Structured Queries for Semistructured Probabilistic Data

Alex Dekhtyar*
Department of Computer Science
University of Kentucky
Lexington, KY, USA
dekhtyar@cs.uky.edu

Krol Kevin Mathias*
Department of Computer Science
University of Kentucky
Lexington, KY, USA
kevin9@uky.edu

Praveen Gutti
Department of Computer Science
University of Kentucky
Lexington, KY, USA
praveengutti@uky.edu

ABSTRACT

We present SPOQL, a structured query language for Semistructured Probabilistic Object (SPO) model [4]. The original query language—SP-Algebra [4], has traditional limitations like terse functional notation and unfamiliarity to application programmers. SPOQL alleviates these problems by providing familiar SQL-like declarative syntax. We show that parsing SPOQL queries is a more involving task than parsing SQL queries. We also present an eager evaluation algorithm for SPOQL queries.

Categories and Subject Descriptors

H.2.3 [**Information Systems**]: DatabasesLanguages

General Terms

Languages, Algorithms

Keywords

probabilistic databases, query langauges

1. INTRODUCTION

With the emergence of new applications of database technologies to dynamic environments in the past couple of years, management of uncertain information has again attracted the attention of database researchers. Replacing early, relational models for storage and processing of probabilistic data [12, 2, 6, 1] new, object-oriented [11, 8] and semistructured models [10, 14, 4, 20] have been proposed.

The need for special approach(es) to management of probabilistic data comes from two key observations. First, probabilities stored in the databases must be manipulated (combined, marginalized, conditionalized) in accordance with the laws of probability theory. Second, probabilities are often associated with objects that have more complex structure than relational tables. The latter observation suggests the use of object-oriented or semistructured approaches, while the former suggests that standard notions of queries have to be revised in order to admit correct manipulation of probabilities.

In [4, 20] Dekhtyar et al. have introduced a framework for management of probabilistic data called Semistructured Probabilistic Objects (SPOs). SPOs are designed to store, as objects, probability distributions of discrete random variables with finite domains together with non-stochastic (context, situation-specific) information. The design of SPOs allows to store in the same Semistructured Probabilistic relation (SP-relation) an arbitrary collection of such probability distributions: different SPOs in the relation can have completely different structure and content.

To query SP-relations, Dekhtyar et al. introduced Semistructured Probabilistic Algebra (SP-Algebra), a query algebra defining the semantics of traditional relational algebra operations, such as selection, projection and join. In addition, SP-Algebra contains an operation of conditionalization (conditioning) used to construct conditional probability distributions. In [20], Zhao et al. report on the implementation of Semistructured Probabilistic DBMS, SPDBMS. Built on top of an RDBMS, it provides a level of abstraction, hiding from the users the details of storage of SPOs in relational databases. Client applications use SP-Algebra queries to access the information stored in SPDBMS.

In its current stage, SPDBMS is used by a number of client applications [3, 21] developed for a research project devoted to planning in the presence of uncertain information[13].
This paper describes the newest component of SPDBMS, Semistructured Probabilistic Object Query Language (SPOQL) – an SQL-like declarative language for queries over SPO data. SPOQL is designed to replace SP-Algebra as the interface of choice for communication with the SPDBMS server. While SP-Algebra allows us to study various properties of SPOs in a rigorous manner, it is inconvenient to use as the end-user (or client) query language. SPOQL provides convenient declarative syntax for queries and represents complex queries over SPO data in a straightforward manner.

In Section 2 we give a brief overview of the SPO model and SP-Algebra. Section 3 then introduces SPOQL. We discuss the syntax of the language, and describe how SPOQL queries are translated into SP-Algebra expressions for further processing within SPDBMS.

