c}} p(\mathsf{c})$

   subject to $\mathsf{OC}(o)$,

   where $p(\mathsf{c})$ is a real-valued variable denoting the probability that c is the actual set of children of o. As usual I assume that all the variables $p(\mathsf{c}) \geq 0$.

Intuitively, the quantity $\mathsf{ccp}_{\mathsf{A}}(o)$ specifies the smallest occurrence probability of object o's children A_o *given that* o is already known to occur in a semistructured instance.

Proposition 4 *For any tree structured probabilistic instance $\mathcal{I}$ and any object o, $\mathsf{ccp}_{\mathsf{A}}(o)$ can be computed in time polynomial in the size of $\mathcal{I}$.*

Then the ***computed occurrence probability,*** $\mathsf{cop}(T)$ ***of*** T ***in*** $\mathcal{I}$ is given by $\prod_{o \in A} \mathsf{ccp}_{\mathsf{A}}(o)$.

The following theorem says that the occurrence probability of T (which is defined declaratively in Definition 3.5.6) corresponds exactly to the computed occurrence probability of T according to the above procedure.

Theorem 3.7 (correctness theorem) *Suppose $\mathcal{I} = (V, \mathsf{lch}, \tau, \mathsf{val}, \mathsf{card}, \mathsf{ipf})$ is a tree structured probabilistic instance, set T of objects $\{o_1, \ldots, o_n\}$, and $o_i \in V$. Then,*

$$\mathsf{cop}(T) = \mathbf{INF}\{\Sigma_{\mathcal{S} \in Domain(\mathcal{I}) \wedge T \subseteq \mathcal{S}} \mathcal{P}(S) \mid \mathcal{P} \models \mathcal{I}\}$$

.

Proof

Suppose d is the depth of $\mathcal{G}_{\mathcal{W}}^{A}$ (the maximum length of the shortest paths from leaf nodes to the root).

If $d = 0$, then T only contains o_r that is the root of $\mathcal{I}$. Hence, $\mathsf{cop}(T) = 1$ by definition. As every compatible semistructured instance contains the root, it follows by the definition of probabilistic interpretation that every probabilistic interpretation assigns 1 to $\Sigma_{\mathcal{S} \in Domain(\mathcal{I}) \wedge o_r \in \mathcal{S}} \mathcal{P}(S)$. This completes the base case.

If $d = 1$, then suppose o_r is the root (which always exists in $\mathcal{G}_{\mathcal{W}}^{A}$), and A_{o_r} is the root's children that are also in $\mathcal{G}_{\mathcal{W}}^{A}$. Since for every leaf object o, $\mathsf{ccp_A}(o) = 1$, so $\mathsf{cop}(T) = \mathsf{ccp_A}(o_r) = 1$. $\mathsf{ccp_A}(o_r)$ is obtained from the result of solving the linear programming problem:

minimize $\Sigma_{\mathsf{c} \in \mathsf{PC}(o_r) \wedge A_{o_r} \subseteq \mathsf{c}} p(\mathsf{c})$

subject to $\mathsf{OC}(o_r)$,

where $p(\mathsf{c})$ is a real-valued variable denoting the probability that c is the actual set of children of o_r. As usual I assume that all the variables $p(\mathsf{c}) \geq 0$. By Proposition 2 and Theorem 3.4, for every global interpretation that satisfies the probabilistic instance, it can be converted to an equivalent local interpretation. Recall Definition 2.3.6, $\frac{\Sigma_{\mathcal{S} \in Domain(\mathcal{I}) \wedge o_r \in \mathcal{S} \wedge A_{o_r} \in \mathcal{S}} \mathcal{P}(S)}{\Sigma_{\mathcal{S} \in Domain(\mathcal{I}) \wedge o_r \in \mathcal{S}} \mathcal{P}(S)} = \frac{\Sigma_{\mathcal{S} \in Domain(\mathcal{I}) \wedge T \subseteq \mathcal{S}} \mathcal{P}(S)}{1}$ actually returns the conditional probability that A_{o_r} is a subset of children of o_r, i.e., $\Sigma_{\mathsf{c} \in \mathsf{PC}(o_r) \wedge A_{o_r} \subseteq \mathsf{c}} p(\mathsf{c})$. Thus, the local interpretation corresponding to the solution of the above linear programming problem also corresponds to the global interpretation that gives the value of $\mathbf{INF}\{\Sigma_{\mathcal{S} \in Domain(\mathcal{I}) \wedge T \subseteq \mathcal{S}} \mathcal{P}(S) \mid \mathcal{P} \models \mathcal{I}\}$. This completes the case of $d = 1$.

Suppose $d = m + 1$ for some integer m. Suppose $\mathcal{G}_{\mathcal{W}}^{A}$ has the root o_r having the set A_{o_r} of children. Suppose $A_{o_r} = \{o_1, \ldots, o_c\}$, B_i is the subtree of $\mathcal{G}_{\mathcal{W}}^{A}$ rooted at $o_i \in A_{o_r}$ with depth $\leq m$, T_i is the subset of T that appears in B_j. Let $\mathcal{I}_i$ be the probabilistic instance which is just like the subtree (of $\mathcal{I}$) rooted at o_i. By the induction hypothesis, I know that $\mathsf{cop}_{\mathcal{I}_i}(T_i) = \mathbf{INF}\{\Sigma_{\mathcal{S}_i \in Domain(\mathcal{I}_i) \wedge B_i \subseteq \mathcal{S}_i \wedge \mathcal{P}_i \models \mathcal{I}_i} \mathcal{P}_i(S_i)\}$ where $\mathcal{P}_i$ is defined as follows. If $\mathcal{S}_i$ is compatible with $\mathcal{I}_i$ then $\mathcal{P}_i(\mathcal{S}_i)$ is the sum of $\mathcal{P}(\mathcal{S})$ for all $\mathcal{S}$ compatible with $\mathcal{I}$ such that $\mathcal{S}_i$ is the subtree of $\mathcal{S}$

rooted at o_i.

Note that $\mathsf{cop}_{\mathcal{I}}(T) = \mathsf{ccp}_{\mathsf{A}}(o_r) \times \prod \mathsf{cop}_{\mathcal{I}_i}(T_i)$ by definition of cop. However, since $\mathcal{P}$ satisfies $\mathcal{W}$ corresponding to $\mathcal{I}$, so by Definition 3.3.4, every object is conditionally independent of any possible set of its non-descendants given that its parent exists. As $\mathsf{cop}_{\mathcal{I}_i}(T_i)$ represents the minimal conditional probability of the existence of B_i given that o_i exists, and every subtree is independent of each other given its parent exists, so $\prod \mathsf{cop}_{\mathcal{I}_i}(T_i)$ returns the minimal probability that all those subtrees (B_i) exist given that all o_i's exist. $\mathsf{ccp}_{\mathsf{A}}(o_r)$ returns the minimal probability that all o_i's exist given that the root o_r exists. As the root always exists, so $\mathsf{cop}_{\mathcal{I}}(T) = \mathsf{ccp}_{\mathsf{A}}(o_r) \times \prod \mathsf{cop}_{\mathcal{I}_i}(T_i)$ returns the minimal probability that all those subtrees (B_i) exist, i.e., $T' \in \mathcal{S}$, which is equal to $\mathbf{INF}\{\Sigma_{\mathcal{S} \in Domain(\mathcal{I}) \wedge T' \in \mathcal{S}} \mathcal{P}(S) \mid \mathcal{P} \models \mathcal{I}\}$. This completes the proof. ∎

It is important to note that the above condition is very similar to the condition on the right side of the inequality of Definition 3.5.6 (it is exactly the same if we only consider sets T of objects that satisfy the query). ***An immediate consequence of the above theorem is that the two-step procedure outlined in this section is correct, as step 2 is only applied to sets of objects that satisfy the SIMO query condition Q.***

Example 3.6.2 *Consider the probabilistic instance in Figure 3.5 and a SIMO query Q which is $(\mathsf{val}(I2.convoy.tank) = T80 \wedge \mathsf{val}(I2.convoy.truck) = mac)$.*

Suppose we want to find a 0.1-answer to Q.

- ***Step 1:*** *The first step is to identify all sets of objects (and substitutions) that satisfy Q. The result is $\{\{tank3,\ truck4\}\}$ with substitution $\theta_T = \{I2.convoy.tank/tank3,\ I2.convoy.truck /truck4\}$.*

- ***Step 2:*** *The second step is to compute the occurrence probability of the set returned in Step 1. I define set $T = \{tank3, truck4\}$. Then I define set $A = \{I2, convoy3,\ convoy4, tank3,\ truck4\}$ to contain T and its ancestors. Then I compute the (minimum) conditional probability of occurrence ($\mathsf{ccp}_{\mathsf{A}}$) of A_{I2}, $A_{convoy3}$ and $A_{convoy4}$ where A_o is the set of children of o that are also in A. By default, I define $\mathsf{ccp}_{\mathsf{A}}(tank3) = \mathsf{ccp}_{\mathsf{A}}(truck4) = 1$ because $tank3$ and $truck4$ are leaves in $\mathcal{G}^A_{\mathcal{W}}$. Let us first compute $\mathsf{ccp}_{\mathsf{A}}(I2)$, where $A_{I2} = \{convoy3, convoy4\}$.*

o	l	$\mathsf{lch}(o, l)$
I2	convoy	{ convoy3, convoy4 }
convoy3	tank	{ tank3 }
convoy3	truck	{ truck2 }
convoy4	truck	{ truck3, truck4 }

$\mathsf{c} \in \mathsf{PC}(I2)$	$\mathsf{ipf}(I2, \mathsf{c})$
{ convoy3}	[0.1, 0.1]
{ convoy4}	[0.1, 0.2]
{ convoy3, convoy4}	[0.7, 0.8]

o	$\tau(o)$	$\mathsf{val}(o)$
tank3	tank-type	T-80
truck2	truck-type	rover
truck3	truck-type	rover
truck4	truck-type	mac

$\mathsf{c} \in \mathsf{PC}(convoy3)$	$\mathsf{ipf}(convoy3, \mathsf{c})$
{ }	[0, 0]
{ tank3}	[0.3, 0.8]
{ truck2}	[0.2, 0.7]
{ tank3, truck2}	[0, 0]

o	l	$\mathsf{card}(o, l)$
I2	convoy	[1,2]
convoy3	tank	[0,1]
convoy3	truck	[0,1]
convoy4	truck	[1,1]

$\mathsf{c} \in \mathsf{PC}(convoy4)$	$\mathsf{ipf}(convoy4, \mathsf{c})$
{ truck3}	[0.1, 0.2]
{ truck4}	[0.8, 0.9]

Figure 3.5: Another probabilistic instance for the surveillance domain.

We can obtain $\mathsf{ccp}_{\mathsf{A}}(I2)$ *by solving the following linear programming problem:*

minimize $\Sigma_{\mathsf{c}\in\mathsf{PC}(I2)\,\wedge\,\{convoy3,convoy4\}\in\mathsf{c}}\, p(\mathsf{c})$

subject to $\mathsf{OC}(I2)$,

where $p(\mathsf{c}) \geq 0$ *is a real-valued variable denoting the probability that* c *is the actual set of children of* $I2$*. The object constraints* $\mathsf{OC}(I2)$ *consist of the following equations:*

$0.1 \leq p(\{convoy3\}) \leq 0.1,$

$0.1 \leq p(\{convoy4\}) \leq 0.2,$

$0.7 \leq p(\{convoy3, convoy4\}) \leq 0.8$ *and*

$p(\{convoy3\}) + p(\{convoy4\}) + p(\{convoy3, convoy4\}) = 1.$

The result is: $\mathsf{ccp}_{\mathsf{A}}(I2) = 0.7$. *Next, let us compute* $\mathsf{ccp}_{\mathsf{A}}(convoy3)$, *where* $A_{convoy3} = \{tank3\}$. *We can obtain* $\mathsf{ccp}_{\mathsf{A}}(convoy3)$ *by solving the following linear programming problem:*

$\textbf{\textit{minimize}}\ \Sigma_{\mathsf{c} \in \mathsf{PC}(convoy3) \wedge \{tank3\} \in \mathsf{c}}\, p(\mathsf{c})$

$\textbf{\textit{subject to}}\ \mathsf{OC}(convoy3)$,

where $p(\mathsf{c}) \geq 0$ *is a real-valued variable denoting the probability that* c *is the actual set of children of* $convoy3$. *The object constraints* $\mathsf{OC}(convoy3)$ *consist of the following equations:*

$0.3 \leq p(\{tank3\}) \leq 0.2,$

$0.8 \leq p(\{truck2\}) \leq 0.7,$

$p(\{tank3\}) + p(\{truck2\}) = 1.$

The result is: $\mathsf{ccp}_{\mathsf{A}}(convoy3) = 0.3$. *Finally, let us compute* $\mathsf{ccp}_{\mathsf{A}}(convoy4)$, *where* $A_{convoy4} = \{truck4\}$. *We can obtain* $\mathsf{ccp}_{\mathsf{A}}(convoy4)$ *by solving the following linear programming problem:*

$\textbf{\textit{minimize}}\ \Sigma_{\mathsf{c} \in \mathsf{PC}(convoy4) \wedge \{truck4\} \in \mathsf{c}}\, p(\mathsf{c})$

$\textbf{\textit{subject to}}\ \mathsf{OC}(convoy4)$,

where $p(\mathsf{c}) \geq 0$ *is a real-valued variable denoting the probability that* c *is the actual set of children of* $convoy4$. *The object constraints* $\mathsf{OC}(convoy4)$ *consist of the following equations:*

$0.1 \leq p(\{truck3\}) \leq 0.2,$

$0.8 \leq p(\{truck4\}) \leq 0.9,$

$p(\{truck3\}) + p(\{truck4\}) = 1.$

The result is: $\mathsf{ccp}_{\mathsf{A}}(convoy2) = 0.8$. *Therefore, the (minimum) computed occurrence probability of* $\{tank3, truck4\}$ *is* $\mathsf{cop}(\{tank3, truck4\}) = \mathsf{ccp}_{\mathsf{A}}(I2) \cdot \mathsf{ccp}_{\mathsf{A}}(convoy3) \cdot \mathsf{ccp}_{\mathsf{A}}(convoy4) \cdot \mathsf{ccp}_{\mathsf{A}}(tank3) \cdot \mathsf{ccp}_{\mathsf{A}}(truck4) = 0.7 \cdot 0.3 \cdot 0.8 \cdot 1 \cdot 1 = 0.168$ *which exceeds the threshold. As a result, the* 0.1-*answer to* Q *is* $\{\{tank3, truck4\}\}$ *with substitution* $\theta_T = \{I2.convoy.tank/tank3,$ $I2.convoy.truck/truck4\}$.

3.6.3 Algorithm to solve IMIMO queries

Recall (cf. Definition 3.4.6) that an IMIMO query Q is a boolean combination of groups (or *query groups*). Each query group is a boolean combination of atomic queries involving one instance and each instance is involved in only one group. Thus, each query group is equivalently an SISO or SIMO query (depending on whether one object or more are needed to satisfy it) and can be solved by the algorithms described in previous sections.

After solving the query groups, we have computed the minimal probability of each possible answer (an object for SISO query group or a set of objects with its substitution for SIMO query group) in each query group.

If the probability that a given object or set of objects is an answer to a particular group is independent from the probability that some other object or set of objects is an answer to another group, then the minimal probability that a given set of objects obtained by combining each answer from each query group satisfies the whole original IMIMO query Q is simply the product of the minimal probability of each component answer from each group (for the conjunctive case).

If the boolean combination of the query groups also involves disjunction, then it is enough to have only one operand of the disjunction operator to be true in order to make the disjunction expression true. Thus, instead of returning a combination of the answers (object sets) from each operand (query group) in the conjunction case, we have the following three outputs in the disjunction case: (1) the answer of the left operand with its original minimal probability, (2) the answer of the right operand with its original minimal probability, and (3) the combination of the answers of the left and right operands with its new minimal probability which may depend on the relationship (independent? mutually exclusive? correlated somehow, etc.) between the disjuncts. For the answer (3), it actually means that we have the answer from the left AND the answer from the right *together*, so in fact we want to have the conjunction of the two's minimal probabilities.

Thus, for each set of answers from all query groups, we check the combination to see whether the query is satisfied. As a result, we generate one or more "candidate" answers, each of which will be kept as the final answer if its resulting minimal probability exceeds the specified threshold.

The whole algorithm to solve this problem involves the following three core steps (Figure 3.6 shows its pseudo code).

- **Step 1:** The first step is to rewrite the IMIMO query as a boolean combination of query groups where each query group involves one instance and each instance is involved in one query group only.

- **Step 2:** The second step is to use algorithms for SISO and SIMO queries to solve each query group (returned in the preceding step) and obtain a set of answers to each query group (an answer is an object or a set of objects with its substitution) with the corresponding minimal probabilities.

- **Step 3:** The final step is to find the "candidate" answers (constructed from the answer returned in preceding step) that satisfy the query and compute their minimal probabilities using the conjunctive strategy specified by the user (or a default strategy provided by the system). As we want to find the minimal occurrence probability of the object set, we can transform each point probability $prob_j$ (of each object subset existing in each $\mathcal{I}_j$) to an interval probability $[prob_j, prob_j]$ and take the lower bound of a specific conjunctive strategy as the value of the conjunction.

  For each query group, we select none or any one of the answer returned in step 2. As a result, we obtain a set of objects from one or more probabilistic instances (with the corresponding substitution). We then check to see whether the query returns true by this substitution. If so, this is a candidate answer and we compute the conjunction of the minimal probabilities of those answers from their query groups. Finally, only the candidates whose results exceeding the threshold will be returned.

Theorem 3.8 (correctness theorem) *Suppose $\mathcal{I}_1, \ldots \mathcal{I}_m$ are tree structured probabilistic instances, the object set $TC = \{o_1, \ldots, o_n\} \subseteq \mathcal{I}_1 \cup \ldots \cup \mathcal{I}_m$. Then,*

$$prob_{TC} = \mathbf{INF}\{Conj_i(\Sigma_{\mathcal{S}_i \in Domain(\mathcal{I}_i) \wedge (T \cap \mathcal{I}_i) \in \mathcal{S}_I} \mathcal{P}_i(\mathcal{S})) \mid \mathcal{P}_i \models \mathcal{I}_i\}.$$

```
• Input: IMIMO query $Q$, threshold $r$
• Output: $r$-answer
    1. Rewrite $Q$ into the form $G_1 \ \gamma \ G_2 \ \gamma \ \ldots \ \gamma \ G_m$ where $G_i$ is a query group in the form
       $(q_{i,1} \ \gamma \ q_{i,2} \ \gamma \ \ldots \ \gamma \ q_{i,n_i})$, $\gamma$ is either $\wedge$ or $\vee$, and $q_{i,j}$ are atomic queries involving only in-
       stance $\mathcal{I}_i$.
    2. For each $G_i$, we execute $SIMO(G_i)$ where $SIMO()$ is the algorithm to solve SIMO queries
       and returns a set $T_i$ of answers (object sets $T_{i,k}$) with substitutions $\theta_{i,k}$ and the minimal
       probabilities $prob_{i,k}$.
    3. For each combination $TC$ (any $T_{i,k}$ or none from each $T_i$) of object sets,
        – if $Q$ returns true by $\theta_{TC}$ (the corresponding substitutions), then compute the minimal
          probability $prob_{TC}$ of $TC$ by using the conjunctive strategies specified. If $prob_{TC} \geq r$,
          then $TC$ by $\theta_{TC}$ is included in the $r$-answer.
```

Figure 3.6: Pseudo code to solve IMIMO queries

Proof

Each query group G_i involves only probabilistic instance $\mathcal{I}_i$. By Theorem 3.7, the probability $prob_{i,k}$ returned by $SIMO(G_i)$ equals $\mathbf{INF}\{\Sigma_{\mathcal{S}_i \in Domain(\mathcal{I}_i) \wedge (T_{i,k} \cap \mathcal{I}_i) \in \mathcal{S}_i} \mathcal{P}_i(\mathcal{S}) \mid \mathcal{P}_i \models \mathcal{I}_i\}$. My algorithm then computes conjunction of **INF**. I show below that "the conjunction of **INF**" and "**INF** of conjunction" are equivalent.

Here the conjunction of **INF** means that for each of sets $S_1, \ldots, S_m$, I apply **INF** on its elements and then I transform each resulting point value p_i into an interval $[p_i, p_i]$. A conjunctive strategy is applied and the lower bound of the result is taken as the final result. In the other words, I am computing $Conj(\mathbf{INF}\{S_1\}, \ldots, \mathbf{INF}\{S_m\})$.

On the other hand, the **INF** of conjunction means that for each hitting set H of the sets $S_1, \ldots, S_m$, I transform each element point value p_i into an interval $[p_i, p_i]$. A conjunctive strategy is applied and the lower bound of the result is taken as the result. Among all these resulting values, I apply **INF** and get the final result. In the other words, I am computing

$\mathbf{INF}\{Conj(H) \mid H$ is a hitting set of $S_1, \ldots, S_m\}$.

First, note that $\{\mathbf{INF}\{S_1\}, \ldots, \mathbf{INF}\{S_m\}\}$ is also a hitting set of $S_1, \ldots, S_m$. Second, from Figure 3.3, I know that all conjunctive strategies are non-decreasing functions w.r.t. its inputs. Therefore, if the input is **INF**, then the result will be also **INF**. As a result, the value of $Conj(\mathbf{INF}\{S_1\}, \ldots, \mathbf{INF}\{S_m\})$ is also minimum among all $Conj(H)$ and will be returned as the result of $\mathbf{INF}\{Conj(H) \mid H$ is a hitting set of $S_1, \ldots, S_m\}$.

Because both the conjunction of **INF** and **INF** of conjunction actually considers all possible cases and return the minimal value, their results are equal and this completes the proof. ∎

Recall that the algorithms for SISO and SIMO queries compute the minimal probability of an answer to a query group, i.e., the lower bound of the probability interval of that answers w.r.t all possible global interpretations satisfying the instance. For example, suppose two query groups G_1, G_2 are combined by conjunction and the conjunctive strategy used is the one of independence. If the minimal probability of the answer $\{o_1, o_2\}$ to G_1 is x_1 and the minimal probability of the answer $\{o_3, o_4, o_5\}$ to G_2 is x_2, then the minimal probability of the answer $\{o_1, o_2, o_3, o_4, o_5\}$ to $G_1 \wedge G_2$ is $x_1 \cdot x_2$. If the two query groups G_1, G_2 are combined by disjunction and the disjunctive strategy used is the one of negative correlation, then besides the original answers $\{o_1, o_2\}$ and $\{o_3, o_4, o_5\}$ (with the minimal probability of x_1 and x_2 respectively), another possible answer is $\{o_1, o_2, o_3, o_4, o_5\}$ with the minimal probability of $min(1, x_1 + x_2)$.

If a user specifies some conjunctive strategies that requires the value of upper bound of a probability interval, then it is straightforward to modify the algorithms for SISO and SIMO to find the *maximal* occurrence probability of a given answer and obtain the minimal probability of the combination of answers to the original IMIMO query in a similar manner.

Example 3.6.3 *Consider the two probabilistic instances in Figure 3.1 and Figure 3.5, and an IMIMO query Q which is* $(\mathsf{val}(w = I1.convoy.tank) = T80 \vee ((\mathsf{val}(w) = T72 \vee \mathsf{val}(I2.convoy.tank) = T80) \wedge (\mathsf{val}(w) = T72 \vee \mathsf{val}(I2.convoy.truck) = mac)))$*. It is IMIMO because it can be rewritten as follows:* $((\mathsf{val}(w = I1.convoy.tank) = T80 \vee \mathsf{val}(w) = T72) \vee (\mathsf{val}(I2.convoy.tank) = T80 \wedge \mathsf{val}(I2.convoy.truck) = mac))$*, which consists of two query groups* $G_1 = (\mathsf{val}(w = I1.convoy.tank)$

$= T80 \vee \mathsf{val}(w) = T72)$ *and* $G_2 = (\mathsf{val}(I2.convoy.tank) = T80 \wedge \mathsf{val}(I2.convoy.truck) = mac)$ *connected by a disjunction. This query indicates that we want only one object (tank) from instance* $I1$ *whose value is either* $T80$ *or* $T72$ *or a set of two objects (a tank and a truck) from instance* $I2$ *whose values are* $T80$ *and mac. Suppose we want to find a* 0.17*-answer to* Q *and the disjunctive strategy used is the one of positive-correlation.* G_1 *is the same as the compound SISO query in Example 3.6.1 and* G_2 *is the same as the SIMO query in Example 3.6.2. From Example 3.6.1, we know that the objects* $tank1$ *and* $tank2$ *in* $I1$ *satisfy* G_1 *and have computed occurrence probabilities of 0.12 and 0.18 respectively. From Example 3.6.2, we know that the object set* $\{tank3, truck4\}$ *in* $I2$ *satisfy* G_2 *and has a computed occurrence probability of 0.168. Since* G_1 *and* G_2 *are connected by a disjunction, so the* 0.17*-answer includes the object* $tank2$ *in* $I1$ *as its (minimum) probability exceeds the threshold* 0.17*. Furthermore, the candidates of the answer also include the following:* $\{tank1, tank3, truck4\}$ *and* $\{tank2, tank3, truck4\}$*. The (minimum) probability of the former is* $max(0.12, 0.168) = 0.168$ *while that of the latter is* $max(0.18, 0.168) = 0.18$*. As a result, the* 0.17*-answer consists of the object* $tank2$ *in* $I1$ *and the object set* $\{tank2, tank3, truck4\}$ *in* $I1, I2$*.*

3.6.4 Algorithm to solve DMIMO queries

In the previous section, I have described how to solve IMIMO queries. An atomic IMIMO query Q involves only one instance and furthermore, it can be expressed as a boolean combination of groups (or *query groups*) such that each query group is a boolean combination of atomic queries involving one instance and each instance is involved in only one group. DMIMO queries do not satisfy these conditions.

For IMIMO queries, we can consider each instance individually to see which object (or set of objects) can satisfy a query group or not and then consider the different combinations of the results of the query groups.

It is obvious that the first condition is necessary to handle one instance each time. In order to understand why the second condition is necessary, let us consider the following illustrative example. Suppose we are given an IMIMO query $Q_1 = (q_{1,1} \wedge q_{1,2}) \vee q_{2,1}$ where $q_{i,j}$ is an atomic query involving the ith instance. It is obvious that Q_1 consists of two query groups, $G_1 = (q_{1,1} \wedge q_{1,2})$ and $G_2 = q_{2,1}$, connected by the operator $\vee$. Hence, we can consider which sets of objects from the first instance satisfy G_1 (this is the first set of answers to Q_1) and which sets of objects in the second instance satisfy G_2 (the second set of answers to Q_1), and then can use a disjunctive strategy to find the minimal probability of the combination of the above two sets of answers. Now, consider a DMIMO query $Q_2 = (q_{1,1} \wedge q_{2,1}) \vee q_{1,2}$. The answers T_1 satisfying $q_{1,2}$ alone can satisfy Q_2, but answers T_2 satisfying $q_{1,1}$ alone cannot satisfy Q_2 without combining with the answers T_3 which satisfy $q_{2,1}$. This means that we need to keep the minimal probabilities of all three cases of object sets in the first instances: T_1, T_2 and $T_1 \cup T_2$. One approach is to keep all necessary information for each instance's object set candidates (e.g., which atomic queries they satisfy) and then use them as a basis to select for the combination with other candidates of other instances.

Nevertheless, the previous approach cannot work for a DMIMO query with some atomic queries involving two or more instances, e.g. $val(\mathcal{I}_1 : p_1) > val(\mathcal{I}_2 : p_2)$.

To address this problem, we must consider multiple instances concurrently. We must examine all the path expressions in the atomic queries and select all candidate objects in each instance satisfying those path expressions. Thus, for each path expression, we have a set of object candidates. We must then try all combinations of object candidates (as well as either or both side of a disjunction) to check whether the whole query is satisfied or not. If it is satisfied, then we can compute the minimal probability of the selected object candidates of each instance, and get the resulting minimal probability by a probabilistic conjunctive strategy (it is a conjunction since we are considering all possible combinations of candidates, and in each combination, we require *all* those selected candidates to exist).

Figure 3.7 shows the pseudo code.

Theorem 3.9 (correctness theorem) *Suppose $\mathcal{I}_1, \ldots \mathcal{I}_m$ are tree structured probabilistic instances, the object set $TC = \{o_1, \ldots, o_n\} \subseteq \mathcal{I}_1 \cup \ldots \cup \mathcal{I}_m$. Then,*

$$prob_{TC} \geq \mathbf{INF}\{Conj_i(\Sigma_{\mathcal{S}_i \in Domain(\mathcal{I}_i) \wedge (T \cap \mathcal{I}_i) \in \mathcal{S}_I} \mathcal{P}_i(\mathcal{S})) \mid \mathcal{P}_i \models \mathcal{I}_i\}.$$

Proof

```
• Input: DMIMO query $Q$, threshold $r$
• Output: $r$-answer
    1. For each path expression $p_i$ in $Q$, identify the set $T_i$ of objects located by $p_i$.
    2. For each combination $TC$ (any one or none from each $T_i$) of objects (and the corresponding
       substitution $\theta_{TC}$),
         – if $Q$ returns true by this substitution, then for each probabilistic instance $I_j$,
             ∗ compute the minimal occurrence probability $prob_j$ of the set of objects from $TC$ that
               exist in $I_j$ in the way similar to step 2 of the algorithm to solve SIMO queries. If no
               such object exists, then $prob_j = 1$.
         Then, compute the minimal occurrence probability $prob_{TC}$ of $TC$ using a specified con-
         junctive strategy (Figure 3.4). If $prob_{TC} \geq r$, then $TC$ by $\theta_{TC}$ is included in the $r$-answer.
```

Figure 3.7: Pseudo code to solve DMIMO queries

The internal iteration of step 2 in the algorithm is identical to step 3 of IMIMO algorithm (Figure 3.6). Thus, it can be proved in a manner similar to the proof of Theorem 3.8. ∎

Example 3.6.4 *Consider the two probabilistic instances in Figure 3.1 and Figure 3.5, and a DMIMO query Q which is* $((\mathsf{val}(I1.convoy.tank) = \mathsf{val}(I2.convoy.tank)) \vee (\mathsf{val}(I1.convoy.truck) \neq \mathsf{val}(I2.convoy.truck)))$. *It is DMIMO because every atomic query involves two instances:* $I1$ *and* $I2$. *In order to solve this query, we will first identify the set of objects satisfying each path expression. The objects in the set* $O_1 = \{tank1, tank2\}$ *satisfy the first path expression* $P_1 = I1.convoy.tank$. *The object in the set* $O_2 = \{tank3\}$ *satisfies the second path expression* $P_2 = I2.convoy.tank$. *The object in the set* $O_3 = \{truck1\}$ *satisfies the third path expression* $P_3 = I1.convoy.truck$. *The objects in the set* $O_4 = \{truck2, truck3, truck4\}$ *satisfy the fourth path expression* $P_4 = I2.convoy.truck$. *For the first atomic query* $(\mathsf{val}(I1.convoy.tank) = \mathsf{val}(I2.convoy.tank))$, *we consider the following two object sets:* $\{tank1, tank3\}$ *and* $\{tank2, tank3\}$. *We can see that only the first object set satisfies the first atomic query. For the second atomic query* $(\mathsf{val}(I1.convoy.truck) \neq \mathsf{val}(I2.convoy.truck))$, *we consider the following three object sets:*

$\{truck1, truck2\}$*,* $\{truck1, truck3\}$ *and* $\{truck1, truck4\}$ *We can see that only the third object set satisfies the second atomic query. Since the two atomic queries are connected by a disjunction, so the object sets* $\{tank1, tank3\}$ *in* $I1$ *(satisfying the first atomic query) and* $\{truck1, truck4\}$ *in* $I2$ *(satisfying the second atomic query) as well as the combination of them, i.e.,* $\{tank1, tank3, truck1, truck4\}$ *in* $I1, I2$ *can satisfy the whole query respectively. The minimal probabilities of the first two candidates can be obtained by SIMO algorithm and that of the last candidate can be obtained by using a specific disjunctive strategy. Finally, only those candidates with probabilities exceeding the threshold* r *will be returned as the* r*-answer to* Q*.*

3.7 Summary

In this chapter, I described the interval-probability version of PXML model, which is called PIXML model. I have developed its formal theory with proofs. I also proposed a query language to query such single or multiple instances. I then provided an operational semantics that is proven to be sound and complete.

We have seen how we can incorporate and manipulate uncertainty information in XML databases. In the next chapter, I will examine the next challenge: how to manipulate ontologies in XML databases.

Chapter 4

Maintaining RDF Databases

In the previous two chapters, we have seen how we can incorporate and manipulate uncertainty information in XML databases. In this chapter, I first introduce to the reader the basics of RDF and RDQL in Section 4.1. RDQL is a commercial RDF language proposed by Hewlett Packard.

A large part of RDF focuses on storing *resource, property, value triples.* For example, if we wish to say that John is Ed's boss, we can describe this via the *resource, property, value* triple (Ed, boss, John) which says that the resource (Ed) has a property called "boss" with value John. Usually in RDF, rather than explicitly say Ed or John, we would use a URL-like syntax called a URI to describe a location that talks about Ed or John, respectively. It is clear that such triples can be stored in a relational database in many different ways (e.g. store them all in one relation called *triples* with schema (Resource, Property, Value) or we can store one relation for each attribute such as "boss" with schema (Resource, Value)). Once such a translation of RDF data is made into the relational representation, then any number of standard relational view maintenance algorithms can be used for maintaining RDF views. This chapter shows that this is a *bad idea.* RDF-instances are usually represented via labeled graphs. In Section 4.2, I present the IMA algorithm to maintain views when insertions are made to an RDF-instance. Likewise, I present the DMA algorithm to maintain views when deletions are made. I also present algorithms TMA and RMA to maintain views when different kinds of modifications are made to an RDF-instance.

In Section 4.3, I describe how to extend RDQL to support aggregations. I also propose the CAA algorithm to compute aggregates. Section 4.4 proposes the AMI, AMD, AMT and AMR algorithms to maintain aggregate views. Section 4.5 shows how all these views can also be maintained by converting an RDF database to a relational database and then using a standard relational view maintenance engine.

Section 4.6 describes my prototype implementation of these algorithms. I conducted exhaustive experiments measuring how my graph-based view maintenance algorithms compare with two of the best known view maintenance algorithms[39] on the problem of maintaining RDF-instances. The results show that, when the database is updated, my incremental maintenance algorithms work much faster than a complete recomputation and are significantly better than the use of standard view maintenance algorithms on relational representations of RDF databases. I am certainly not implying that classical view maintenance algorithms are bad. They apply to arbitrary relations, whereas my algorithms only apply to the types of relations generated by translating RDF-instances into a relational form. Classical view maintenance algorithms are clearly effective and should be used for non-RDF approaches.

I describe how to extend this problem to RDF containers, collections and reification in Section 4.7.

4.1 Overview of RDF and RDQL

4.1.1 RDF Model

RDF's main goal is to express information about the *values* of *properties* of *resources*. As a consequence, RDF statements express what we call *resource, property, value* triples. The resource, property and value are also often referred to as *subject, predicate* and *object* respectively. Each resource is expressed via a *Uniform Resource Indicator* (URI) which looks very similar to a URL. *Note that the value of a property of a given resource can be another resource.*

RDF also has the concept of an RDF *schema* which is used to express class-subclass as well as property-subproperty relationships. Figure 4.1(a) shows an example RDF-schema. This figure shows that "Painter" (the class type of http://www.culture.net#picasso132) is a subclass of "Artist." It is important to note that in RDF schema, properties are defined *independently* of classes, rather than inside classes as is common in object-oriented languages. Figure 4.1(a) states that `fname` is a property that applies to the "Artist" class. Thus, the "Painter" class (which is a subclass of "Artist") inherits this property.

Figure 4.1(b) shows a sample RDF instance. It states that there is a resource at http://www.culture.net#picasso132, for example, which has a property called `fname` whose value is "Pablo".

RDF-schemas can be viewed as labeled graphs whose vertices are either classes or data types. There is an edge from vertex v to vertex v' labeled with property p if the domain of p is v and the range of p is v'.

RDF-instances can also be viewed as graphs whose vertices are either resources or values. There is an edge from vertex v to vertex v' labeled with property p if the value of the property p of vertex v is v'. Figure 4.2's top half and bottom half show example graphs for an RDF-schema and an RDF-instance, parts of which are shown in Figure 4.1(a) and (b). The example schema graph shows one-to-one connections between nodes, but this is not necessary in general because the domain and/or range of a property can be defined to be one or more classes (or datatypes, in the case of the range). To be more precise, the "property" in an RDF schema graph should be a node rather than an edge. The property node is connected to its domain and range nodes by the "domain" edges and "range" nodes. Figure 4.1(a) is a simplified version.

4.1.2 RDQL, Views and Graph Patterns

There are currently just a few query languages for RDF databases such as RDQL[72], RQL[47], SeRQL[62] and RAL[28] (RAL is an algebra rather than a declarative query language). My view maintenance algorithms assume that views are defined in Hewlett Packard's RDQL language - my choice was based on the fact that RDQL seems to have the most industry support though the eventual winner in the "best RDF query language" sweepstakes is far from determined.

RDQL queries follow the usual SELECT-FROM-WHERE structure to locate RDF statements satisfying particular triple patterns. An (optional) AND clause can be used to specify conjunctive conditions, while an optional USING clause may be used to specify that only URIs having a certain prefix should be considered.

To find all (sculpture, museum) pairs where the sculpture was created by Rodin, the museum houses the given sculpture, and the museum web site was not modified since Jan 1, 2001, we can

(a)

```
<?xml version="1.0"?>

<!DOCTYPE rdf:RDF [<!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">]>

<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"

  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"

  xml:base="http://www.icom.com/schema1#">

<rdfs:Class rdf:ID="Artist"/>

<rdfs:Class rdf:ID="Painter"><rdfs:subClassOf rdf:resource="#Artist"/></rdfs:Class>

<rdfs:Datatype rdf:about="&xsd;string"/>

<rdf:Property rdf:ID="fname">

  <rdfs:domain rdf:resource="#Artist"/><rdfs:range rdf:resource="&xsd;string"/>

</rdf:Property>

</rdf:RDF>
```

(b)

```
<?xml version="1.0"?>

<!DOCTYPE rdf:RDF  [<!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">]>

<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"

xmlns:ns1="http://www.icom.com/schema1#">

  <rdf:Description rdf:about="http://www.culture.net#picasso132">

    <rdf:type rdf:resource="ns1:Painter"/>

    <ns1:fname rdf:datatype="&xsd;string">Pablo </ns1:fname>

  </rdf:Description>

</rdf:RDF>
```

Figure 4.1: (a) Part of example RDF schema in XML (b) Part of example RDF instance in XML

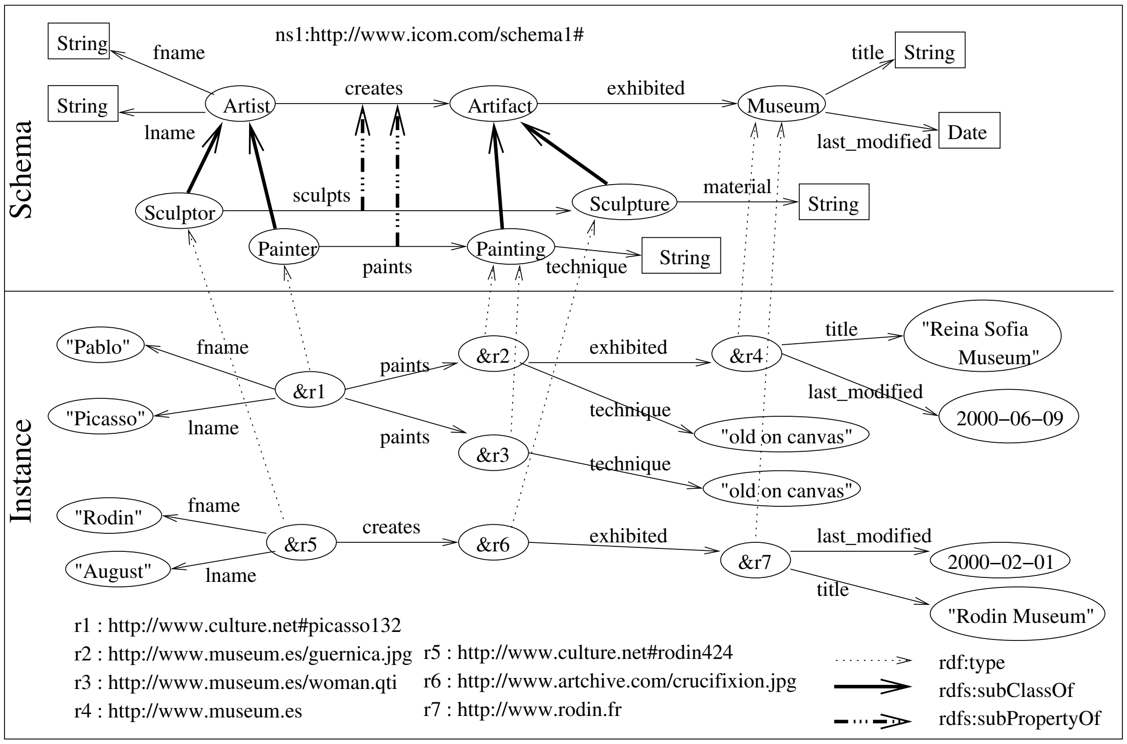


Figure 4.2: An RDF example for a Museum Artifact Catalog describing artifact resources.

ask the following RDQL query.

Example 4.1.1 *SELECT ?sculpture, ?museum*

WHERE (?sculptor, <ns1:lname>, "Rodin"),

(?sculptor, <ns1:creates>, ?sculpture),

(?sculpture, <ns1:exhibited>, ?museum),

(?museum, <ns1:last_modified>, ?date)

AND ?date < 2001-01-01

USING ns1 FOR <http://www.icom.com/schema1#>

The result of this query contains $\{(\&r6, \&r7)\}$.