2. SEMISTRUCTURED PROBABILISTIC OBJECTS

2.1 Motivating Example

Semistructured Probabilistic Object model provides straightforward and convenient means of storing probability distributions of discrete random variables as single objects. Early work on uncertainty in databases added probabilities to relational tuples, but also attempted to treat groups of tuples within the same relation as a

*Work supported, in part, by the NSF grant ITR-0325063.

probability distribution [2, 6]. A more sophisticated approach [1] abandoned first normal form in probabilistic relations, and stored an entire probability distribution as part of a single tuple. Still, the probabilistic database model of [1] was relational, which, in particular, meant that only similarly structured probability distributions could be stored in the same relation.

The original motivation for the SPO model came from the work of one of the authors on building Bayesian models for complex domains[5, 13]. Bayesian networks [15] are graphical representations of conditional dependencies (and independencies) between various (discrete) random variables (with finite domains). Traditional Bayesian network applications yield relatively small (as far as databases are concerned) models, with relatively few probability tables to be stored and manipulated. However, some of the emerging applications present a case for proper data management strategies, as illustrated on the following example, borrowed from [13].

At present, our research group is working on modelling advising in the Welfare-to-Work (WtW) system using Bayesian networks [3, 13]. In a nutshell, WtW is a government program that provides certain benefits (food stamps, health insurance, daycare, stipend) to its clients[1] in exchange for their participation in services and activities designed to improve the clients' employability. A WtW case manager is assigned to work with each client, provide advice and monitor progress. Case managers base their advice to clients on the estimates of the clients' likely success in specific activities. Such estimates are based on client characteristics (such as literacy level, work preparedness, and time-management skills); the case managers knowledge about available services (such as mental-health care and job-training classes) and regulations; the client's current record with the program; and the case manager's experience with this and other clients.

We model client performance in specific activities as random variables conditionally dependent on various client characteristics listed above. In addition, going through an activity and succeeding (or failing) in it also has a stochastic effect on client characteristics themselves.

In addition to random variables influencing the probability of success in a specific activity, there are numerous non-stochastic factors as well. For example, such characteristics of the client as age,and number of children are not stochastic, but may clearly affect the client's ability to succeed in volunteer work, GED study or any of other possible activities. In addition, activities themselves may possess non-stochastic characteristics, such as location and/or name of the provider/facility, time of day the activity is offered, etc. As a result of the presence of this non-stochastic information, often referred to as "meta-data" or "context", multiple CPTs get associated with each random variable.

The Bayesian network inference/planning software, on the other hand, does not need to work with *all* CPTs at the same time. Based on the currently available information about an individual client, that software would need to select appropriate CPTs from the available collection. The latter calls for careful approach to storage and manipulation of CPTs and other probability distributions. SPO model and SP-Algebra, described below, answer that challenge.

2.2 SPO model and SP-Algebra

In this section we define SPOs — Semistructured Probabilistic Objects — a flexible data structure to represent probability distributions, and SP-Algebra, an algebra of atomic query operations on SPOs. An **SPO** S [20] is a tuple $S = \langle T, V, P, C, \omega \rangle$, where

[1] In practice, welfare-to-work clients are unemployed single parents.

ω: **S1**
work-type: nursing
city: Lexington

VOP	P
success	0.75
failure	0.25

A = high G = high
S = high WR = low
C = high

(a)

ω: **S2**
city: Lexington

WR	P
high	0.8
low	0.2

WH = good

(b)

ω: **S3**

S	P
high	0.7
low	0.3

WH = good

(c)

ω: $\sigma_{tbl.prob \geq 0.75}(S1)$
work-type: nursing
city: Lexington

VOP	P
success	0.75

A = high G = high
S = high WR = low
C = high

(d)

ω: $\pi_f S1$
city: Lexington

VOP	P
success	0.75
failure	0.25

A = high

(e)

ω: $S4 = S2 \times S3$
city: Lexington

WR	S	P
high	high	0.56
low	high	0.14
high	low	0.24
low	low	0.06

WH = good

(f)

ω: $\mu_{WR=high}(S4)$
city: Lexington

S	P
high	0.7
low	0.3

WH = good
WR = high

(g)

Figure 1: Probability distributions for the random variables in the activity "Volunteer Placement (VOP)", where projection condition f is $cnt.city \wedge cnd.A$.