Note that the WHERE clause is just shorthand for a logical statement. For example, in the above query, the WHERE clause is RDF-notation for the logical query:

$in(S, http://www.icom.com/schema1\#lname, \text{“}Rodin''\text{”}) \wedge$

$in(S, http://www.icom.com/schema1\#creates, S') \wedge$

$in(S', http://www.icom.com/schema1\#exhibited, M) \wedge$

$(M, http://www.icom.com/schema1\#last\_modified, D)$.

The variables S, S', M, D in the above logical statement correspond exactly to the variables $?sculptor$, $?sculpture$, $?museum$ and $?date$ respectively in the preceding RDQL-query. As in logic, we want to find all bindings of variables *in* the RDF-instance (hence the predicate *in* above) that satisfy the query.

Formally, a WHERE-triple is an expression of the form (R, P, V) where R is either a resource or a variable ranging over resources, P is either a property or a variable ranging over properties, and V is either a value or a variable ranging over values. In RDQL, variables are strings that start with the "?" symbol.

Likewise, an AND-constraint is an expression of the form $?x$ *op* $?y$ where $?x, ?y$ are both variables or an expression of the form $?x$ *op* v where $?x$ is a variable and v is a value. In either case, *op* $\in \{ =, \neq, <, \leq, >, \geq,$ eq, ne, $\}$. The first kind of constraint is often called a *join constraint*, while the second is often called a *value constraint*.

Formally, an RDQL-query has the syntax:

SELECT V

WHERE $w_1, \ldots, w_n$

AND $a_1, \ldots, a_m$

USING u

where V is a sequence of variables, $w_1, \ldots, w_n$ are WHERE-triples, $a_1, \ldots, a_m$ are AND-constraints and u denotes a URI-prefix.

As is common in databases, a view merely consists of an RDQL query together with a request to create the view. This is expressed by: "CREATEVIEW view_name AS rdql_query". The materialized view is a *table* whose columns are the items in the SELECT clause of the RDQL query in the view definition, and rows are sets of values satisfying the query. I do not restrict views (like [79]) to be defined only on a subset of queries that return results (i) containing class instances

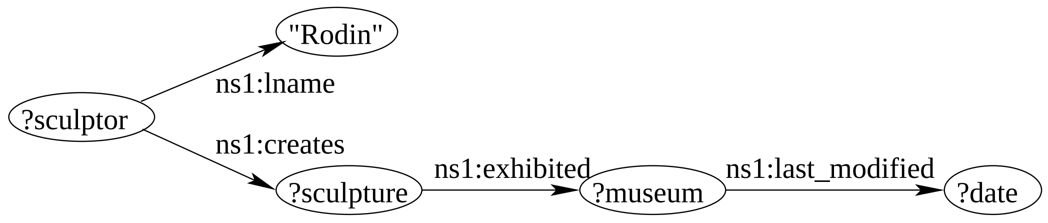


Figure 4.3: An example graph pattern

(i.e., a subject or object variable), or (ii) in the pattern of RDF statement (i.e., a triple containing subject, predicate and object).

Because RDQL is a graph matching language, each RDQL query has an associated graph pattern[72]. A *graph pattern (GP) for a given* RDQL *view* Q is a graph whose nodes are labeled by a set of constraints such that:

1. for each triple (r, p, v) in the WHERE clause of Q, r and v are nodes in the graph, and there is an edge from r to v in the graph;

2. if two triples share a variable or resource or literal, the two nodes for the same variable or resource or literal will be collapsed into one while preserving their edges.

3. if C is a value constraint in the AND clause of the form $?x \ op \ v$ then C is in the set of value constraints associated with the node $?x$.

Figure 4.3 shows the corresponding graph pattern for the query shown in Example 4.1.1. Notice that the constraint $?date < $ 2001-01-01 associated with the node $?date$ is not shown in the figure. Although a root at `?sculptor` exists in Figure 4.3, roots may not always exist as the reader can see from the following example.

Example 4.1.2 *Suppose we want to find some painter who painted a portrait of someone such that that someone was a student of that painter. This can be answered by the following* RDQL*-query:*

SELECT ?painter

WHERE (?painter, <ns1:paints>, ?painting),

(?painting, <ns1:portrait_of>, ?model),

(?model, <ns1:student>, ?painter)

USING ns1 FOR <http://www.icom.com/schema1#>

The above query forms a cyclic graph pattern without a root.

A graph pattern may include more than one connected component. My algorithms work well when one connected component is present. They may be less efficient when multiple connected components are present.

4.2 Maintenance Algorithms

4.2.1 Insertion Maintenance Algorithms

I am now ready to consider the problem of maintaining views over RDF-instances when a triple is added to the RDF-instance. As there is a one to one correspondence between RDF-instances and their corresponding graphs, I will often use these terms synonymously.

All insertions to RDF graphs can be represented via triples. For example, suppose we wish to insert the "Painter" resource (http://www.culture.net#matisse) associated with Matisse. This may be accomplished by inserting the following triple "(http://www.culture.net#matisse, <rdf:type>, <ns1:Painter>)". When a new triple is inserted, the algorithm will detect the new delegate objects[1] to be inserted into the materialized view. If the inserted triple involves a new resource and/or value, it means that a new node is inserted with a new edge into the RDF graph.

My IMA algorithm uses the important notion of triple unification, graph unification and pattern matching between the graph pattern of query and an RDF instance graph which is somewhat different from the classical notion of unification in logic [57]. Traditionally, a *substitution* is a mapping from the set of variables to a set of terms, where a term is a variable or a constant; a substitution θ is a unifier of two terms t_1, t_2 iff $t_1\theta = t_2\theta$[2].

Definition 4.2.1 (triple-unification) *Suppose* (R_1, P_1, V_1) *is a* WHERE*-triple and* (R_2, P_2, V_2)

[1]The original database has some objects with IDs. A delegate object is a version of the original object and preserves a link to the original object. Due to space limitation, we do not go into details of delegate objects here.

[2]$t_1\theta$ is the expression obtained by applying θ to t_1.

is an instance triple (i.e., a triple in an instance). We say that (R_1, P_1, V_1) triple-matches (R_2, P_2, V_2) *iff there exists a substitution* θ *such that:*

1. $R_1\theta = R_2\theta$ *or* $R_2\theta$ *is a subclass of* $R_1\theta$

2. $P_1\theta = P_2\theta$ *or* $P_2\theta$ *is a subproperty of* $P_1\theta$

3. $V_1\theta = V_2\theta$ *or* $V_2\theta$ *is a subclass of* $V_1\theta$

It is important to note that the *triple-matches* relationship is not symmetric (because of the subclass/subproperty condition in the above definition). The following example illustrates this.

Example 4.2.1 *Consider the following* WHERE*-triple* t_1*= (?painter, <ns1:creates>, ?painting) and the following instance triple* t_2*= (http://www.culture.net#picasso132, <ns1:paints>, http://www.museum.es/guernica.jpg).* t_1 *triple-matches* t_2 *because there exists a substitution* $\theta = \{(?painter) \rightarrow http://www.culture.net\#picasso132,$ $?painting \rightarrow http://www.museum.es/guernica.jpg\}$ *and <ns1:paints> is a subproperty of <ns1:creates>. However,* t_1 *does not triple-match* t_2*.*

The reason for this asymmetry is that given a WHERE-triple in a query (or a graph pattern), we want to find instances of it in the RDF-instance that satisfy the conditions of the query (rather than the other way around). The RDF-instance of course has no variables in it, while the query might have zero or more variables in it.

Similarly, I define a unifier of two labeled graphs as follows. Given a labeled graph $G(V, E)$ (e.g., an RDF graph), I define $l(v_1, v_2)$ to be the label (e.g., a property) associated with (v_1, v_2) for all $(v_1, v_2) \in E$.

Definition 4.2.2 (Graph Unifier) *A substitution* θ *unifies a labeled graph* $G_1 = (V_1, E_1)$ *with another labeled graph* $G_2 = (V_2, E_2)$*, i.e.* $G_1\theta = G_2\theta$*, iff there exists a bijection* ϕ *between* V_1 *and* V_2 *such that*

1. v_1 *is unifiable with* $\phi(v_1)$ *using substitution* θ *for all* $v_1 \in V_1$ *and*
2. *if* $(v_1, v_2) \in E_1$*, then* $(\phi(v_1), \phi(v_2)) \in E_2$*, and vice versa, and*

3. $l(v_1, v_2)$ is unifiable with $l(\phi(v_1), \phi(v_2))$ using substitution θ for all $(v_1, v_2) \in E_1$.

Based on this definition, I can define a matching between a graph pattern GP of query Q and a subgraph $I'(V', E')$ of an RDF graph $I(V, E)$ ($I'(V', E')$ itself is a graph and $V' \subseteq V$, $E' \subseteq E$).

Definition 4.2.3 (Pattern Matching) *Suppose Q is a query, $GP(V_1, E_1)$ is the graph pattern of Q, and $I(V_2, E_2)$ is an* RDF *graph. A substitution θ matches $GP(V_1, E_1)$ with a subgraph $I'(V_2', E_2')$ of I iff*

1. $GP\theta = I'\theta$ and [3]

2. θ satisfies the constraints in Q.

Example 4.2.2 *Consider the view defined in Example 4.1.1. There is one subgraph of the* RDF-*instance in Figure 4.2 which unifies with the graph pattern of the view via the following substitution:* $\{?sculptor \rightarrow \&r5, ?sculpture \rightarrow \&r6, ?museum \rightarrow \&r7, ?date \rightarrow \text{“}2000-02-01''\}$.

My IMA algorithm uses the following subroutines.

- *BuildGP(Q)* constructs a graph pattern GP from view definition Q.
- *TMatch(GP, t)* returns "true" for a triple t and a graph pattern GP if a *WHERE*-triple w in GP triple-matches t, i.e., if there exists a subsitution such that t unifies with a triple in GP.
- Given GP, Q, a triple t and the updated RDF graph $(I \cup t)$,[4] the subroutine *MSearch(GP, Q, t, $I \cup t$)* returns the set $\{\theta \mid$ there exists a WHERE-triple t' in GP and a triple set T in $(I \cup t)$ such that, via substitution θ, (i) t' triple-matches t and (ii) $\forall t_1 \in GP$, $\exists t_2 \in T$, t_1 triple-matches t_2, and (iii) θ satisfies the WHERE and AND clauses in $GP\}$, i.e., it finds all substitutions θ that match GP with some subgraph of I that contains t.

[3]For a value (i.e., non-variable) to unify with another value (non variable), instead of the original requirement that both values must be identical, we allow a value v_1 in a triple in GP to unify with another value v_1' in a triple in I' if and only if (1) v_1 is the same as v_1'; or (2) v_1' is v_1's subclass if both of them are classes; or (3) v_1' is v_1's subproperty if both of them are properties.

[4]For simplicity, we use $I \cup t$ instead of $I \cup \{t\}$, and $I - t$ instead of $I - \{t\}$.

- Given a substitution θ, and output variables (specified in a SELECT clause) $X = (?x_1, \ldots, ?x_n)$, we define *VRetrieve(θ, GP, X)* to retrieve values $XV = (\theta(?x_1), \ldots, \theta(?x_n))$ for X from $MSearch(GP, Q, t, I \cup t)$.

Figure 4.4 presents the IMA algorithm (short for **Insertion Maintenance Algorithm**) to find new delegate objects when an insertion occurs.

```
algorithm IMA($I, Q, M(Q, I), t$)
/* Input: RDF graph $I$, view specification $Q$, materialized view $M(Q, I)$, inserted triple $t$ */
/* Output: $M(Q, I \cup t)$ */
1)  $GP \leftarrow$ BuildGP($Q$); $X \leftarrow$ output variables of $Q$;
2)  if TMatch($GP$, $t$) == TRUE, then
3)      $\Delta M \leftarrow \{$VRetrieve$(\theta, GP, X) \mid \theta \in MSearch(GP, Q, t, I \cup t)\}$;
4)  return $M(Q, I \cup t) \leftarrow M(Q, I) \cup \Delta M$;
```

Figure 4.4: Insertion Maintenance Algorithm IMA

Example 4.2.3 *Consider the* RDF *graph I of Figure 4.2 and the view Q in Example 4.1.1. The materialized view M(Q,I) is* $\{(\&r6, \&r7)\}$. *Suppose we insert* $t = (\&r5,$ *<ns1:creates>*$, \&r2)$. IMA *first builds a graph pattern GP (Figure 4.3). The output variables are* $X = (?sculptor, ?museum)$. *TMatch*(GP, t) *returns true via the substitution* $\{t/(?sculptor,$ *<ns1:creates>*$, ?sculpture)\}$ *in* GP. $MSearch(GP, Q, t, I \cup t)$ *returns the substitution:* $?sculptor \rightarrow \&r5$*,* $?sculpture \rightarrow \&r2$*,* $?museum \rightarrow \&r4$*,* $?date \rightarrow$ *"2000-06-09". For this substitution, VRetrieve*(θ, GP, X) *returns* $(\&r2, \&r4)$. *As a result,* $M(Q, I \cup t) = M(Q, I) \cup \{(\&r2, \&r4)\}$.

Complexity Analysis of IMA. The time taken to execute BuildGP(Q) in line 1 is linear in the number of triples in the WHERE clause and the number of constraints in the AND clause. If we materialize or cache graph patterns, we can retrieve the stored copy instead of calling BuildGP(Q). In lines 2 and 3, both TMatch(GP, t) and VRetrieve(θ, GP, X) take linear time w.r.t. $|GP|$, the number of edges in the graph pattern.

```
algorithm $MSearch(GP, Q, t, I)$
/* Input: graph pattern $GP$, view specification $Q$, triple $t$, RDF graph $I$ */
/* Output: $\Theta = \{\theta \mid \theta$ is a substitution$\}$ */
1)  $\Theta \leftarrow \emptyset$;
2)  for each $t' \in GP$ s.t. $\exists \theta', t\theta' = t'\theta'$,
3)      for each $\theta \in bSearch(t, t', GP, I)$,
4)          if $\theta$ satisfies the constraints in $Q$, then $\Theta \leftarrow \Theta \cup \theta$;
5)  return $\Theta$;
```

Figure 4.5: Algorithm for *MSearch*

In line 3, I can modify a breadth-first search for $MSearch$ starting with triple t. Basically, by substituting t for some WHERE-triple t' in GP, we can search for a substitution of GP to a subgraph of $I \cup t$ containing t by a breath-first search starting at t'. There may be more than one possible way of assigning triples in GP to t in GP (say, from t to a triple t' in GP). This means that each time, we may have to search starting at t for a substitution (θ) to $I \cup t$ from the graph pattern GP rooted at each possible t'. Figure 4.5 presents the algorithm for $MSearch(GP, Q, t, I)$ to find all such substitutions θ. $bSearch(t, t', GP, I)$ denotes the breadth-first search starting with some $t' \in GP$ to find the subgraphs of I (containing t) that GP substitutes to.

In addition, checking whether substitution θ satisfies the first type of AND-constraint can be done in linear time w.r.t. the number of AND-constraints in the AND clause. Thus, the time complexity of $MSearch$ is $O(|GP|b^{D(GP)})$ where b is the average branching factor of I (considering both in and out degrees) and $D(GP)$ indicates the depth of graph pattern as an undirected graph. Therefore, the worst-case time complexity of the IMA algorithm is $O(|GP|\ b^{D(GP)})$. As graph patterns are usually not too deep (3 or 4 perhaps), this is not unreasonable.

Speeding up IMA. IMA's performance can be improved by improving $bSearch$ in $MSearch$ in two ways. (i) First, the value constraints associated with the nodes in pattern GP should be used as quick filters when testing whether GP triples can triple-match a set of triples in graph $I \cup t$. If we start from a possible substitution for t and find that a particular triple in GP can triple-match no triple in $I \cup t$, we can immediately stop searching this path. If this happens for all possible substitutions, we can conclude that the update of I does not affect $M(Q, I)$. (ii) Second, in the graph $I \cup t$, the properties of a node can be classified via a namespace. For a triple t' in GP, we need only search the triples in $I \cup t$ whose property has the same namespace as the property of t'. For example, the node &$r5$ in Figure 4.2 has properties in namespace ns1 and rdf. Suppose a triple in the graph pattern is (?sculptor, <ns1:creates>, ?sculpture). For &$r5$, we only need to consider the properties in namespace ns1. This is useful when the branching factor b is large and there are several namespaces in I.

Figure 4.6 presents the pseudocode for this enhanced $bSearch(t, t', GP, I)$, modified from a standard breadth-first search algorithm. Here I treat triples as vertices in a graph. Two vertices (triples) are connected if one triple's resource is the other's value. Just like the standard breadth-first search, *bSearch* colors each vertex in GP to keep track of the search progress. A vertex is initially white, but becomes gray or black after it is discovered (encountered in the search). Black vertices have all neighbors non-white (discovered) while gray vertices may have some white (non-discovered) neighbors. The following additional data structures are used. *Adj* is an adjacency list where $Adj[u]$ returns the neighbors of vertex u. $color[u]$ and $\pi[u]$ store the color (white, gray or black) and predecessor of vertex u. $d[u]$ denotes the distance (the smallest number of edges) from the starting vertex t'. I use a first-in-first-out queue Q to store gray vertices.

The following theorem says that IMA is correct.

Theorem 4.1 *Given an RDF graph I, view specification Q, an inserted RDF triple t, the result $M(Q, I) \cup \Delta M$ of the algorithm* $\mathsf{IMA}(I, Q, M(Q, I), t)$ *is equivalent to the complete recomputation of the view $M(Q, I \cup t)$ from $I \cup t$.*

Proof

For every row r of values in $M(Q, I)$, there exists at least one substitution θ from GP to I satisfying Q that outputs r. As $\theta(GP)$ is a subset of I, after adding any nodes or edges into I, all original rows in $M(Q, I)$ will still exist in $M(Q, I \cup t)$.

```
algorithm $bSearch(t, t', GP, I)$
/* Input: instance triple $t$, WHERE-triple $t'$, graph pattern $GP$, RDF graph $I$ */
/* Output: $\Theta_{t'} = \{\theta \mid \theta$ is a substitution$\}$ */
1)  for each vertex $u \in GP - \{t'\}$,
2)      $color[u] \leftarrow$ WHITE;
3)      $d[u] \leftarrow \infty$;
4)      $\pi[u] \leftarrow$ NIL;
5)  $color[t'] \leftarrow$ GRAY;
6)  $d[t'] \leftarrow 0$;
7)  $\pi[t'] \leftarrow$ NIL;
8)  $Q' \leftarrow \{t'\}$;
9)  create a mapping $M$ s.t. $M(t') = t$;
10) while $Q' \neq \emptyset$,
11)     $u' \leftarrow head[Q']$;
12)     for each $v' \in Adj[u']$,
13)         if $color[v']$ = WHITE, then
14)             $color[v'] \leftarrow$ GRAY;
15)             $d[v'] \leftarrow d[u'] + 1$;
16)             $\pi[v'] \leftarrow u'$;
17)             Enqueue($Q', v'$);
18)             for each mapping $M$,
19)                 for each pair $(v', v)$ where $v \in Adj[M(u')]$,
20)                     if (i) $v'$ and $v$ are in the same namespace, (ii) $v'$ triple-matches $v$ and
                            (iii) if $v$ contains a value, $v$ satisfies all value constraints associated with $v'$, then
                                create a new mapping $M' \leftarrow M$; $M'(v') \leftarrow v$;
21)                 delete $M$;
22)             if no $M$ exists, return $\Theta_{t'} = \emptyset$;
23)     Dequeue($Q'$);
24)     $color[u'] \leftarrow$ BLACK;
25) return $\Theta_{t'} = \{\theta | \theta$ is a substitution corresponding to some mapping $M$ $\}$;
```

Figure 4.6: Algorithm for enhanced *bSearch*

Suppose $\Delta M' = M(Q, I \cup t) - M(Q, I)$. To prove the theorem is equivalent to prove that $\Delta M' = \Delta M - M(Q, I)$, i.e. the additional rows in $M(Q, I \cup t)$ but not in $M(Q, I)$ are equivalent to the rows in ΔM but not in $M(Q, I)$.

$\Delta M'$ consists of all rows r' of values such that for every r', there exists at least one substitution θ' from GP to $I \cup t$ satisfying Q that outputs r'. Note that $t \in \theta'(GP)$. If this were not true, there would be a substitution θ that outputs a row $r \in M(Q, I)$ such that $\theta = \theta'$. Since IMA find all substitutions from GP to $I \cup t$ such that t is included and Q is satisfied, IMA find all the possible substitutions that contribute to $\Delta M'$. On the other hand, IMA does not return substitutions that do not include t, so it does not return substitutions that output rows of values not existing in $M(Q, I) \cup \Delta M'$, i.e., $M(Q, I \cup t)$. The reason is that the output rows of values of the substitutions that include t must be captured by $\Delta M'$ unless they are already included in $M(Q, I)$.

Therefore, $\Delta M' = \Delta M - M(Q, I)$. ∎

4.2.2 Deletion Maintenance Algorithm

I consider triple deletion in this section. When we delete a node from an RDF graph, I assume that all edges connected to it are deleted automatically. I will discuss resource deletion in Section 4.2.4.

Figure 4.7 presents the DMA algorithm (DMA stands for **Deletion Maintenance Algorithm**). DMA uses the same functions used by IMA. In addition, it uses a new function $MSearch2(GP, Q, r, I - t, X)$ which is just like $MSearch$ except for one difference: $MSearch2$ returns true if there exists at least one substitution θ such that (1) $r == VRetrieve(\theta, GP, X)$; (2) θ satisfies the value constraints in GP; (3) θ satisfies the join constraints in Q. Thus, $MSearch2$ returns true if there exists some substitution such that (i) every triple in GP triple-matches some triple in $I - t$, (ii) Q is satisfied and (iii) the same row r in $M(Q, I)$ can be returned. The algorithm *MSearch2* is shown in Figure 4.8.

Example 4.2.4 *Consider the* RDF *graph I of Figure 4.2 and the view Q in Example 4.1.1. The materialized view $M(Q, I)$ is $\{(\&r6, \&r7)\}$. Suppose the triple t = $(\&r5,$ <ns1:creates>,*

```
algorithm DMA($I, Q, M(Q, I), t$)
/* Input: RDF graph $I$, view specification $Q$, materialized view $M(Q, I)$, deleted triple $t$ */
/* Output: $M(Q, I - t)$ */
1)  if $M(Q, I) = \emptyset$, then return $M(Q, I - t) \leftarrow \emptyset$;
2)  $GP \leftarrow$ BuildGP($Q$); $\Delta M \leftarrow \emptyset$;
3)  $X \leftarrow output\ variables\ of\ Q$;
4)  if TMatch($GP$, $t$) == TRUE, then
5)      $\Delta M \leftarrow \{VRetrieve(\theta, GP, X) \mid \theta \in MSearch(GP, Q, t, I)\}$;
6)      $\forall r \in \Delta M$,
            if $MSearch2(GP, Q, r, I - t, X)$ == TRUE, then $\Delta M \leftarrow \Delta M - r$;
7)  return $M(Q, I - t) \leftarrow M(Q, I) - \Delta M$;
```

Figure 4.7: Deletion Maintenance Algorithm DMA

```
algorithm $MSearch2(GP, Q, r, I, X)$
/* Input: graph pattern $GP$, view specification $Q$, row $r$, RDF graph $I$, output variables $X$ */
/* Output: TRUE or FALSE */
1)  for each $t' \in GP$, $t \in I$ s.t. $\exists \theta', t\theta' = t'\theta'$,
2)      for each $\theta \in bSearch(t, t', GP, I)$,
3)          if $\theta$ satisfies the constraints in $Q$ AND
                $r == VRetrieve(\theta, GP, X)$, then return TRUE;
4)  return FALSE;
```

Figure 4.8: Algorithm for $MSearch2$

&$r6$) *is deleted.* DMA *first builds a graph pattern GP (Figure 4.3). The output variables are $X = (?sculptor, ?museum)$. $TMatch(GP, t)$ returns true because we can have t to unify ($?sculptor$, <ns1:creates>, $?sculpture$) in GP. $MSearch(GP, Q, t, I)$ returns the substitution: $?sculptor \rightarrow \&r5$, $?sculpture \rightarrow \&r6$, $?museum \rightarrow \&r7$, $?date \rightarrow$ "2000-02-01". For this substitution, VRetrieve(θ, GP, X) returns $(\&r6, \&r7)$. Thus, $\Delta M = \{(\&6, \&7)\}$. $MSearch2$ then tries to find a substitution from GP to $I - t$ such that Q is satisfied with $(\&6, \&7)$ as output. Because there does not exist such a substitution, we will delete $(\&6, \&7)$ from $M(Q, I)$. The resulting $M(Q, I - t)$ is empty.*

Complexity Analysis of DMA. The main difference between $MSearch2$ and $MSearch$ is that the starting point of the breadth-first search can be any node (subject/object in a triple) specified in r. During the search, the values of nodes and edges (corresponding to those output variables) must match those in GP specified by r. Similarly, the constraints and non-variable nodes/edges are quick filters and reduce the actual search space even though the degrees of nodes are large. If r contains only edge values (i.e., edge labels in a graph or predicate in triples) $(l_1, \ldots, l_n)$, then we can find (by using an index), the count $|l_i|$ of l_i in $I - t$ $\forall i = 1, \ldots, n$. Suppose argmin($\{|l_i| \mid i = 1, \ldots, n\}$) $= j$. I use $|l_{min}|$ to denote $|l_j|$. We can then start our breadth-first search from each edge (with label l_j) in turn. We can stop as soon as a substitution is found that outputs r, satisfies Q and returns TRUE, or until all edges with label l_j are tried. The worst-case time complexity of $MSearch2$ is $O(|l_{min}|b^{D(GP)})$ where $|l_{min}| = 1$ when r also contains some node, b is the average branching factor of I (considering both in and out degrees) and $D(GP)$ is the depth of graph pattern as an undirected graph. Therefore, the worst-case time complexity of the DMA algorithm is $O((|GP| + |\Delta M||l_{min}|)b^{D(GP)})$.

Theorem 4.2 *Given an RDF graph I, view specification Q, a deleted RDF triple t, the result $M(Q, I) - \Delta M$ of the algorithm* DMA*$(I, Q, M(Q, I), t)$ is equivalent to the complete recomputation of the view $M(Q, I - t)$ from $I - t$.*

Proof

Suppose $\Delta M' = M(Q, I) - M(Q, I - t)$. To prove the theorem is equivalent to prove that $\Delta M' = \Delta M$, i.e. the original rows in $M(Q, I)$ but not in $M(Q, I - t)$ are equivalent to the rows in ΔM.

$\Delta M'$ consists of all rows r' of values such that for every r', there exists at a substitution θ' from GP to I satisfying Q that outputs r' but there exists no substitution θ'' from GP to $I - t$ satisfying Q that outputs r'. All θ' above must contain t (i.e., $t \in \theta'(GP)$) because if one such θ' does not contain t, then we can substitute by θ' from GP to $I - t$ satisfying Q that outputs r'.

ΔM of line 5 consists of all rows r such that for every r, there exists a substitution θ from GP to I satisfying Q that contains t and outputs r. After line 6, ΔM excludes those rows r which has at least one substitution θ from GP to $I - t$ satisfying Q that does not contain t but outputs r. In the other words, ΔM consists of all rows r such that for every r, (1) there exists at least one substitution θ from GP to I satisfying Q that outputs r; (2) all such θ must contain t; (3) there exists no substitution θ'' from GP to $I - t$ satisfying Q that outputs r.

Therefore, $\Delta M' = \Delta M$. ∎

4.2.3 Triple Modification Algorithm

An atomic modification update to a set of RDF statements falls into one of five categories:

1. a triple's resource changes;
2. a triple's property changes;
3. a triple's value which is a resource changes;
4. a triple's value which is not a resource changes;
5. a resource itself (the change of rdf:about) from R to R' which will cause all edges connecting *to or from* R to change to connect *to or from* R'.

I consider these cases one by one.

One straightforward way to handle cases (1) to (4), is to process the modification as a deletion of the old triple t_{del} and an insertion of a new triple t_{ins} (with the updated subject,

```
algorithm TMA($I, Q, M(Q,I), t_{del}, t_{ins}$)
/* Input: RDF graph $I$, view specification $Q$, materialized view $M(Q,I)$,
          deleted triple $t_{del}$, inserted triple $t_{ins}$ */
/* Output: $M(Q,(I - t_{del}) \cup t_{ins})$ */
1)   $GP \leftarrow$ BuildGP($Q$); $\Delta M_d \leftarrow \emptyset$;
2)   $X \leftarrow output\ variables\ of\ Q$; $F = 0$;
3)   if TMatch($GP$, $t_{del}$) == TRUE, then $F \leftarrow F + 1$;
4)   if TMatch($GP$, $t_{ins}$) == TRUE, then $F \leftarrow F + 2$;
5)   if $F == 0$, then return $M(Q,(I - t_{del}) \cup t_{ins}) \leftarrow M(Q,I)$;
6)   if $F == 1$, then return $M(Q,(I - t_{del}) \cup t_{ins}) \leftarrow DMA(I \cup t_{ins}, Q, M(Q,I), t_{del})$;
7)   if $F == 2$, then return $M(Q,(I - t_{del}) \cup t_{ins}) \leftarrow IMA(I - t_{del}, Q, M(Q,I), t_{ins})$;
8)   $\Delta M_i \leftarrow \{$VRetrieve$(\theta, GP, X) \mid \theta \in MSearch(GP, Q, t_{ins}, (I - t_{del}) \cup t_{ins})\}$;
9)   if $M(Q,I) \neq \emptyset, then$
10)      $\Delta M_d \leftarrow \{$VRetrieve$(\theta_1, GP, X) \mid \theta_1 \in MSearch(GP, Q, t_{del}, I)\}$;
11)  $\Delta M_d \leftarrow \Delta M_d - \Delta M_i$;
12)  $\forall r \in \Delta M_d$,
         if $MSearch2(GP, Q, r, I - t_{del} \cup t_{ins}, X)$ == TRUE, then $\Delta M_d \leftarrow \Delta M_d - r$;
13)  return $M(Q,(I - t_{del}) \cup t_{ins}) \leftarrow (M(Q,I) - \Delta M_d) \cup \Delta M_i$;
```

Figure 4.9: Triple Modification Maintenance Algorithm TMA

predicate or object). Fortunately, I can do better by first finding the rows to be inserted into $M(Q, I)$ as a result of adding t_{ins} using IMA before executing line 5 of DMA. The rows to be both deleted (potentially) and inserted should be kept in the final answer. Figure 4.9 presents the TMA algorithm (short for **Triple Modification Maintenance Algorithm**) to make necessary updates to a materialized view when a modification occurs in a triple.

Example 4.2.5 *Consider the RDF graph I of Figure 4.2, the view Q in Example 4.1.1. The materialized view $M(Q, I)$ is $\{(\&r6, \&r7)\}$. Suppose the triple $t_{del} = (\&r5, \text{<ns1:creates>}, \&r6)$ is modified to become $t_{ins} = (\&r5, \text{<ns1:paints>}, \&r6)$. TMA first builds a graph pattern GP (Figure 4.3). The output variables are $X = (?sculptor, ?museum)$. Both TMatch(GP, t_{del}) and TMatch(GP, t_{ins}) return true. The insertion of t_{ins} leads to an inserted row $\Delta M_i = \{(\&r6, \&r7)\}$. $MSearch(GP, Q, t_{del}, I)$ returns a substitution θ_1 as follows (only variables are shown): ?sculptor $\rightarrow$ &r5, ?sculpture $\rightarrow$ &r6, ?museum $\rightarrow$ &r7, ?date $\rightarrow$ "2000-02-01". For this substitution, VRetrieve(θ_1, GP, X) returns $r_d = (\&r6, \&r7)$. As $r_d \in \Delta M_i$, so $r_d \notin \Delta M_d$. As $M(Q, I) = \{(\&r6, \&r7)\}$, $\Delta M_d = \emptyset$, $\Delta M_i = \{(\&r6, \&r7)\}$, $M(Q, (I - t_{del}) \cup t_{ins}) = M(Q, I)$.*

Complexity Analysis of TMA. The time complexity is the sum of those of IMA and DMA, i.e. $O(|GP|b^{D(GP)} + (|GP| + |\Delta M_d|\ |l_{min}|)\ b^{D(GP)}) = O((|GP| + |\Delta M_d||l_{min}|)b^{D(GP)})$ (same as DMA).

Theorem 4.3 *Given an RDF graph I, view specification Q, an RDF triple updated from t_{del} to t_{ins}, the result $(M(Q, I) - \Delta M_d) \cup \Delta M_i$ of the algorithm TMA($I, Q, M(Q, I), t_{del}, t_{ins}$) is equivalent to the complete recomputation of the view $M(Q, (I - t_{del}) \cup t_{ins})$ from $(I - t_{del}) \cup t_{ins}$.*

Proof

Basically, TMA uses IMA to find the set ΔM_i of rows to add into the view and then uses DMA to find the set ΔM_d of rows to remove from the view. The additional things are from lines 6-7 and 11. Lines 6-7 simply calls DMA if it is impossible to have new rows to add, or calls IMA if it is impossible to have old rows to remove. Line 11 simply removes from ΔM_d those common rows in ΔM_d and ΔM_i. It is correct because in line 12, those common rows removed will be added back. As DMA and IMA are proved correct, TMA is also correct. ∎

4.2.4 Resource Modification Algorithm

In this section, I consider the fifth kind of modification to an RDF-instance. This is the case where a resource is changed from R_d to R_i. In this case, we need to delete rows from the original view associated with R_d and insert rows associated with R_i.

Suppose T_d is the set of edges connected to R_d. One straightforward approach is to call IMA and DMA (for inserting and deleting the edges one by one) $|T_d|$ times.

Fortunately, we can instead consider the deletion and insertion of all those edges simultaneously. In this second approach, we consider the substitutions of the deleted/inserted resource to the resource/value of a triple in GP instead of the substitutions of the deleted/inserted edges to a triple in GP. In this case, the worst case time complexity will not depend on $|T_d|$.

Figure 4.10 presents the RMA algorithm (short for **Resource Modification Maintenance Algorithm**) to make necessary updates to a materialized view when a modification occurs in a resource. I use $I_{R_d \to R_i}$ to denote the updated RDF graph after replacing R_d by R_i and $M(Q, I_{R_d \to R_i})$ as the updated view.

My RMA algorithm uses the following new subroutines.

- *TMatchR(GP, R)* returns "true" for a resource R and a graph pattern GP if R unifies with a GP triple's subject or object.
- Given GP, Q, a resource R and an RDF graph I, I define *MSearchR(GP, Q, R, I)* to find all substitutions θ, such that (1) for every triple t (or node) in GP, there exists a triple t' (or node) in I such that t triple-matches some t' whose resource or value is R and (2) θ satisfies value constraints in GP; (3) θ satisfies join constraints in Q; (4) there is at least one variable in GP that is mapped onto R by θ.

In Figure 4.11, I present an algorithm to implement $MSearchR$.

Example 4.2.6 *Consider the RDF graph I of Figure 4.2, the view Q of Example 4.1.1. The materialized view $M(Q, I)$ is $\{(\&r6, \&r7)\}$. Suppose $R_d = \&r6$ and $R_i = \&r8$ where $\&r8$ refers to*

```
algorithm RMA($I, Q, M(Q,I), R_{del}, R_{ins}$)
/* Input: RDF graph $I$, view specification $Q$, materialized view, $M(Q,I)$,
          deleted resource $R_d$, inserted resource $R_i$ */
/* Output: $M(Q, I_{R_d \to R_i})$ */
1)  $GP \leftarrow$ BuildGP($Q$); $\Delta M_d \leftarrow \emptyset$; $\Delta M_i \leftarrow \emptyset$;
2)  $X \leftarrow output\ variables\ of\ Q$; $F = 0$;
3)  if TMatchR($GP$, $R_d$) == TRUE, then $F \leftarrow F + 1$;
4)  if TMatchR($GP$, $R_i$) == TRUE, then $F \leftarrow F + 2$;
5)  if $F == 0$, then return $M(Q, I_{R_d \to R_i}) \leftarrow M(Q,I)$;
6)  if $F > 1$, then $\Delta M_i \leftarrow \{$VRetrieve$(\theta, GP, X) \mid \theta \in MSearchR(GP, Q, R_i, I_{R_d \to R_i})\}$;
7)  if ($F == 1$ OR $F == 3$) AND $M(Q,I) \neq \emptyset$, then
8)      $\Delta M_d \leftarrow \{$VRetrieve$(\theta_1, GP, X) \mid \theta_1 \in MSearchR(GP, Q, R_d, I)\}$;
9)      $\Delta M_d \leftarrow \Delta M_d - \Delta M_i$;
10)     $\forall r \in \Delta M_d$,
          if $MSearch2(GP, Q, r, I_{R_d \to R_i}, X)$ == TRUE, then $\Delta M_d \leftarrow \Delta M_d - r$;
11) return $M(Q, I_{R_d \to R_i}) \leftarrow (M(Q,I) - \Delta M_d) \cup \Delta M_i$;
```

Figure 4.10: Resource Modification Maintenance Algorithm RMA

http://www.museum.es/man.jpg. RMA first builds a graph pattern GP (Figure 4.3). The output variables are $X = (?sculptor, ?museum)$. Both TMatchR(GP, R_d) and TMatchR(GP, R_i) return true. The insertion of R_i leads to an inserted row $\Delta M_i = \{(\&r8, \&r7)\}$. $MSearchR(GP, Q, R_d, I)$ returns a substitution θ_1 which gives output $\Delta M_d = \{(\&r6, \&r7)\}$. $MSearch2$ then cannot find another substitution to $I_{R_d \to R_i}$ to output $\{(\&r6, \&r7)\}$, so we need to remove that from the view. As a result, $M(Q, I_{R_d \to R_i}) = \{(\&r8, \&r7)\}$.

Complexity Analysis of RMA. TMatchR(GP, R) can be executed in linear time w.r.t. $|GP|$. $MSearchR$ is similar to $MSearch$ except that instead of starting with a substitution of the triple t in I to some triple in GP, we substitute the resource R for the subject/object of some triple in GP.

```
algorithm $MSearchR(GP, Q, R, I)$
/* Input: graph pattern $GP$, view specification $Q$, resource $R$, RDF graph $I$ */
/* Output: $\Theta = \{\theta \mid \theta$ is a substitution$\}$ */
1)    $\Theta \leftarrow \emptyset$;
2)    for each $t' \in GP$ s.t. $\exists\theta'$ and $t \in I$ where $R$ is a resource or value of $t$ and $t\theta' = t'\theta'$,
3)        for each $\theta \in bSearch(t, t', GP, I)$,
4)            if $\theta$ satisfies the constraints in $Q$, then $\Theta \leftarrow \Theta \cup \theta$;
5)    return $\Theta$;
```

Figure 4.11: Algorithm for $MSearchR$

The breadth-first search works similarly as $MSearch$. Since there may be at most $2|GP|$ different substitutions of R, the time complexity of $MSearchR$ is same as $MSearch$, i.e. $O(|GP|b^{D(GP)})$. Therefore, the worst-case time complexity of the RMA algorithm is same as TMA i.e., $O((|GP| + |\Delta M_d||l_{min}|)b^{D(GP)})$.

Theorem 4.4 *Given an RDF graph I, view specification Q, a resource updated from R_d to R_i, the result $(M(Q, I) - \Delta M_d) \cup \Delta M_i$ of the algorithm* RMA$(I, Q, M(Q, I), R_d, R_i)$ *is equivalent to the complete recomputation of the view $M(Q, I_{R_d \to R_i})$ from $I_{R_d \to R_i}$.*

Proof

Basically, TMA is a variation of RMA where TMatch and $MSearch$ in TMA are replaced by TMatchR and $MSearchR$ which work with a resource instead of a triple. Instead of calling DMA or IMA which deal with deletion or insertion of a single triple, RMA mandatorily uses $MSearchR$ to find ΔM_d and ΔM_i caused by deletion of T_d and insertion of T_i. As TMA is proved correct, so is RMA. ∎

4.3 RDF Aggregates

I illustrate the need for RDF aggregate operators with some motivating examples inspired by data from Christies' auction house. Figure 4.12's bottom half shows an example RDF-instance describing the auction data while the top half shows its RDF-schema. The following examples are based on this RDF-instance. I assume that unit conversions are already considered.