- T is a relational tuple over some *semistructured schema* over $\mathcal{R}$, the universe of context attributes. T refers to the ***context*** of S.
- $V = \{v_1, \ldots, v_q\} \subseteq \mathcal{V}$ is a set of ***random variables*** that participate in S, where $\mathcal{V}$ is the universe of random variables and $V \neq \emptyset$.
- $P : dom(V) \longrightarrow [0, 1]$ is the ***probability table*** of S. Note that P need not be complete.
- $C = \{(u_1, X_1), \ldots (u_s, X_s)\}$, where $\{u_1, \ldots, u_s\} = U \subseteq \mathcal{V}$, and $X_i \subseteq dom(u_i)$, $1 \leq i \leq n$, such that $V \cap U = \emptyset$. C refers to the ***conditional*** of S.
- ω, called the ***path expression***, is an expression of SP-Algebra.

Informally, SPOs store probability distributions as follows. The actual distribution is described by the participating random variables and probability table parts of the object. Its path tells us how this object has been constructed. Atomic (single unique identifier) paths indicate that the object has been *directly inserted into the DBMS* whereas complex paths indicate which database objects participated in its creation and what SP-Algebra operations have been applied to obtain it. Context part of the SPO stores the non-stochastic information related to the distribution, while the conditional part, stores conditioning information.

A collection $\{S_1, \ldots, S_n\}$ of SPOs is called an **SP-relation**.

EXAMPLE 1. *We use SPOs to represent the probability distributions associated with the model described in Section 2.1. Figure 1 contains some of the distributions stored for the random variables associated with activity "Volunteer Placement (VOP)". The SPO in Figure 1(a), for example, represents the probability distribution of client's performance in volunteer placement related to nursing for a client who resides in the city of Lexington, is highly skilled and goal-oriented, has high confidence and aptitude, but low work-readiness. In this SPO, VOP is the participating random variable representing client performance in the volunteer placement activity. Its domain has two values, success and failure and their respective probabilities are given in the probability table part of the SPO. The context of the SPO is formed by two name-value pairs, specifying the type of work (nursing) and the city of client's residence (Lexington). Finally, the conditional part of the SPO contains information about client's aptitude (A), skills level (S), goals (G), confidence (C) and work-readiness (WR). The path expression identifying this SPO, S1 indicates that it had been inserted into the database as-is.*

Similarly, we can take a look at a slightly more complex SPO in Figure 1(f). This SPO represents the joint probability distribution of two random variables, Work Readiness (WR) and Skills (S), for a client who resides in Lexington and has good work history. WR and S are the two participating random variables. Each has a binary domain (high, low), and the probability table for all four possible combinations is included in the SPO. The context for the SPO is city:Lexington, and the conditional contains a single conditioning statement, specifying the the client's work history is good (WH=good). The path of this SPO is S2× S3, an SP-Algebra expression (see below) indicating that this SPO had been constructed out of SPOs with path IDs S2 and S3 using the SP-algebra operation of cartesian product, described later in this section.

In the remainder of this section we describe the SP-Algebra, query algebra for the SPO model. It includes the definitions of standard relational operations of selection, projection, cartesian product and join. It also includes the conditionalization operation (see also [6]), specific to the probabilistic databases. In this paper, we have extended the original SP-Algebra to include the mix operation. We have also defined join conditions that can be applied to the SP-Algebra operations of cartesian product, join, and mix. The formal definitions of the mix operation and join conditions are provided in this paper. For the formal definitions of the remaining SP-Algebra operations, please refer to [20].

In the definitions for **Atomic projection** list, **Atomic selection** conditions, and **Atomic conditionalization** expression; var, cnt, cnd, and tbl are the notations used to represent **random variable**, **context**, **conditional** and **probability table** parts of an **SPO** object. Each of the above notational elements can be referenced by $[Name.]$var, $[Name.]$cnt, $[Name.]$cnd, and $[Name.]$tbl respectively. The optional parameter $Name$ represents the name of the SP-relation.