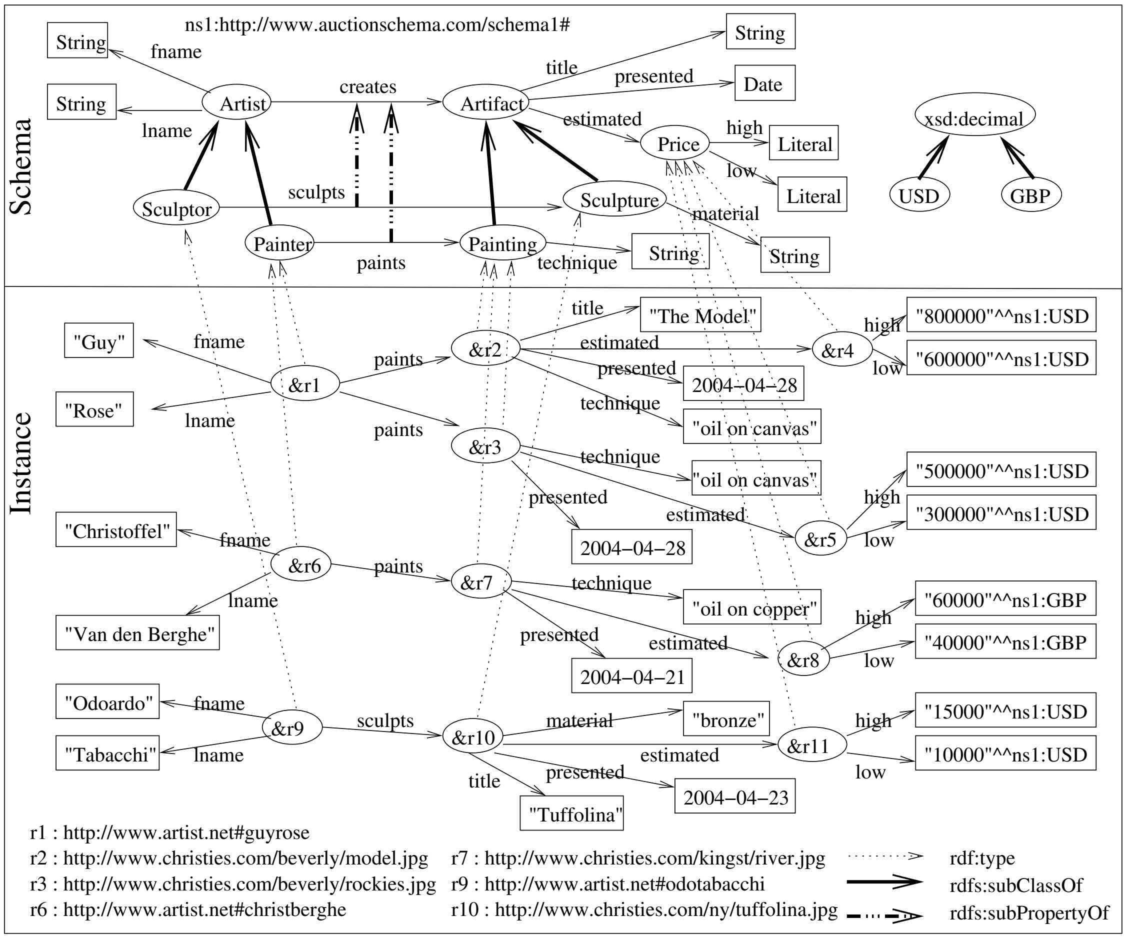


Figure 4.12: An RDF example describing artifact auctions in Christie's auction house

Example 4.3.1 (min) *Suppose Jack, a bidder, is going to buy an artifact but has a limited budget. He may want to know the minimum estimated price of the artifacts for auction in April, 2004. By comparing the values of the low property of the estimated prices for all artifacts in Figure 4.12, Jack will know that the minimum price is 10,000 USD.*

Example 4.3.2 (max) *Suppose Jack is interested in Impressionism. He may want to know the maximum estimated price of paintings by Guy Rose, a pure Impressionist. By comparing the values of the high property of the estimated prices for Guy Rose's works in Figure 4.12, he will know that the maximum estimated price is 800,000 USD.*

Example 4.3.3 (count) *As a fan of Guy Rose, Jack may want to know how many of his paintings will be presented for auction in April, 2004. By counting Guy Rose's works for auction in April, 2004, Jack will know that two of them are going to be sold.*

Example 4.3.4 (sum) *Suppose Smith is a manager of the auction house. He may want to know the prospective revenue by maximizing estimation for all the auctions in April, 2004. By summing up the values of the high property of the estimated prices of all the artifacts, Smith will know that the revenue is 1,425,244 USD approximately.*

Example 4.3.5 (average) *Suppose the auction house publishes price information on the web, including the average of the estimated high prices of the artifacts for each artist. By grouping the high prices and computing the average w.r.t. each artist respectively, the auction house can publish this information as follows:*

artist	**average of high prices**
Guy Rose	*650,000 USD*
Christoffel Van den Berghe	*60,000 GBP*
Odoardo Tabacchi	*15,000 USD*

I give the formal definition of an aggregate query below which is sufficiently generalized to include all kinds of aggregate queries .

Definition 4.3.1 (Aggregate Query) *Let I be an* RDF *graph, $x_1, x_2, \ldots, x_n$ be variables, and $o_1, o_2, \ldots, o_m$ be aggregate operators, where $o_i \in \{min, max, count, sum, average\}$, for i =1, . . . , m. An* RDF *aggregate query Q involving those variables and operators as well as an optional* GROUP BY *clause takes I as input and outputs a table T*[5] *containing a set of tuples.*

[5]The result of an aggregate query can be represented as an RDF instance or a table. The advantage of producing

Notice that the columns of the table T are the items in the SELECT clause of the query Q. If there is no GROUP BY clause and with the specific aggregate operators considered: *min, max, count, sum, average*, T normally contains only one tuple. The query Q_2 of Example 4.3.2 can be expressed in RDQL as follows:

Example 4.3.6 `SELECT max(?highprice)`

```
WHERE (?artist, <ns1:lname>, "Rose"),
(?artist, <ns1:fname>, "Guy"),
(?artist, <ns1:creates>, ?artifact),
(?artifact, <ns1:estimated>, ?price),
(?price, <ns1:high>, ?highprice),
(?artifact, <ns1:presented>, ?date)
AND 2004-04-01 <= ?date <= 2004-04-30
USING ns1 FOR <http://www.auctionschema.com/schema1#>
```

Here a string with the prefix "?" (e.g. `?artist`) represents a variable . Figure 4.13 illustrates the graph pattern of Q_2. The result of Q_2 is [("800000"^^ns1:USD)]. The symbol ^^ in a typed literal connects a value (such as "800000") and the literal's data type (such as the URI for USD)[71].

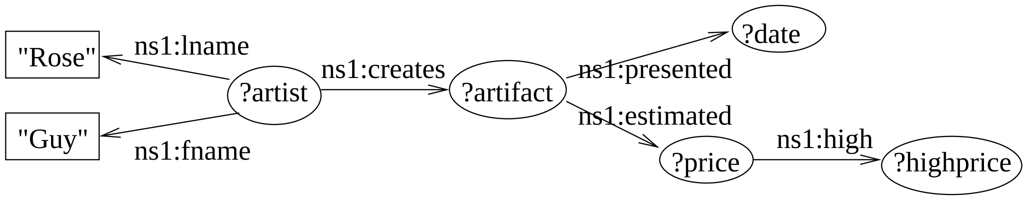


Figure 4.13: An example graph pattern

An RDF graph can be created as an answer returned by an aggregate query using an appropriate SELECT clause[6]. For example, Q_2 can return a valid RDF statement:

the result as an RDF instance is that it allows us to further query the result using the existing RDF query languages. However, I choose the tabular representation here for the sake of simplicity.

[6]I am expanding the syntax of RDQL so that it allows constants in SELECT clauses which equivalently creates new resources and properties using the constants.

(<ns1:works_by_guyrose>, <ns1:maxprice>, "800000"^^ns1:USD) if its SELECT clause is *SELECT <ns1:works_by_guyrose>, <ns1:maxprice>, max(?highprice).*

Example 4.3.7 *Consider query* Q_2 *in Example 4.3.6. There are two subgraphs of the* RDF-*instance in Figure 4.12 whose graph pattern unifies with that of* Q_2 *via the following two substitutions:* $\{?artist \rightarrow \&r1, ?artifact \rightarrow \&r2, ?price \rightarrow \&r4, ?highprice \rightarrow$ "800000"^^$ns1{:}USD$, $?date \rightarrow$ 2004-04-28$\}$ *and* $\{?artist \rightarrow \&r1, ?artifact \rightarrow \&r3, ?price \rightarrow \&r5, ?highprice \rightarrow$ "500000"^^$ns1{:}USD, ?date \rightarrow$ 2004-04-28$\}$. *The maximum value of* $?highprice$ *is 800,000 USD. As a result,* Q_2 *returns* $\{$*("800000"^^ns1:USD)*$\}$.

4.3.1 Algorithm to Compute Aggregates

Figure 4.14 presents an abstract algorithm CAA (short for **Compute Aggregates Algorithm**) to compute the result for aggregate queries. This algorithm is applicable to all aggregate queries including those with a GROUP BY clause.

```
algorithm $CAA(I, Q)$
/* Input: RDF graph $I$, query $Q$ */
/* Output: table $T(Q, I)$ */
1)    $GP \leftarrow$ BuildGP($Q$); $X \leftarrow aggregate\ variables\ of\ Q$;
2)    $Y \leftarrow GROUP\ BY\ variables\ of\ Q$;
3)    $S \leftarrow$ [VRetrieve($\theta, GP, X \cup Y$) $\mid \theta \in$ MSearchAll($GP, Q, I$)];
4)    return $T(Q, I) \leftarrow$ TCompute($S, Q$);
```

Figure 4.14: Compute Aggregates Algorithm CAA

The new subroutines introduced in CAA algorithm are as follows:

- Given GP, Q and the RDF graph I, *MSearchAll(GP, Q, I)* returns all most general substitutions that match GP with some subgraph of I.[7]
- Given a bag[8] S of values for variables $X \cup Y$, *TCompute(S, Q)* computes the aggregates as

[7]Only one substitution is returned for each matching in order to avoid double counting.

[8]Please note that I adapt the bag semantics for aggregation, i.e., we allow duplicates. I use [] to denote a bag.

specified by SELECT clause (and GROUP BY clause if any) specified in Q. As S can be regarded as a relational table, we can use existing relational DBMS algorithms to compute the result T in *TCompute(S, Q)*.

Although MSearchAll returns a set of substitutions (with no duplicate), VRetrieve may produce duplicate tuples from the set of substitutions. Thus, S is defined as a bag and assigned with a bag of tuples which may contain duplicates.

Example 4.3.8 *Consider the* RDF *graph I of Figure 4.12 and the query Q in Example 4.3.2.* CAA *first builds the graph pattern shown in Figure 4.13. X is set to (?highprice). Y is set to $\emptyset$. MSearchAll(GP, Q, I) returns two substitutions as described in Example 4.3.7. The bag S contains two tuples* $\{$*(*“800000”^^*ns*1*:USD), (*“500000”^^*ns*1*:USD)*$\}$ *returned by VRetrieve(θ, GP, $X \cup Y$) for those two substitutions. Thus, T contains one tuple* $\{$*(*“800000”^^*ns*1*:USD)*$\}$ *returned by TCompute(S, Q) which is the maximum of the two values.*

4.4 Aggregation Maintenance Algorithms

Before presenting algorithms to maintain aggregate views, I first describe properties of distributive and non-distributive aggregates, which affects the design of my algorithms.

An aggregate function f is **distributive** w.r.t a source update operation if and only if after such an operation, the updated value of the function can be computed based on its old value and the value(s) of the source update without reference to the source. More formally, f is distributive w.r.t. an update operation U if and only if there exists a function g such that $f(I') = g(f(I), v)$ where $f(I)$ is the aggregate value, I' is the updated instance after the update operation $U(I, v)$, and v is the value(s) used in the update (e.g., the new value to add, the old value to remove, etc).

Examples of distributive aggregate functions include count, sum, average w.r.t. insertion, deletion and update. For average, I will need an additional attribute *size* which stores the size of S (in line 4 of CAA) in order to compute the correct updated value (or, I can use sum, count to calculate it). max and min are distributive wr.t. insertion, but not deletion and update (which also involves deletion). Auxiliary data computed from the source (such as S) can help to maintain non-distributive aggregate functions to avoid the need to refer to the source. Note that I do allow more than one aggregate function to be used in a query. For example, I allow "SELECT max(?highprice), min(?lowprice)". However, by examining the view (query) specification, different aggregate functions will be processed differently and different auxiliary data may be required.

4.4.1 Insertion

```
algorithm $AMI(I, Q, A(Q, I), T(Q, I), t)$
/* Input:RDF graph $I$, query $Q$, auxiliary data $A(Q, I)$, query result $T(Q, I)$, inserted triple $t$ */
/* Output: table $T(Q, I \cup t)$, auxiliary data $A(Q, I \cup t)$ */
1)  $GP \leftarrow$ BuildGP($Q$);
2)  $X \leftarrow aggregate\ variables\ of\ Q$;
3)  $Y \leftarrow GROUP\ BY\ variables\ of\ Q$;
4)  if TMatch($GP$, $t$) == TRUE, then
5)      $\Delta S \leftarrow$ [VRetrieve($\theta, GP, X \cup Y$) | $\theta \in$ MSearch($GP, Q, t, I \cup t$)];
6)      return $(T(Q, I \cup t), A(Q, I \cup t)) \leftarrow TMaintain_I(T(Q, I), \Delta S, A(Q, I), Q)$;
7)  else, return $(T(Q, I \cup t), A(Q, I \cup t)) \leftarrow (T(Q, I), A(Q, I))$ ;
```

Figure 4.15: Aggregate Maintenance Algorithm for Insertion AMI

Figure 4.15 presents the AMI algorithm (short for **Aggregate Maintenance Algorithm for Insertion**) to find new objects to include in the answer when an insertion occurs. My AMI algorithm uses the following new subroutine.

- $T(Q, I)$ is the original view storing the answer of Q on I. ΔS contains the portion of S caused by t. A is the auxiliary data that depends on the original view and the specification of Q which will be covered by later sections. $TMaintain_I(T(Q, I), \Delta S, A(Q, I), Q)$ takes the above, computes the updated view $T(Q, I \cup t)$ and updates the auxiliary data.

Handling GROUP BY: When a GROUP BY clause exists, we require one more step in addition to the above procedure. By examining the GROUP BY clause and ΔS, we know which group each tuple in ΔS belongs to. Thus $TMaintain_I$ should only maintain each affected group by using the affecting tuples. It will delete any empty groups and create new groups if necessary. Each group will also have its auxiliary data (if any). The cases for deletion, modification, etc, in later sections are handled similarly.

TMaintain$_I$: Below shows how $TMaintain_I$ works for some common aggregate functions in this insertion case.

Handling sum, count, min, max We require no auxiliary data. Suppose we are considering an aggregate function $f(x)$ on an attribute x and F is the original aggregate value. The updated aggregate value F' can be computed as:

- $F' = F + \Sigma_{v \in \pi_x(\Delta S)} v$ if $f = \mathsf{sum}$
- $F' = F + |\Delta S|$ if $f = \mathsf{count}$
- $F' = min([F] \cup \pi_x(\Delta S))$ if $f = \mathsf{min}$
- $F' = max([F] \cup \pi_x(\Delta S))$ if $f = \mathsf{max}$

where $\pi_x(\Delta S)$ projects a bag of values of attribute x from ΔS and $\cup$ above is a bag union operation. *If a GROUP BY clause exists, the projection should be done for each group of values separately instead of altogether.*

Note that min, max DO need auxiliary data for the deletion case. Thus, we must update the auxiliary during insertion. We must store $\pi_x(S)$ in $A(Q, I)$ where $\pi_x(S)$ contains a bag of values of attribute x used to produce aggregate value. When adding the values stated in $\pi_x(\Delta S)$, we should update $\pi_x(S)$ to $\pi_x(S + \Delta S) = \pi_x(S) \cup \pi_x(\Delta S)$ and store it in $A(Q, I \cup t)$.

Handling average

For average, the additional information we need is the *size* of S in the original view, which can be stored in $A(Q, I)$. We can get the updated aggregate value F' and new $size'$ as follows:

$$size' = size + |\Delta S|$$
$$F' = \frac{F \cdot size + \Sigma_{v \in \pi_x(\Delta S)} v}{size'}$$

Example 4.4.1 *Suppose we want to know, for each artist, the smallest estimate for an artifact (with the title), and the largest estimate for an artifact (with title). This can be answered by the following query.*

```
SELECT ?lname, min(?lowprice), sum(?highprice)
WHERE (?artist, <ns1:lname>, ?lname),
(?artist, <ns1:creates>, ?artifact),
(?artifact, <ns1:title>, ?title),
(?artifact, <ns1:estimated>, ?price),
(?price, <ns1:low>, ?lowprice),
(?price, <ns1:high>, ?highprice),
USING ns1 FOR <http://www.auctionschema.com/schema1#>
GROUP BY ?lname
```

The result is [(*"Rose"*, "600000"^^*ns1:USD*, "800000"^^*ns1:USD*), (*"Tabacchi"*, "10000"^^*ns1:USD*, "15000"^^*ns1:USD*)]. *Suppose we insert the following triple to the source:* $t = (\&r3, <$*ns1:title*$>$, *"In the High Canadian Rockies"*). AMI *will first compute* $\Delta S =$ [(*"Rose"*, "300000"^^*ns1:USD*, "500000"^^*ns1:USD*)]. $TMaintain_I$ *finds that the only tuple in* ΔS *is in the group of "Rose", which is the only affected group.*

For $min(?lowprice)$, *we can update it by:* $min(600000, 300000) = 300000$. *We also need to update the auxiliary data for "Rose" group which is* $\pi_{?lowprice}(S)$ = ["600000"^^*ns1:USD*] *originally. Now it can be updated as:* $\pi_{?lowprice}(S \cup t)$ = ["600000"^^*ns1:USD*, "300000"^^*ns1:USD*].

For $sum(?highprice)$, *we can update it by:* $800000 + 500000 = 1300000$.

The resulting updated answer is: [(*"Rose"*, "300000"^^*ns1:USD*, "1300000"^^*ns1:USD*), (*"Tabacchi"*, "10000"^^*ns1:USD*, "15000"^^*ns1:USD*)].

Complexity Analysis of AMI.

For $TMaintain_I$, the worst case time complexity is $O(|S|)$ where $|S|$ is the number of records in S (the worst case refers to the update of auxiliary data, e.g. for min, max, without indexing).

Thus, MSearch dominates the time complexity of AMI algorithm. The worst case time complexity of AMI is $O(|GP|b^{D(GP)})$. *The similar analysis can be applied to other maintenance algorithms in later sections.*

4.4.2 Deletion

Deletion is a harder problem for min, max because we cannot compute the new aggregate value based on the old value and the update only. They are non-distributive w.r.t. deletion.

```
algorithm $AMD(I, Q, A(Q, I), T(Q, I), t)$
/* Input: RDF graph $I$, query $Q$, auxiliary data $A(Q, I)$, query result $T(Q, I)$, deleted triple $t$ */
/* Output: table $T(Q, I - t)$, auxiliary data $A(Q, I - t)$ */
1)    $GP \leftarrow$ BuildGP($Q$);
2)    $X \leftarrow aggregate\ variables\ of\ Q$;
3)    $Y \leftarrow GROUP\ BY\ variables\ of\ Q$;
4)    if TMatch($GP$, $t$) == TRUE, then
5)        $\Delta S \leftarrow$ [VRetrieve($\theta, GP, X \cup Y$) | $\theta \in$ MSearch($GP, Q, t, I$)];
6)        return $(T(Q, I - t), A(Q, I - t)) \leftarrow TMaintain_D(T(Q, I), \Delta S, A(Q, I), Q)$;
7)    else, return $(T(Q, I - t), A(Q, I - t)) \leftarrow (T(Q, I), A(Q, I))$ ;
```

Figure 4.16: Aggregate Maintenance Algorithm for Deletion AMD

Figure 4.16 presents the AMD algorithm (short for **Aggregate Maintenance Algorithm for Deletion**) to maintain views when a deletion occurs.

The following shows how $TMaintain_D$ works for different aggregate functions.

Handling sum, count We do not require any auxiliary data. Suppose we are considering an aggregate function $f(x)$ on an attribute x and F is the original aggregate value. The updated aggregate value F' can be computed as:

- $F' = F - \Sigma_{v \in \pi_x(\Delta S)} v$ if $f =$ sum

- $F' = F - |\Delta S|$ if $f = \mathsf{count}$

where $\pi_x(\Delta S)$ projects a bag of values of attribute x from ΔS.

Handling average For average, the additional information we need is the *size* of S in the original view, which can be stored in $A(Q, I)$. We can get the updated aggregated value F' and new $size'$ by the following:

$$size' = size - |\Delta S|$$
$$F' = \frac{F \cdot size - \Sigma_{v \in \pi_x(\Delta S)} v}{size'}$$

Note: if $size' = 0$, we should return F' as undefined.

Handling min, max min, max are not distributive w.r.t. deletion. We need to store $\pi_x(S)$ in $A(Q, I)$ where $\pi_x(S)$ contains a bag of values of attribute x used to produce aggregate value. When removing the values stated in $\pi_x(\Delta S)$, we should update $\pi_x(S)$ to become $\pi_x(S - \Delta S) = \pi_x(S) - \pi_x(\Delta S)$ and store it in $A(Q, I - t)$. Note that ΔS is a sub-bag of S, so the above expression is always true.

The updated aggregate value F' will be computed as:

- $F' = min(\pi_x(S - \Delta S))$ if $f = \mathsf{min}$
- $F' = max(\pi_x(S - \Delta S))$ if $f = \mathsf{max}$

Example 4.4.2 *Consider the aggregate query in Example 4.4.1 and the source data that is updated as described. Suppose we delete the triple* $t = (\&r2, <ns1{:}title>, \text{“The Model”})$. AMD *will first compute* $\Delta S = [(\text{“Rose”}, \text{“600000”}\text{^^}ns1{:}USD, \text{“800000”}\text{^^}ns1{:}USD)]$. $TMaintain_I$ *finds that the only tuple in* ΔS *is in the group of “Rose”, which is the only affected group.*

For min(?lowprice), we need to update and use the auxiliary data for “Rose” group which is $\pi_{?lowprice}(S) = [\text{“600000”}\text{^^}ns1{:}USD, \text{“300000”}\text{^^}ns1{:}USD]$ *originally. Now it can be updated as:* $\pi_{?lowprice}(S - \Delta S) = [\text{“300000”}\text{^^}ns1{:}USD]$. *Then, we can update the aggregate value as:* $min(300000) = 300000$.

For sum(?highprice), we can update it by: $1300000 - 800000 = 500000$.

The resulting updated answer is: $[(\text{“Rose”}, \text{“300000”}\text{^^}ns1{:}USD, \text{“500000”}\text{^^}ns1{:}USD), (\text{“Tabacchi”}, \text{“10000”}\text{^^}ns1{:}USD, \text{“15000”}\text{^^}ns1{:}USD)]$.

4.4.3 Triple Modification

An atomic modification update to a set of RDF statements can be classified as the modification of (1) a triple's resource (i.e., the starting point of an edge in the RDF graph changes); (2) a triple's property (the edge label changes); (3) a triple's value which is a resource (i.e., the ending point of an edge changes); (4) a triple's value which is a literal (i.e., the value stored at the ending point of an edge changes); (5) a resource itself (the change of rdf:about) from R to R' which will cause all edges connecting *to or from* R to change to connect *to or from* R'.

I will consider cases (1) to (4) in this section and (5) in the next section. For cases (1) to (4), we can simply process the modification as a deletion of the old triple t_d and an insertion of a new triple t_i (with the updated subject, predicate or object). When we modify a triple, some new subgraphs of I may be matched (producing ΔS_i, portion of new S to be inserted), but some subgraphs that matched before the update may no longer match (producing ΔS_d, portion of old S to be removed). However, if there are any common tuples between ΔS_i and ΔS_d, (portions of S caused by insertion and deletion), we can simply ignore them as the effect of their insertion will balance out the effect of their deletion.

Figure 4.17 presents the AMT algorithm (short for **Aggregate Maintenance Algorithm for Triple Modification**) to maintain a materialized view when a modification occurs in a triple. We now show how $TMaintain_T$ works for different aggregate functions.

Handling sum, count We do not require any auxiliary data. Suppose we are considering an aggregate function $f(x)$ on an attribute x and F is the original aggregate value. For $f \in \{\mathsf{sum}, \mathsf{count}\}$, the updated aggregate value F' can be computed as:

- $F' = F + \Sigma_{v \in \pi_x(\Delta S_i)} v - \Sigma_{v \in \pi_x(\Delta S_d)} v$ if $f = \mathsf{sum}$
- $F' = F + |\Delta S_i| - |\Delta S_d|$ if $f = \mathsf{count}$

where $\pi_x(\Delta S)$ projects a bag of values of attribute x from ΔS.

Handling average For average, the additional information we need to know is the *size* of S in the original view, which can be stored in $A(Q, I)$. We can get the updated aggregated value F' and new $size'$ by the following:

```
**algorithm** $AMT(I, Q, A(Q, I), T(Q, I), t_i, t_d)$
/* Input: RDF graph $I$, query $Q$, auxiliary data $A(Q, I)$,
        query result $T(Q, I)$, inserted triple $t_i$, deleted triple $t_d$ */
/* Output: table $T(Q, I')$, auxiliary data $A(Q, I')(I' = (I - t_d) \cup t_i)$ */
1)  $GP \leftarrow$ BuildGP($Q$);
2)  $X \leftarrow aggregate\ variables\ of\ Q$;
3)  $Y \leftarrow GROUP\ BY\ variables\ of\ Q$;
4)  $\Delta S_i \leftarrow []$; $\Delta S_d \leftarrow []$;
5)  if TMatch($GP$, $t_d$) == TRUE, then
6)      $\Delta S_d \leftarrow$ [VRetrieve($\theta, GP, X \cup Y$) | $\theta \in$ MSearch($GP, Q, t_d, I$)];
7)  if TMatch($GP$, $t_i$) == TRUE, then
8)      $\Delta S_i \leftarrow$ [VRetrieve($\theta, GP, X \cup Y$) | $\theta \in$ MSearch($GP, Q, t_i, I \cup t_i$)];
9)  $\Delta S_\cap \leftarrow \Delta S_i \cap \Delta S_d$;
10) $\Delta S_i \leftarrow \Delta S_i - S_\cap$; $\Delta S_d \leftarrow \Delta S_d - S_\cap$;
11) if $|\Delta S_i| + |\Delta S_d| == 0$, then return $(T(Q, I'), A(Q, I')) \leftarrow (T(Q, I), A(Q, I))$ ;
12) else, return $(T(Q, I'), A(Q, I')) \leftarrow TMaintain_T(T(Q, I), \Delta S_i, \Delta S_d, A(Q, I), Q)$;
```

Figure 4.17: Aggregate Maintenance Algorithm for Triple Modification AMT

$size' = size + |\Delta S_i| - |\Delta S_d|$

$F' = \dfrac{F \cdot size + \Sigma_{v \in \pi_x(\Delta S_i)} v - \Sigma_{v \in \pi_x(\Delta S_d)} v}{size'}$

Note: if $size' = 0$, we should return F' as undefined.

Handling min, max min, max are not distributive w.r.t. deletion. We need to store $\pi_x(S)$ in $A(Q, I)$ where $\pi_x(S)$ contains a bag of values of attribute x used to produce aggregate value. We use S' to denote $(S - \Delta S_d) \cup \Delta S_i$. We should update $\pi_x(S)$ to become $\pi_x(S') = (\pi_x(S) - \pi_x(\Delta S_d) \cup \pi_x(\Delta S_i))$ and store it in $A(Q, I')$.

The updated aggregate value F' will be computed as:

- $F' = min(\pi_x(S'))$ if $f =$ min
- $F' = max(\pi_x(S'))$ if $f =$ max

Example 4.4.3 *Consider the aggregate query in Example 4.4.1 and the source data that is updated as described. Suppose we modify the following triple from the source:* $t = (\&r5, <$*ns1:low*$>$, "300000"^^*ns1:USD*) *to* $t = (\&r5, <$*ns1:low*$>$, "200000"^^*ns1:USD*)

AMT *will first compute* ΔS_d = [(*"Rose"*, "300000"^^*ns1:USD*, "500000"^^*ns1:USD*)]. ΔS_i = [(*"Rose"*, "200000"^^*ns1:USD*, "500000"^^*ns1:USD*)].

$TMaintain_T$ *finds that the only tuple in* ΔS_i *and* ΔS_d *is in the group of "Rose", which is the only affected group.*

For $min(?lowprice)$, *we need to update and use the auxiliary data for "Rose" group which is* $\pi_{?lowprice}(S)$ = ["600000"^^*ns1:USD*, "300000"^^*ns1:USD*] *originally. It can now be updated as:* $\pi_{?lowprice}((S - \Delta S_d) \cup \Delta S_i)$ = ["600000"^^*ns1:USD*, "200000"^^*ns1:USD*]. *We may then update the aggregate value to:* $min(600000, 200000) = 200000$.

For $sum(?highprice)$, *we can update it by:* $1300000 - 500000 + 500000 = 1300000$.

The resulting updated answer is: [(*"Rose"*, "200000"^^*ns1:USD*, "1300000"^^*ns1:USD*), (*"Tabacchi"*, "10000"^^*ns1:USD*, "15000"^^*ns1:USD*)].

4.4.4 Resource Modification

In case (5) where a resource is changed from R_d to R_i, we need to consider the new values caused by the insertion of a new resource R_i and the new edges connected to it AND we need to remove old values caused by the deletion of resource R_d and the edges connected to it.

Suppose E_d is the set of edges connected to R_d. One approach is to call AMI and AMD $|E_d|$ times (for inserting and deleting the edges one by one). Alternatively we can instead consider the deletion and insertion of all those edges together. In this second approach, we consider the substitutions of the deleted/inserted resource to the resource/value of a triple in GP instead of the substitutions of the deleted/inserted edges to a triple in GP.

Figure 4.18 presents the AMR algorithm (short for **Aggregate Maintenance Algorithm for Resource Modification**) to make necessary updates to a materialized view when a modification occurs to a resource. We denote I' as the updated RDF graph after replacing R_d by R_i.

Our AMR algorithm uses the following new subroutines.

- *TMatchR(GP, R)* returns "true" for a resource R and a graph pattern GP if R unifies with a GP triple's subject or object.
- Given GP, Q, resource R and the updated RDF graph $(I \cup t)$, we define *MSearchR(GP, Q, R, I)* to find all substitutions[9] θ that match GP with some subgraph of I that contains R. MSearchR is a variant of MSearch where R rather than t gives a fixed starting point.

AMR uses the same function $TMaintain_T$ of AMT.

Example 4.4.4 *We modify the aggregate query in Example 4.4.1 to the following:*

```
SELECT ?artist, min(?lowprice), sum(?highprice)
WHERE (?artist, <ns1:creates>, ?artifact),
(?artifact, <ns1:title>, ?title),
(?artifact, <ns1:estimated>, ?price),
```

[9]Substitutions that are subsumed by another substitution need not be returned.

```
**algorithm** $AMR(I, Q, A(Q, I), T(Q, I), R_i, R_d)$
/* Input: RDF graph $I$, query $Q$, auxiliary data $A(Q, I)$,
        query result $T(Q, I)$, inserted resource $R_i$, deleted resource $R_d$ */
/* Output: table $T(Q, I')$, auxiliary data $A(Q, I')(I' = (I - R_d) \cup R_i)$ */
1)  $GP \leftarrow$ BuildGP($Q$);
2)  $X \leftarrow aggregate\ variables\ of\ Q$;
3)  $Y \leftarrow GROUP\ BY\ variables\ of\ Q$;
4)  $\Delta S_i \leftarrow []$; $\Delta S_d \leftarrow []$;
5)  if TMatchR($GP$, $R_d$) == TRUE, then
6)      $\Delta S_d \leftarrow$ [VRetrieve($\theta, GP, X \cup Y$) | $\theta \in$ MSearchR($GP, Q, R_d, I$)];
7)  if TMatchR($GP$, $R_i$) == TRUE, then
8)      $\Delta S_i \leftarrow$ [VRetrieve($\theta, GP, X \cup Y$) | $\theta \in$ MSearchR($GP, Q, R_i, I'$)];
9)  $\Delta S_\cap \leftarrow \Delta S_i \cap \Delta S_d$;
10) $\Delta S_i \leftarrow \Delta S_i - S_\cap$; $\Delta S_d \leftarrow \Delta S_d - S_\cap$;
11) if $|\Delta S_i| + |\Delta S_d| == 0$, then
12)     return $(T(Q, I'), A(Q, I')) \leftarrow (T(Q, I), A(Q, I))$ ;
13) else, return $(T(Q, I'), A(Q, I')) \leftarrow TMaintain_T(T(Q, I), \Delta S_i, \Delta S_d, A(Q, I), Q)$;
```

Figure 4.18: Aggregate Maintenance Algorithm for Resource Modification AMR

```
(?price, <ns1:low>, ?lowprice),

(?price, <ns1:high>, ?highprice),

USING ns1 FOR <http://www.auctionschema.com/schema1#>

GROUP BY ?artist
```

The result is [(&$r1$, "600000"^^*ns1:USD,* "800000"^^*ns1:USD*), (&$r9$, "10000"^^*ns1:USD,* "15000"^^*ns1:USD*)].

Suppose we change the resource &$r1$ *to* &$r13$ *which refers to* *http://www.artist.net#g-rose.* AMR *will first compute* ΔS_d = [(&$r1$, "600000"^^*ns1:USD,* "800000"^^*ns1:USD*)]. AMR *then computes* ΔS_i = [(&$r13$, "600000"^^*ns1:USD,* "800000"^^*ns1:USD*)].

$TMaintain_T$ *finds that the* &$r1$ *group becomes empty after removing* ΔS_d*, but it creates a new* &$r13$ *group for* ΔS_i*.*

The resulting updated answer is [(&$r13$, "600000"^^*ns1:USD,* "800000"^^*ns1:USD*), (&$r9$, "10000"^^*ns1:USD,* "15000"^^*ns1:USD*)].

4.5 Relational Approach

Instead of using a graph structure, we can store and query an RDF graph using a standard relational database system. Florescu and Kossmann[27] have suggested a few alternative approaches to store XML data in a relational database. An RDF-instance I is a set of triples that can directly be stored in a relational table having schema (Resource,Prop,Value). An insertion of an RDF triple corresponds to an insertion of a tuple into this relational table. Any standard view maintenance algorithm can be used[8, 13, 38, 39] – we use IMAr, DMAr, TMAr and RMAr to denote the use of standard view maintenance algorithms (*DRed [39]* in my implementation) to maintain non-aggregate views when insertions, deletions, tuple modifications and resource modifications are made. Similarly, we use RAMI, RAMD, RAMT and RAMR to denote the use of *Counting* algorithm [39] for aggregate views.

4.6 Implementation and Scalability Experiments

4.6.1 Maintaining Non-aggregate Views

I implemented a prototype system in Java which consists of 5099 lines of code. The system consists of three main components: XMA, XMAr and Recomp where X is either I, D, T or R. XMA provides functionalities of IMA, DMA, TMA and RMA. It constructs its own graph model from the data stored in Hewlett Packard's Jena 2.1 system. XMAr provides functionalities of IMAr, DMAr, TMAr and RMAr. The Jena 2.1 system uses a statement table to store RDF statement just like what I described the relational approach in the previous section.[81] Jena also uses various optimization techniques including indexing to evaluate RDQL queries efficiently. Therefore I use Jena to evaluate delta rules produced by XMAr specified by the well known view maintenance algorithm (*DRed*) in [39]. Recomp uses Jena API to compute modified materialized views from scratch.

In all my scalability experiments, I used the data from Open Directory RDF Dump[21]. Each query we used has the following form (the graph pattern contains at least 5 triples):

```
CREATE VIEW dmoz AS
SELECT ?topic,?atitle,?btitle
WHERE (?topic,<ns1:catid>,?id),
(?topic,<ns1:link>,?alink),
(?alink,<ns2:Title>,?atitle),
(?topic,<ns1:link>,?blink),
(?blink,<ns2:Title>,?btitle)
AND ?id < 472029, ?alink NE ?blink
USING ns1 FOR <http://dmoz.org/rdf#>,
ns2 FOR <http://purl.org/dc/elements/1.0/>
```

I randomly inserted/deleted/modified some single triple/resource. For each combination of algorithm and database size, I repeated the experiment to have at least 5 non-empty results (ΔM in insertions and deletions) or intermediate results ($\Delta M_i, \Delta M_d$ in modifications). I then took the average running time and original view size. The experiments were run on a PC with 1.4GHz CPU, 524MB memory and Windows 2000 Professional platform.

The times taken in all experiments only includes the time to (i) build graph patterns (ii) do updates to the database and (iii) compute ΔM. However, it excludes the time to load RDF data into the Jena ***model (for all algorithms) and construct my graph model (for*** XMA***) because this can be done once.***

When comparing my algorithms with algorithms that operate on the relational version of the RDF *data, I compared my algorithms with the well known view maintenance algorithms (*DRed*) of Gupta et. al. [39].*

Insertion

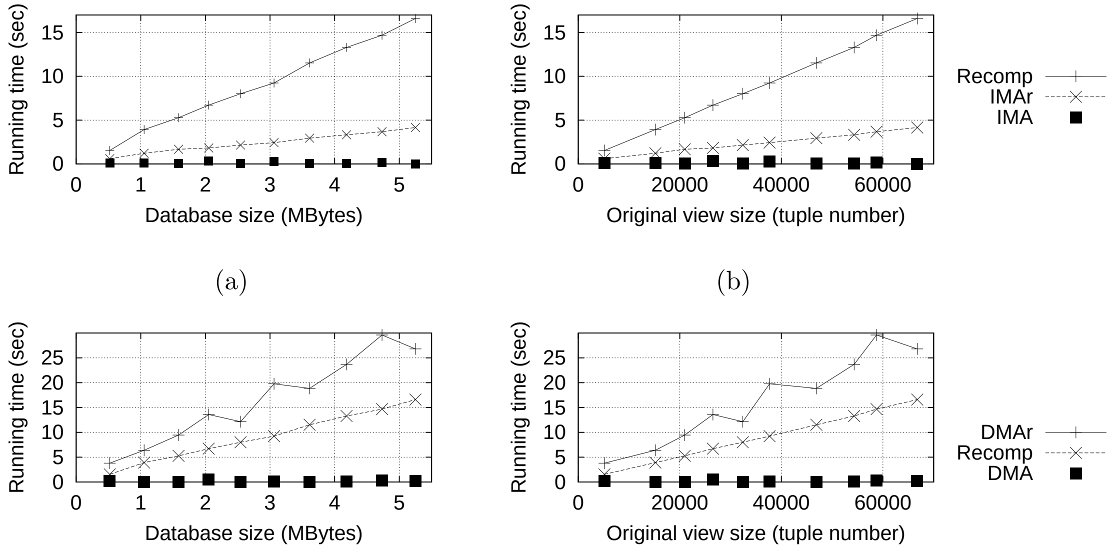


Figure 4.19: Running time of IMA, IMAr and Recomp against (a) database size, (b) original view size. Running time of DMA, DMAr and Recomp against (c) database size, (d) original view size.

Figure 4.19 (a) and (b) show the running time of IMA, the relational approach IMAr, and Recomp against (a) the database size and (b) the original view size. IMA's running time is independent of the database size and original view size because it uses a local search for the substitutions of graph pattern. In contrast, the relational approach IMAr's running time increases linearly with the database size and original view size.

It is important to note that IMA *is highly scalable - it almost hugs the x-axis in the graph shown and can process materialized views containing over 65,000 tuples in under 0.4 second, at least 12 times faster than the relational approach.*

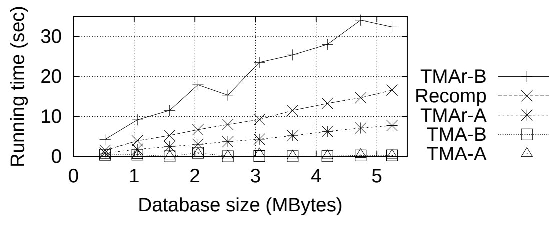
Deletion

Figure 4.19 (c) and (d) show the running time of DMA, the relational approach DMAr and Recomp against (c) the database size and (d) the original view size. DMA's running time is again independent of the database size and original view size with the same reason in IMA. In contrast, DMAr's running time is much longer than DMA's and even exceeds that of Recomp. It may be because of the overheads of sending a lot of delta rules. Compared with Recomp, the large number of queries outweighs the shorter processing time of each query.

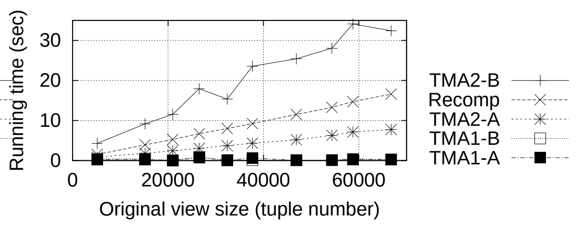
As in the case of IMA*, the reader will note that* DMA *also performs very well, handling views of over 65K tuples in well under 0.6 seconds, 49 times faster than the relational approach.*

Modification

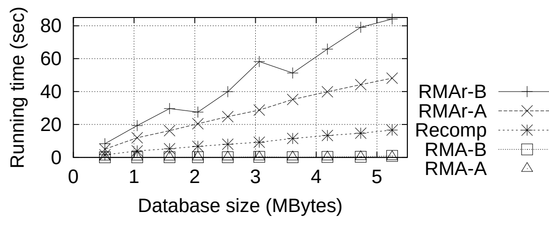
Figure 4.20 shows the running time of TMA, the relational approach TMAr and Recomp as we vary (a) the database size and (b) the original view size. Figure 4.20 (c) and (d) shows the counterparts for RMA. TMA and RMA both compute triples to be inserted (Ins) and triples to be *potentially* deleted ($Pdel$). In my experiments I tested two extreme cases - (i) those where $Pdel \subseteq Ins$ and those where (ii) $Pdel \cap Ins = \emptyset$. Of course, most view maintenance operations fall somewhere between these two extremes. I use TMA-A and TMA-B to denote the application of TMA in cases (i) and (ii) above, respectively. I use the notation TMAr-A, TMAr-B, RMA-A,RMA-B, RMAr-A,RMAr-B in a similar way. The running time of TMA-A, TMA-B, RMA-A and RMA-B are again under 0.92 seconds and independent of the database size (and original view size, whose figures are not shown due to the limitation of space) for the same reason as for IMA. Both running times are very close. In contrast, we can see a big difference between TMAr-A and TMAr-B. Without the need to submit a large number of delta rules, TMAr-A runs much faster than TMAr-B. TMAr-B's running time in



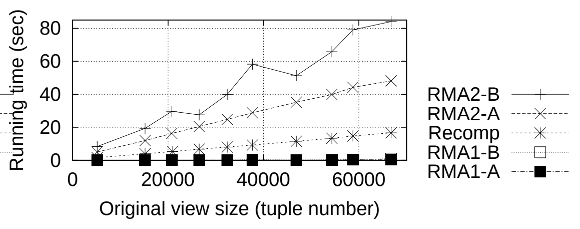
(a)



(b)



(c)



(d)

Figure 4.20: Running time of TMA, TMAr and Recomp against (a) database size, (b) original view size. Running time of RMA, RMAr and Recomp against (c) database size, (d) original view size. Curves of TMA-A/B and RMA-A/B are hard to distinguish among each other because their values are very close to zero.

some cases can even exceed that of Recomp because of this. The above also applies to RMAr. *As in the case of* IMA *and* DMA*,* TMA *and* RMA *also perform very well, handling views of over 65K tuples under 0.92 second, 19 to 177 times faster than the relational approach.*