Atomic projection list is defined as:
F ::= varlist | cntlist | cndlist
varlist ::= "var".$\langle name\rangle$(, "var".$\langle name\rangle)^\star$, where $name \in \mathcal{V}$
cntlist ::= "cnt".$\langle name\rangle$(, "cnt".$\langle name\rangle)^\star$, where $name \in \mathcal{R}$
cndlist ::= "cnd".$\langle name\rangle$(, "cnd".$\langle name\rangle)^\star$, where $name \in \mathcal{V}$

Atomic selection conditions are described in Table 1.
Atomic conditionalization expression is defined as:
"var".$\langle name\rangle$ = Value, where $name \in \mathcal{V}$
"var".$\langle name\rangle \in$ Value(, Value)*, where $name \in \mathcal{V}$

Type	Expression	Explanation
Context condition	"cnt".$\langle name\rangle\langle Op\rangle$ constant "cnt".$\langle name\rangle \in T$	$name \in \mathcal{R}$ Op : $=, \leq, \geq, \neq, <, >$
Variable condition	"var".$\langle name\rangle \in V$	$name$ - variable name, where $name \in \mathcal{V}$
Conditional condition	"cnd".$\langle name\rangle$ = Value, "cnd".$\langle name\rangle \in C$ "cnd".$\langle name\rangle \in$ Value(, Value)*	$name$ - variable name, where $name \in \mathcal{V}$
Table condition	"tbl".$\langle name\rangle$ = Value "tbl"."prob" Op RValue	Op : $=, \leq, \geq, \neq, <, >$ RValue $\in$[0,1]

Table 1: Atomic selection conditions

Selection condition is inductively defined as:
Base: Atomic selection condition is a selection condition.
Induction: Let c_1 and c_2 be selection conditions. Then, the following are selection conditions: $c_1 \vee c_2$, $c_1 \wedge c_2$, $\neg c_1$.

Join condition is defined as:
$\langle Name\rangle$."cnt".$\langle name\rangle$ $\langle Op\rangle$ $\langle Name\rangle$."cnt".$\langle name\rangle$ where,
$Name$ - name of sp-relation, $name \in \mathcal{R}$, and
Op : $=, \leq, \geq, \neq, <, >$.

SP-Algebra expressions are inductively defined as:
Base: Let S be a name of an SP-relation. S_a is an SP-Algebra expression.
Induction: Let e_1 and e_2 be SP-Algebra expressions. Let c be a selection condition (SC), f be a projection list (PL), d be a conditionalization expression (CE) and g be a join condition (JC). The SP-Algebra expressions are described in Table 2.

Type	SP-Algebra expression
selection	$\sigma_c(e_1)$
projection	$\pi_f(e_1)$
conditionalization	$\mu_d(e_1)$
cartesian product(CP)	$e_1 \times e_2$
CP with join condition	$e_1 \times_g e_2$
join	$e_1 \ltimes e_2, e_1 \rtimes e_2$
join with join condition	$e_1 \ltimes_g e_2, e_1 \rtimes_g e_2$
mix	$e_1 \otimes e_2$
mix with join condition	$e_1 \otimes_g e_2$

Table 2: SP-Algebra expressions

Selection($\sigma_c(S)$) operation finds SPOs in an SP-relation that satisfy a particular selection condition. The selection operations on *context*, *participating variables* or *conditionals* do not alter the content of the selected objects (SPOs). When the selection operations are either on probabilities or the probability table, the SPO returned retains the same context, participating variables and conditional information, but will only include probability table rows that match the selection condition. Figure 1(d) shows an SPO that answers the query $\sigma_{\text{tbl.prob} \geq 0.75}(S)$, where $S = \{S1\}$. In English this query is expressed as follows: "Find all probability distributions in which there is an outcome with probability equal to or greater than 0.75 and select those outcomes."