4.6.2 Maintaining Aggregate Views

I have implemented a prototype system in Java (5701 lines of code) that contains AMI, AMD, AMT and AMR algorithms (collectively called AMX algorithms) as well as standard view maintenance algorithm (the *counting* algorithm in [39]) applied to the relational version of an RDF instance - these are called RAMI, RAMD, RAMT and RAMR (collectively called RAMX algorithms) respectively. AMX algorithms construct their own graph model from the data stored in Hewlett Packard's Jena 2.1 system so that I can navigate through the graph directly and reduce the overhead of Jena API calls. Jena 2.1 uses a statement table to store RDF statement as described the relational approach in earlier sections[81]. In addition, Jena uses various optimization techniques including

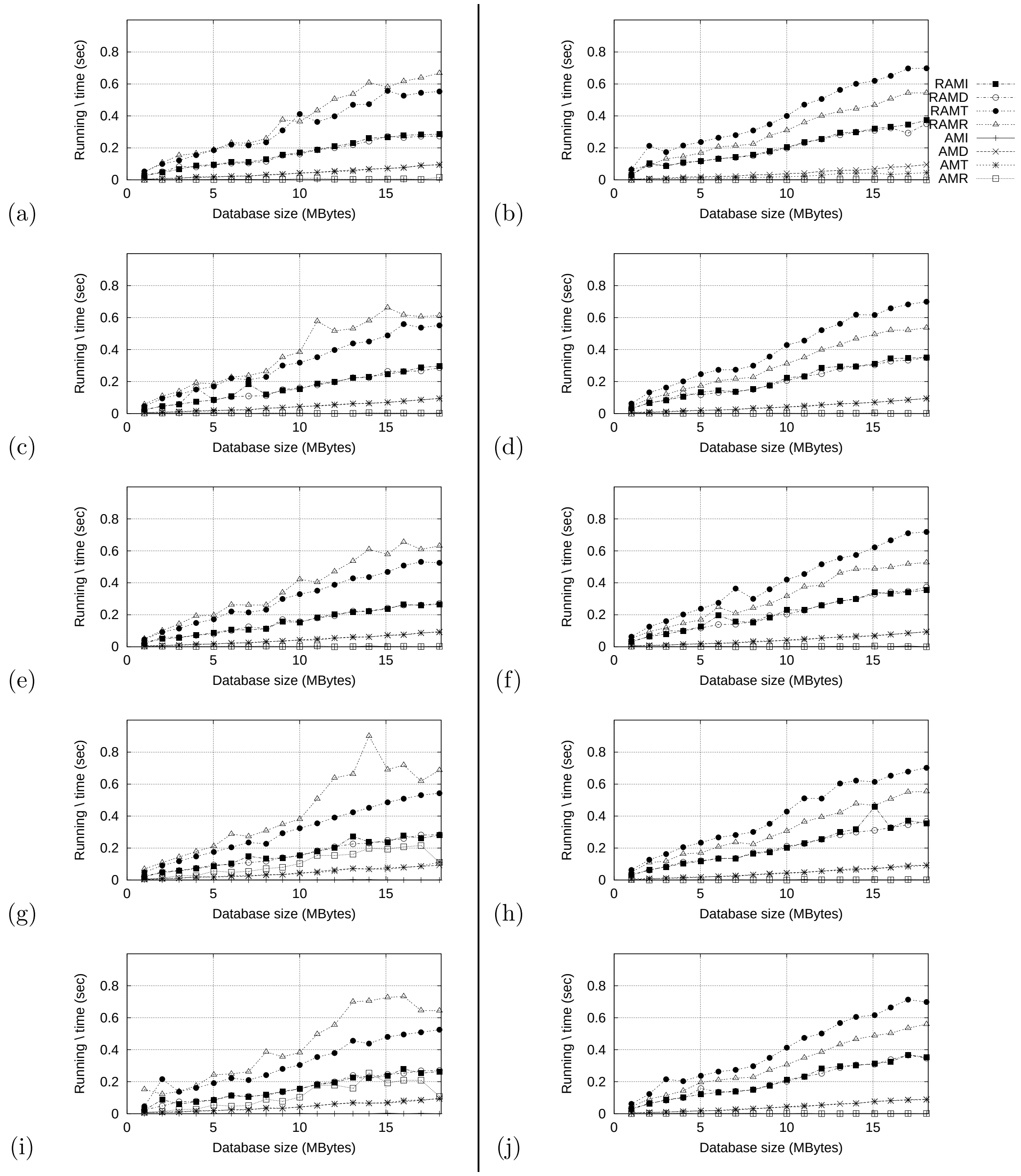


Figure 4.21: Running time of AMI, AMD, AMT and AMR, and their counterparts in the relational apporach (RAMI, RAMD, RAMT and RAMR) for queries with aggregate functions (a,b)*count*, (c,d)*sum*, (e,f)*avg*, (g,h)*max*, (i,j)*min*. *GROUP BY* clauses are used in the experiments in the left column but not in the right column. Curves of AMI, AMD, AMT and AMR are hard to distinguish among each other because their values are very close to zero.

indexing to evaluate RDQL queries efficiently. Therefore I use Jena to evaluate delta rules produced by RAMX.

In all my scalability experiments, I used the restaurant data from Chef Moz dining guide (accessible at Chef Moz Project RDF Dump, http://chefmoz.org). Each query has the following form with its graph pattern containing at least 7 triples:

```
SELECT ?cu,agg(?x) WHERE (?r,<ns1:AmbianceRating>,?a),
(?r,<ns1:FoodRating>,?b),(?r,<ns1:ServiceRating>,?c),
(?r,<ns1:OverallRating>,?d),(?r,<ns1:Cuisine>, ?cu),
(?r,<ns1:Review>,?rev),(?rev, <ns1:FoodRating>, ?e)
USING ns1 For <http://chefmoz.org/rdf/elements/1.0/#>
GROUP BY ?cu
```

where *agg* is one of the following five aggregate functions: *count, sum, avg, max, min*; $?x$ is one of the following five variables: $?a, ?b, ?c, ?d, ?e$. Although my system can handle multiple different aggregates at the same time, I use the above query in order to examine the performance of my algorithms on each aggregates.

I randomly inserted/deleted/modified some single triple/resource. For each combination of algorithm, aggregates, database size, with or without *GROUP BY*, I repeated the experiment 15 times with different aggregated variables and different updates. I then took the average running time. The experiments were run on a PC with 1.4GHz CPU, 524MB memory and Windows 2000 Professional platform.

Each row of graphs in Figure 4.21 shows the results of each aggregate function. *GROUP BY* clauses are used in the experiments in the left column but not the right column. In all experiments, my algorithms run much faster than their counterparts in the relational approach in three to seven times. The curves of AMI, AMD, AMT and AMR almost hug the x-axis in the graphs shown and can process 18 MB of source data in well under a second (in fact, under 0.22 second). They are in general at least three times to 100 times faster then the relational counterparts because they use a local search strategy for the substitutions of the graph pattern starting from the data to be updated in order to find what the new view should add into or remove from the original view.

4.7 RDF Containers, Collections and Reification

RDF provides basic vocabulary to describe three kinds of containers: *bag, seq (sequence) and alt (alternative).*

In the formal RDF semantics[71], there are no special semantic conditions concerning a container other than its general structure and general RDF semantic conditions like other RDF statements. For example, it is only an intended meaning that a *bag* has a set of unordered and perhaps duplicate members. Such intended meanings have no built-in understanding in RDF and only depend on the implementation of applications.

There are two approaches to define queries in RDF databases that have containers. One is to explicitly handle the intended meaning of containers. Queries should be written to check explicitly, using WHERE and AND clauses, whether a resource is a container and, if yes, to access its members. For example, RDQL syntax can be extended easily to handle containers by (1) using (?a, <rdf:type>, "rdf:bag"), (?a, <rdf:li>, ?b) in the WHERE clause or (2) using both (?a, <rdf:type>, "rdf:bag"), (?a, ?p, ?b) in the WHERE clause and ?p="<rdf:_*>" in the AND clause. Here I use the abbreviations <rdf:type>, rdf:bag and <rdf:li> rather than their full names. <rdf:li> should be recognized by RDQL evaluation engine as a way to represent ?b as a member of ?a rather than a real property name directly because <rdf:li> is only provided by RDF/XML as a convenience element to aviod having to explicitly number each membership property as <rdf:_1>, <rdf:_2>, etc. The * in <rdf:_*> is treated as a wild card. The other approach is, with queries written as usual, to handle containers implicitly by some mechanisms. For example, when the engine reaches a resource which is a container, it can automatically substitute it by each of its members. The advantage of this method is the high *portability* of queries – containers are handled automatically without the need of checking expressed in the queries as above. The disadvantage is the possible ambiguity when we want to access the properties of an object which may be a container – should we return the properties of the container or those of its members? Due to this semantic problem, I choose the first approach in this paper, which means that my algorithms in previous sections can also be applied to containers.

RDF collections[71] are more complicated than RDF containers because an RDF collection uses a list structure, which means that every element in a collection has a different path length (number of edges to reach) from the collection resource node. In order to simplify a query, a possible extension of RDQL can use a special property <rdf:li> to access the elements of a collection, similar to a container as described above. With this united form, users can write queries to access elements of a container or a collection without worrying about whether it is actually a container or a collection. If RDQL is extended in this way, the methods described in this paper handle containers and collections directly.

RDF reification[71] allows the use of RDF to describe properties of an RDF triple, such as its date of composition. We can handle a reification triple just like other ordinary triples. The only difference is that the type of its resource is rdf:Statement, but it does not affect our algorithms.

4.8 Summary

In this chapter, I proposed algorithms to maintain RDF views for various updates to RDF databases such as insertion, deletion and modification. I then extended RDQL (RDF query language) to support aggregations and proposed another set of algorithms to maintain such aggregate views. Experimental results show that my algorithms which are based on local search in graphs are more efficient than by simply adapting general relational view maintenance algorithms.

Chapter 5

Related Work

5.1 Probabilistic Semistructured Databases

There has been considerable work done on storing probabilistic information in relational databases [50, 19, 20, 24, 32, 33], object databases[23] and temporal databases[17], to date, there has been little work on supporting uncertainty in semistructured models.

More recently, Nierman and Jagadish developed a framework called ProTDB to extend the XML data model to include probabilities[61]. XML DTD's are extended to use a `Prob` attribute for each element. As a query language, they use a variation of their earlier work on the TAX algebra for XML[42] and use pattern trees. ProTDB proposed by Nierman and Jagadish[61] is similar in spirit to my model – however there are a few important differences. In ProTDB, independent conditional probabilities are assigned to each individual child of an object (i.e., independent of the other children of a node); PXML supports arbitrary distributions over sets of children. Furthermore, dependencies are required to be tree-structured in ProTDB, whereas PXML allows arbitrary acyclic dependency models. In the other words, their answers are correct under the assumption of conditional independence and under the condition that the underlying graph is tree-structured. Thus the PXML data model subsumes the ProTDB data model. In addition, I prove here that the semantics of PXML is probabilistically coherent. Another important difference is in the queries supported. There is no direct mapping among my algebra and their query language. For example, in their conjunctive query, given a query pattern tree, they return a set of subtrees (with some modified node probabilities) from the given instance, each with a global probability. There is no direct mapping between their conjunctive query and my ancestor projection because they find subtrees matching the pattern tree, while I use a path expression. Each of their subtrees is restricted to match the query pattern tree and has a fixed structure while my output is a probabilistic instance which implicitly includes many possible structures. In my PIXML model, I use an *interval probability* model rather than a point probability model. This is useful because almost all statistical evidence involves margins of error. So when a statistical estimate says that something is true with 95% probability with a ±2% margin of error, then this really corresponds to saying the event's probability lies in the interval $[0.93, 0.97]$. Likewise, using intervals is valuable when one does not know the relationship between different events. For example, if we know the probabilities of events e_1, e_2 and we want to know the probability of both of them holding, then we can, in general, only infer an interval for the conjunction of e_1, e_2 (cf.[9, 26]) *unless* we know something more about the dependencies or lack thereof between the events. Third, I provide two formal declarative semantics for probabilistic semistructured databases - no such model theory is proposed by [61]. I additionally prove that these model theories have a variety of interesting relationships. Finally, I prove that my PIXML query language is sound and complete w.r.t. the above model theory.

The work of Dekhtyar et al.[16] was the first to deal with probabilities and semistructured data. They pioneered the integration of probabilities and semistructured data by introducing a semistructured model to support storage and querying of probabilistic information in flexible forms such as a simple interval probability distribution, a joint interval probability distribution, or a simple or joint conditional interval probability distribution. Their model allows us to use an object (semistructured probabilistic object or SPO) to represent the probability table of one or more random variables, the extended context and the extended conditionals. Intuitively, contexts provide information about when a probability distribution is applicable. They then go ahead and developed an elegant algebra to query databases of such SPOs and a prototype implementation. Their work appears to be similar to PXML but in fact it is quite different. An SPO itself can be represented in a semistructured way, but its main body is just a flat table. It cannot show the semistructured relationship among variables. Only contexts (but not random variables) are represented in a semistructured form. Contexts are "regular relational attributes", i.e., the context provides already known information when the probability distribution is given on real "random variables". In contrast, my model is based on the widely used model OEM[64], which allows data to be represented in a truly semistructured manner. I modify the syntax and semantics of the model by introducing cardinality and object probability functions to demonstrate the uncertainty of the number and the identity of objects existing in possible worlds. Every possible world is a semistructured instance compatible with the probabilistic instance. The representation of a possible world (semistructured instance) is the same as the one widely accepted nowadays. However, the model of Dekhtyar et al. cannot do this. Their model also requires random variables to have distinct variable names (or edge labels) (in my model, they are the children connected to their parents with the same edge label). Consequently, their model cannot allow two or more variables with the same variable names (no matter their values are the same or different) in a single possible world. Their model also cannot capture the uncertainty of cardinality. On the other hand, my model can represent their table. For each random variable, define a set of children (with the possible variable values) connected to their parent with the same edge label (set as the variable name). The cardinality associates with the parent object with each label is set to $[1, 1]$ so that each random variable can have exactly one value in each possible world. The extended context and extended conditionals in SPO can be represented by two subtrees with corresponding edge labels and values connected to the parent object. In [35, 85, 86], they extended their work to handle interval probabilities. My query language described in Chapter 3 (my PIXML model) is a very simple logical one. I also provide a model theory (two in fact) and an operational semantics and show that my operational semantics is correct.

The above two pieces of work are closest to mine. In addition, there has been extensive work on probabilistic databases in general. Kiessling et al.'s DUCK system[36, 48] provides a logical, axiomatic theory for rule based uncertainty. Lakshmanan and Sadri[51] show how selected probabilistic strategies can be used to extend the previous probabilistic models. Lakshmanan and Shiri[52] have shown how deductive databases may be parameterized through the use of conjunction and disjunction strategies. Barbara et al.[5] develop a point probabilistic data model and propose probabilistic operators. Cavallo and Pittarelli[12]'s important probabilistic relational database model uses probabilistic projection and join operations, but the other relational algebra operations are not specified. Also, a relation in their model is analogous to a single tuple in the framework of Barbara et al.[5] Dey and Sarkar[19] propose an elegant 1NF approach to handling probabilistic databases. They support (i) having uncertainty about some objects but certain information about others, (ii) first normal form which is easy to understand and use, (iii) new operations like conditionalization. The ProbView system by Lakshmanan et al.[50] 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 removes the independence assumption of previous works. More recently, Dyreson and Snodgrass pioneered the use of probabilities over time in databases[22].

XPath[82] and XQuery[83] are languages that use path expressions (defined in XPath) to extract objects. SAL[6] and TAX[42] are two algebras for semistructured data. SAL binds objects to variables, manipulates the bindings and then removes bindings constructing a result. The reason that I cannot use XPath, Xquery and SAL directly is that the original parent-children relationships and probabilities associated with objects cannot be kept directly in the results since individual objects are selected during the process. However, my algebra uses the well-defined path expressions as a tool to locate the objects we are interested and manipulates the graph structure of semistructured data directly. On the other hand, TAX uses a pattern tree to extract subsets of nodes (called witness trees), one for each embedding of the pattern tree in an input tree (instance). Its algebraic operations are similar to mine in some aspects. The reason that I cannot use theirs directly is the fixed structure of the result, e.g., fixed number of children, which restricts the representation of the uncertainty in cardinality.

Though there has been a substantial amount of work on probabilistic databases, there has been almost no work on probabilistic aggregates in even probabilistic relational databases. Below are related work on probabilistic aggregates and on aggregates on semistructured databases.

Ross et al.[73, 74] extend probabilistic relational databases to handle aggregate queries. They define a semantics for such queries using linear programming and then develop an algorithm to compute aggregate queries. The key differences between this work and theirs is that I am building on top of XML rather than relational data sources. The structure of XML documents are far more complex to deal with than relational sources especially as the values to be aggregated are stored in a semistructured way and I require the output of an aggregate operator to also be an XML instance.

Aggregate operations defined in XQuery proposed by W3C can be found in the working draft[83]. Besides textual languages, a graphical language XML-GL[14] was also proposed to handle aggregates. A compact representation for exponentially large number of answers to a query on XML documents and the algorithm to compute that can be found in the work of Meuss and Schulz[60]. Some methods to collect summary statistics for selectivity estimation were introduced[29, 67, 68]. Related work on aggregates for information retrieval is described in XIRQL, a query language proposed by Fuhr and GroBjohann[31].

5.2 View Maintenance of Ontology Databases

There is very little related work on RDF views. Volz et al.[79] were the first to introduce a view mechanism for RDF data. Their views require that (i) the results contain class instances (i.e., a subject or object variable), or (ii) the result itself has the pattern of RDF statement (i.e., a triple containing subject, predicate and object). My algorithms apply to all possible RDQL queries. Magkanaraki et al.[55] proposed RVL, a view definition language that can also create virtual RDF schemas and restructure class and property hierarchies such that new resources, property values, classes and property types can be created. Wilkinson et al.[81] proposed a methodology to store and query persistent RDF graphs in Jena. They also introduced two tools to assist in designing application-specific RDF storage schema. None of these works addresses RDF aggregates or the problem of maintaining aggregate RDF views.

To my best knowledge, there is no work on RDF view maintenance to date. However, there has been a huge amount of work on maintaining views [37]. Relational DB view maintenance works include [8, 13, 38, 39]. Zhuge and Garcia-Molina[87] develop methods to maintain graph structured views. There are substantial variations between the RDF RDQL and their notion of a graph structured database (GSDB). In a GSDB (which must be rooted), views are defined using a simple language that (i) uses one path expression to locate a node n to be returned, and (ii) using some other path expressions to identify paths starting from node n that satisfy various conditions. Thus, a view is defined to contain a set of nodes, each of which satisfies some path expression and condition(s). In contrast, in my RDF framework, materialized views are not just sets of nodes – they can contain edge labels as well as nodes in any form. In addition, path expressions in GSDBs [87] starts from a given set of nodes (e.g. the root). In contrast, in an RDF database system and in RDQL, there is no root. RDQL allows variables in subjects (i.e., a path expression can start from arbitrary nodes in the graph). The incremental maintenance algorithm proposed in[87] assumes that the database is tree-structured - I do not make any such assumption. They also proposed a self-maintainability test[88] to determine whether access to the base data is required to incrementally maintain a view after a database update – I do not do this.

Abiteboul et al.[4] discuss techniques to incrementally maintain views for arbitrary graph-structured databases. They used the view specification language of [2]. Their views include not only objects but edges between objects. They defined a cost model and provided a detailed cost analysis with experimental results. However, they have two key assumptions which are inapplicable to my setting. First, their path expression still has an entry node identified by a special name label (may not be the root though). Second, in their view definition, only objects are variables while edges are not. However, in RDQL, even the start node of a path expression could be a variable. Furthermore, RDQL allows variables on properties (edge labels). They do not discuss either aggregate computation or aggregate view maintenance.

Volz et al.[78] described how to incrementally maintain materialized views of queries on a semantic web when changes occur. They represented a semantic web using Datalog rules and proposed how to solve the maintenance problem when the rules change, which is new to deductive databases.

Kang and Lim[45] described a framework for XML materialized view refreshing and addressed issues in its deferred incremental refresh using an object-relational DBMS storing the

XML documents and views. Kang et al.[46] then proposed algorithms which outperform recomputation shown by experimental results. EL-Sayed et al.[25] proposed an algebraic approach for incremental materialized XQuery view maintenance. A source update is transformed into a set of update primitives propagated through the XAT (their XML algebra) tree in a bottom-up approach. Yi et al.[84] worked on incremental maintenance of XML structural indexes.

In [75], Rundensteiner proposed a methodology called *MultiView* for supporting multiple view schemata in object-oriented databases by breaking view specification into independent tasks: class derivation, global schema integration, view class selection, and view schema generation. Kuno and Rundensteiner proposed a number of techniques in [49] for incremental maintenance of materialized views in MultiView, e.g., pruning unnecessary update propagations, providing registration services, and introducing notion of hierarchical registrations. They also present a cost model and report experimental results.

Quass [69] proposed a framework to maintain aggregate views including group by attributes and multiple aggregates. Gupta et al. [40] propose a framework of change-tables and a refresh operator to incrementally maintain views involving relational and aggregate operators. Palpanas et al. [63] present a general and efficient solution for both the distributive and the non-distributive aggregate functions. An important step in their work is selective recomputation. They also discuss a series of optimizations on the query plans for the maintenance. However, all of these works are based on relational data model. My work is based on the RDF graph model although I choose tabular representation for the query results. Furthermore, my approach works independently of the view definition languages, i.e., not limited to tabular representation. My experiments show a better performance for my algorithms compared with the relational counterparts.

Paparizos et al. [65] discussed how to specify grouping constructs in their tree algebra for XML (TAX) and how to rewrite nested queries in XQuery to queries with grouping in TAX. In addition, they described the implementation and performance benefits of grouping over the equivalent nested join queries. They briefly mention aggregation but do not address it in depth.

Tufte et al. [77] proposed a Merge operation and a flexible mechanism, called Merge Template, to create aggregates over XML stream data. This operation is designed to combine tree-based structures and compute aggregation over values. Their work handles value replacement (to result documents) and insertion, but cannot handle deletion and update. Furthermore, they do not consider incremental maintenance of aggregates.

Chapter 6

Conclusion

As XML is used more widely to represent textual sources, multimedia sources, biological and chemical applications, Nierman and Jagadish[61] have argued eloquently for the need to represent uncertainty and handle probabilistic reasoning in semistructured data of applications like information retrieval and protein chemistry applications. There are many other applications ranging from image surveillance applications to the management of predictive information.

In the first part of this dissertation, I have presented two new probabilistic semistructured data models, PXML and PIXML models and developed a formal theory for probabilistic semistructured data. Specifically, I have shown how graph models of semistructured data may be augmented to include probabilistic information, where point probabilities are used in PXML while interval probabilities in PIXML . In addition, I have provided two formal declarative semantics for such databases. The first is a global semantics that draws inspiration from classical probabilistic model theory as exemplified in the work of Fagin et al.[26] and applies it to the probabilistic XML model proposed here. I also propose a local semantics that can be manipulated much more efficiently to avoid the exponential blowup in handling the global semantics. I have proven that the two semantics are probabilistically coherent. I have presented an algebra for the PXML model. The algebra has some interesting differences from existing XML algebras. I have shown how queries can be answered efficiently in my PXML system. In the PIXML model, I have proposed a query language to query such sources and provided an operational semantics that is proven to be sound and complete. To my knowledge, the declarative semantics and the soundness and completeness results are the first of their kind.

To date, there has been no work I am aware of in the area of aggregate computations for probabilistic XML sources. I have introduced two formal models for probabilistic aggregates—the *possible-worlds* semantics and the *expectation* semantics. Though these semantics are declaratively defined over a large space of “compatible” semistructured instances, I am able to find a succinct way of representing them and manipulating them. I have presented algorithms to compute these aggregates and report on experiments I conducted showing the feasibility of my approach.

The marriage of semistructured model with probabilistic models is a natural pairing. It supports uncertainty not only over the schema but also over the instance. This is particularly useful in the processing of complex, noisy data that abounds in real world domains. In addition to uncertainty information, ontology provides another kind of information that are usually absent in traditional database systems.

With RDF’s recent approval as a web recommendation by the W3C, there is growing industrial interest in RDF databases and more and more companies endorse RDF as a web standard for describing ontology. As a result, RDF databases are expected to grow in size and number. The ability to search the web using more sophisticated methods than keyword search is appealing, and RDF provides a simple paradigm to accomplish this. Thus, it is critical to be able to build and query RDF databases.

Views are *key* to many issues in databases - relating both to performance and security. In the second part of this dissertation, I have studied the problem of *maintaining materialized views over an* RDF *database* and I have proposed algorithms that can update materialized views when the base RDF instance changes in any one of the following ways: (i) triples are added to it, (ii) triples are deleted from it, (iii) triples are modified, and (iv) resources are modified. None of these scenarios is theoretical - rather they occur all the time as new resources are deployed on the web, old resources disappear, and new relationships (e.g. links) connect existing resources. For each of these problems, I have provided an algorithm which develops a local search strategy starting from the data to be updated in order to find what the new view should add into or remove from the original view.

Furthermore, I have extended the problem to aggregate views where *aggregate* operations such as COUNT,SUM,AVG,MIN,MAX and so on as well as GROUPBY operations can be handled. I have then provided algorithms to maintain such views. In addition, I have also described how this problem can be transformed into a relational view maintenance problem which can use any standard view maintenance algorithms.

I have implemented a prototype system and conducted extensive scalability experiments. The results show that my approach works well for all kinds of database updates with very short running time. Most importantly, the naive idea of just dumping all the RDF data into a relational database and using standard relational view maintenance algorithms leads to significantly slower performance than if my algorithms were used directly on the RDF-graph data. Depending upon the precise update operation being considered, my algorithms are 3 to 177 times faster than the corresponding relational operations.

BIBLIOGRAPHY

[1] S. Abiteboul, P. Buneman, and D. Suciu. *Data on the Web.* Morgan Kaufmann, 2000.

[2] S. Abiteboul, R. Goldman, J. McHugh, V. Vassalos, and Y. Zhuge. Views for semistructured data. In *Workshop on Management of Semistructured Data*, Tucson, Arizona, May 1997.

[3] S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J.L. Wiener. The Lorel query language for semistructured data. *Journal of Digital Libraries*, 1(1):68–88, November 1996.

[4] Serge Abiteboul, Jason McHugh, Michael Rys, Vasilis Vassalos, and Janet L. Wiener. Incremental maintenance for materialized views over semistructured data. In *Proc. of the 24th International Conference on Very Large Databases (VLDB)*, pages 38–49, New York City, USA, August 1998.

[5] D. Barbara, H. Garcia-Molina, and D. Porter. The management of probabilistic data. *IEEE Trans. on Knowledge and Data Engineering*, 4:487–502, 1992.

[6] C. Beeri and Y. Tzaban. SAL: An algebra for semistructured data and XML. In *Informal Proceedings of the ACM International Workshop on the Web and Databases (WebDB'99)*, Philadelphia, Pennsylvania, USA, June 1999.

[7] P. Biron and A. Malhotra. XML schema part 2: Datatypes. W3C Recommendation, October 2004. Available at `http://www.w3.org/TR/xmlschema-2/`.

[8] Jos A. Blakeley, Per ke Larson, and Frank W. Tompa. Efficiently updating materialized views. In *SIGMOD Conference*, pages 61–71, 1986.

[9] G. Boole. *The Laws of Thought.* Macmillan, 1954.

[10] B. Bouwerman and R.T. O'Connell. *Forecasting and Time Series: An Applied Approach.* Brooks/Cole Publishing, Florence, Kentucky, USA, 2000.

[11] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, and Franois Yergeau. Extensible markup language (XML) 1.0 (third edition). W3C Recommendation, February 2004. Available at `http://www.w3c.org/TR/2004/REC-xml-20040204/`.

[12] R. Cavallo and M. Pittarelli. The theory of probabilistic databases. In *Proc. of the 13th VLDB Conference*, Brighton, England, September 1987.

[13] Stefano Ceri and Jennifer Widom. Deriving production rules for incremental view maintenance. In *Proceedings of the 17th International Conference on Very Large Data Bases*, pages 577–589, September 1991.

[14] S. Comai and P. Fraternali. Computing graphical queries over XML data. *ACM Transactions on Information Systems*, 19(4):371–430, October 2001.

[15] R. Dechter. Bucket elimination: A unifying framework for probabilistic inference. In *Proceedings of the 12th Conference on Uncertainty in Artificial Intelligence*, pages 211–219, Portland, Oregon, USA, 1996.

[16] A. Dekhtyar, J. Goldsmith, and S.R. Hawkes. Semistructured probabilistic databases. In *Proceedings of 2001 Conference on Statisitcal and Scientific Database Management (SSDBM)*, Fairfax, VA, USA, July 2001.

[17] A. Dekhtyar, R. Ross, , and V.S. Subrahmanian. Probabilistic temporal databases, i. *ACM Transactions on Database Systems*, 26(1), 2001.

[18] A. Deutsch, M. Fernandez, D. Florescu, A. Levy, and D. Suciu. XML-QL: A query language for XML. Available at `http://www.w3.org/xml/`, August 1998.

[19] D. Dey and S. Sarkar. A probabilistic relational model and algebra. *ACM Transactions on Database Systems*, 21(3):339 – 369, 1996.

[20] D. Dey and S. Sarkar. PSQL: A query language for probabilistic relational data. *Data and Knowledge Engineering*, 28, 1998.

[21] Open directory project rdf dump. http://rdf.dmoz.org/.

[22] C. Dyreson and R. Snodgrass. Supporting valid-time indeterminacy. *ACM Transactions on Database Systems*, 23(1):1 – 57, 1998.

[23] T. Eiter, J. Lu, T. Lukasiewicz, and V.S. Subrahmanian. Probabilistic object bases. *ACM Transactions on Database Systems*, September 2001.

[24] T. Eiter, T. Lukasiewicz, and M. Walter. Extension of the relational algebra to probabilistic complex values. In *Proc. of Int. Symposium on Foundations of Information and Knowledge Systems*, pages 94–115, Burg, Germany, 2000.

[25] Maged EL-Sayed, Ling Wang, Luping Ding, and Elke A. Rundensteiner. An algebraic approach for incremental maintenance of materialized XQuery views. In *ACM WIDM*, 2002.

[26] R. Fagin, J.Y. Halpern, and N. Megiddo. A logic for reasoning about probabilities. *Information and Computation*, pages 78–128, 1990.

[27] D. Florescu and D. Kossmann. A performance evaluation of alternative mapping schemes for storing XML data in a relational database. In *Proc. of the VLDB*, 1999.

[28] F. Frasincar, G. Houben, R. Vdovjak, and P. Barna. RAL: An algebra for querying RDF. *World Wide Web: Internet and Web Information Systems*, 7:83–109, 2004.

[29] J. Freire, J. R. Haritsa, M. Ramanath, P. Roy, and J. Simeon. StatiX: Making XML Count. In *Proc. of ACM SIGMOD '2002*, Madison, USA, June 2002.

[30] N. Friedman, L. Getoor, D. Koller, and A. Pfeffer. Learning probabilistic relational models. In *Proc. of IJCAI-99*, Stockholm, Sweden, 1999.

[31] N. Fuhr and K. GroBjohann. XIRQL a query language for information retrieval in XML documents. In *Proc. of SIGIR '2001*, New Orleans, USA, September 2001.

[32] N. Fuhr and T. Rolleke. A probabilistic NF2 relational algebra for intergrated information retrieval and database systems. In *Proc. of the 2nd World Conference on Integrated Design and Process Technology*, pages 17–30, 1996.

[33] N. Fuhr and T. Rolleke. A probabilistic relational algebra for the integration of information retrieval and database systems. *ACM Transactions on Information Systems*, 15(1):32–66, 1997.

[34] S. Goldman and R. Rivest. A non-iterative maximum entropy algorithm. In *Proceedings of the 2nd Annual Conference on Uncertainty in Artificial Intelligence (UAI-86)*, pages 133–148, New York, NY, 1986. Elsevier Science Publishing Comapny, Inc.

[35] Judy Goldsmith, Alex Dekhtyar, and Wenzhong Zhao. Can probabilistic databases help elect qualified officials? In *Proceedings of 2003 Florida Atlantic Artificial Intelligence Research Symposium (FLAIRS)*, pages 501 – 505, St. Augustine, FL, 2003.

[36] U. Guntzer, W. Kiessling, and H. Thone. New directions for uncertainty reasoning in deductive databases. In *Proceedings of ACM SIGMOD*, pages 178–187, 1991.

[37] A. Gupta and I.S. Mumick, editors. *Materialized Views: Techniques, Implementations and Applications*. The MIT Press, 1999.

[38] A. Gupta, I.S. Mumick, and D. Katiyar. Counting solutions to the view maintenance problem. In *Workshop on Deductive Databases, JICSLP*, 1992.

[39] Ashish Gupta, Inderpal Singh Mumick, and V.S. Subrahmanian. Maintaining views incrementally. In *Proceedings of the 1993 ACM SIGMOD international conference on Management of data*, pages 157–166, Washington, D.C., United States, May 1993.

[40] Himanshu Gupta and Inderpal Singh Mumick. Incremental maintenance of aggregate and outerjoin expressions. *Information System*, 2005.

[41] Hewlett-Packard. RDQL - RDF data query language. http://www.hpl.hp.com/semweb/rdql.htm.

[42] H.V. Jagadish, V.S. Lakshmanan, Divesh Srivastava, and Keith Thompson. TAX: A tree algebra for XML. In *Proc. of Int. Workshop on Database Programming Languages (DBPL'01)*, Roma, Italy, September 2001.

[43] Hewlett-packard. jena 2 – a semantic web framework. Available at http://www.hpl.hp.com/semweb/jena2.htm.

[44] G. Kamberova and R. Bajcsy. Stereo depth estimation: the confidence interval approach. In *Proc. of Intl. Conf. Computer Vision (ICCV98)*, Bombay, India, January 1998.

[45] Hyunchul Kang and JaeGuk Lim. Deferred incremental refresh of XML materialized views. In *CAiSE*, pages 742–746, 2002.

[46] Hyunchul Kang, Hosang Sung, and ChanHo Moon. Deferred incremental refresh of XML materialized views: Algorithms and performance evaluation. In *ADC*, pages 217–226, 2003.

[47] G. Karvounarakis, A. Magkanaraki, S. Alexaki, V. Christophides, D. Plexousakis, M. Scholl, and K. Tolle. Querying the semantic web with RQL. *Computer Networks and ISDN Systems Journal*, 42(5):617–640, August 2003.

[48] W. Kiessling, H. Thone, and U. Guntzer. Database support for problematic knowledge. In *Proceedings of EDBT, Springer LNCS Vol. 580*, pages 421–436, 1992.

[49] Harumi A. Kuno and Elke A. Rundensteiner. Incremental maintenance of materialized object-oriented views in multiview: Strategies and performance evaluation. *IEEE Transactions on Knowledge and Data Engineering*, 10, 1998.

[50] V.S. Lakshmanan, N. Leone, R. Ross, and V.S. Subrahmanian. ProbView: A flexible probabilistic database system. *ACM Transactions on Database Systems*, 22(3):419–469, 1997.

[51] V.S. Lakshmanan and F. Sadri. Probabilistic deductive databases. In *Proc. Int. Logic Programming Symp., (ILPS'94)*. MIT Press, 1994.

[52] V.S. Lakshmanan and N. Shiri. Parametric approach with deductive databases with uncertainty. In *Proc. Workshop on Logic In Databases*, pages 61–81, 1996.

[53] S. L. Lauritzen and D. J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society*, B 50(2):415–448, 1988.

[54] Tony T. Lee. An information-theoretic analysis of relational databases—part i: data dependencies and information metric. *IEEE Trans. Softw. Eng.*, 13(10):1049–1061, 1987.

[55] A. Magkanaraki, V. Tannen, V. Christophides, and D. Plexousakis. Viewing the semantic web through RVL lenses. In *Second International Semantic Web Conference (ISWC'03)*, Sanibel Island, Florida, USA, October 2003.

[56] Francesco M. Malvestuto. Statistical versus relational join dependencies. In James C. French and Hans Hinterberger, editors, *Seventh International Working Conference on Scientific and Statistical Database Management, September 28-30, 1994, Charlottesville, Virginia, USA, Proceedings*, pages 64–73. IEEE Computer Society, 1994.

[57] Alberto Martell and Ugo Montanari. An efficient unification algorithm. *TOPLAS*, 4(2):258–282, 1982.

[58] J. McHugh, S. Abiteboul, R. Goldman, D. Quass, and J. Widom. Lore: A database management system for semistructured data. *SIGMOD Record*, 26(3):54–66, September 1997.

[59] J. McHugh, S. Abiteboul, R. Goldman, D. Quass, and J. Widom. Lore: A database management system for semistructured data. *SIGMOD Record*, 26(3):54–66, 1997.

[60] H. Meuss and K. U. Schulz. Complete answer aggregates for tree-like databases: A novel approach to combine querying and navigation. *ACM Transactions on Information Systems*, 19(2):161–215, April 2001.

[61] A. Nierman and H.V. Jagadish. ProTDB: Probabilistic data in XML. In *Proc. of the 28th VLDB Conference*, Hong Kong, China, August 2002.

[62] openRDF.org. The serql query language. http://www.openrdf.org/doc/users/ch05.html.

[63] Themistoklis Palpanas, Richard Sidle, Roberta Cochrane, and Hamid Pirahesh. Incremental maintenance for non-distributive aggregate funcitons. In *Proceedings of the 28th VLDB Conference*, Hong Kong, China, 2002.

[64] Y. Papakonstantinou, H. Garcia-Molina, and J. Widom. Object exchange across heterogeneous information sources. In *Proc. of the Eleventh International Conference on Data Engineering*, pages 251–260, Taipei, Taiwan, March 1995.

[65] Stelios Paparizos, Shurug Al-Khalifa, H. V. Jagadish, Laks V.S. Lakshmanan, Andrew Nierman, Divesh Srivastava, and Yuqing Wu. Grouping in XML. In *EDBT Workshop on XML Data Management (XMLDM'02), Lecture Notes in Computer Science vol.2490.* Springer-Verlag, 2002.

[66] J. Pearl. *Probabilistic Reasoning in Intelligent Systems.* Morgan Kaufmann, San Francisco, 1988.

[67] N. Polyzotis and M. Garofalakis. Statistical synopses for graph-structured XML databases. In *Proc. of ACM SIGMOD '2002*, Madison, USA, June 2002.

[68] N. Polyzotis and M. Garofalakis. Structure and value synopses for XML data graphs. In *Proc. of the 28th VLDB Conference*, Hong Kong, China, August 2002.

[69] Dallan Quass. Maintenance expressions for views with aggregation. In *Proceedings of the SIGMOD'96 Workshop on Materialized Views*, pages 110–118, 1996.

[70] D. Radev, W. Fan, and H. Qi. Probabilistic question answering from the web. In *Proceedings of the 11th International World Wide Web Conference*, pages 408–419, Honolulu, Haiwaii, May 2002.

[71] RDF semantics. W3C Recommendation, 10 Feb 2004. http://www.w3.org/TR/rdf-mt/.

[72] RDQL - a query language for RDF. W3C Member Submission, 9 Jan 2004. http://www.w3.org/Submission/RDQL/.

[73] R. Ross, V.S. Subrahmanian, and J. Grant. Probabilistic aggregates. In *Proc. ISMIS 2002, Springer Lecture Notes in Artificial Intelligence (ed. Z. Ras)*, pages 553–564, Lyon, France, June 2002.

[74] R. Ross, V.S. Subrahmanian, and J. Grant. Aggregate operators in probabilistic databases. *Journal of the ACM*, 52(1):54–101, January 2005.

[75] Elke A. Rundensteiner. Multiview: A methodology for supporting multiple views in object-oriented databases. In *Proc. of 18th VLDB Conference*, Vancouver, Canada, 1992.

[76] H. Thompson, D. Beech, M. Maloney, and N. Mendelsohn. XML schema part 1: Structures. W3C Recommendation, October 2004. Available at `http://www.w3.org/TR/xmlschema-1/`.

[77] Kristin Tufte and David Maier. Aggregation and accumulation of XML data. *Data Engineering Bulletin*, 24(2), June 2001.

[78] R. Volz, S. Staab, and B. Motik. Incremental maintenance of materialized ontologies. In *Proceedings of CoopIS/DOA/ODBASE*, pages 707–724, 2003.

[79] Raphael Volz, Daniel Oberle, and Rudi Studer. Towards views in the semantic web. In *2nd Int'l Workshop on Databases, Documents and Information Fusion (DBFUSION02)*, Karlsruhe, Germany, 2002.

[80] Extensible Markup Language (XML). Available at `http://www.w3.org/XML`.

[81] Kevin Wilkinson, Craig Sayers, Harumi A. Kuno, Dave Reynolds, and Luping Ding. Supporting scalable, persistent semantic web applications. *IEEE Data Eng. Bull.*, 26(4):33–39, 2003.

[82] XML Path Language (XPath) 2.0, W3C working draft. Available at `http://www.w3.org/TR/xpath20/`.

[83] XQuery 1.0: An XML Query Language, W3C working draft. Available at `http://www.w3.org/TR/xquery/`.

[84] Ke Yi, Hao He, Ioana Stanoi, and Jun Yang. Incremental maintenance of XML structural indexes. In *SIGMOD*, 2004.

[85] Wenzhong Zhao, Alex Dekhtyar, and Judy Goldsmith. Query algebra operations for interval probabilities. In *Proceedings of 2003 Iternational Conference on Database and Expert Systems Applications (DEXA)*, pages 527 – 536, Prague, Czech Republic, 2003.

[86] Wenzhong Zhao, Alex Dekhtyar, and Judy Goldsmith. Databases for interval probabilities. *International Journal of Intelligent Systems (IJIS)*, 19(9):789–815, 2004.

[87] Yue Zhuge and Hector Garcia-Molina. Graph structured views and their incremental maintenance. In *Proceedings of the 14th International Conference on Data Engineering (ICDE)*, pages 116–125, 1998.

[88] Yue Zhuge and Hector Garcia-Molina. Self-maintainability of graph structured views. Technical report, Stanford University, September 1998.