Projection($\pi_f(S)$) is the operation of simplifying SPOs. The projection operations on *context* and *conditionals* are similar to the traditional relational algebra projection: either a context attribute or a conditional is removed from an SPO object, which does not change otherwise. The projection operation on the set of *participating random variables* corresponds to removing the other random variables from consideration in a joint probability distribution. The result of this operation is a new *marginal probability distribution* that is stored in the probability table component of the resulting SPO. Figure 1(e) provides an SPO that gives result of projecting out all conditionals except those related to client's Aptitude, and the work-type context attribute from the SPO $S1$ (Figure 1(a)), expressed as $\pi_{\mathsf{cnt.city}}(\pi_{\mathsf{cnd.A}}(S1))$. Note slight deviation from traditional relational algebra notation: the inner expression $\pi_{\mathsf{cnd.A}}(S1)$ returns an SPO which has the same context, participating variables and probability table as $S1$, projecting out only conditional parts — if a component is omitted in the projection list altogether, it is kept intact.

Conditionalization ($\mu_d(S)$) is the operation of conditioning the joint probability distribution. First, it removes from the probability table of the SPO all rows that do not match the condition. Then the variable column (given in the condition) is removed from the table. The remaining rows are coalesced (if needed) in the same manner as in the projection operation and the probability values are normalized. Finally, the condition is added to the set of conditionals of the resulting SPO. Figure 1(g) shows the probability distribution of client skills (S) given that the client has high work-readiness (WR) and good work history (WH), obtained from $S4$ (Figure 1(f)): $\mu_{\mathsf{var.WR=high}}(S4)$.

Cartesian product ($\times$) and **join** ($\ltimes$, $\rtimes$) construct a joint probability distribution from the input SPOs. The difference is that join is applicable to the SPOs that have common participating random variables whereas cartesian product is applicable to SPOs with disjoint lists of participating variables. Two SPOs are *cp-compatible*, if their participating variables are disjoint, and their conditionals coincide. When the sets of participating variables are not disjoint, but their conditionals coincide, the two SPOs are *join-compatible*. Figure 1(f) provides an SPO representing the joint distribution (Skills and Work-readiness) created from their respective SPOs in Figure 1(c) and Figure 1(b).

Mix($\otimes$) operation also constructs a joint probability distribution from the input SPOs.

When two SP-relations S and S' are provided as input, they will have join-compatible SPOs , cp-compatible SPOs and neither join-nor cp-compatible SPOs. The mix operation is the union of the join and the cartesian product S and S'.

Let $S = \langle T, V, P, C, \omega\rangle$ and $S' = \langle T', V', P', C', \omega'\rangle$ are two cp-compatible or join-compatible SPOs. Then, the result of their mix operation, denoted $S \otimes S'$, is defined as follows [2]:

$S \otimes S' = (S \times S') \cup (S \ltimes S')$

Cartesian product, join, mix with join conditions extend the combination operations of SP-Algebra to incorporate joining based on relationships between context attributes in the SPOs being joined. In addition to the conditions that are applicable to SPOs that participate in cartesian product, join and mix operations, their **context** elements should satisfy the join condition. Elements that do not satisfy the join condition are not combined together. For example, in Figure 1, SPOs $S2$ and $S3$ are cp-compatible but cannot be combined under the join condition '$S2.cnt.city = S3.cnt.city$'(since $S3$ does not contain context element 'city').

[2]In [20] two join operations are defined - left join ($\ltimes$) and right joint ($\rtimes$). The participating variables in the result of join include exactly one copy of common variables; left/right joins differ by whether they are conditioned from second or first SPO. Technically, we need to introduce two mix operations, one for left and one for right joins. For simplicity, we consider only one operation in this paper.

2.3 SP-Algebra equivalences

In [19], in preparation for query optimization for SP-Algebra, Zhao has proved SP-Algebra equivalences shown in Table 3. In addition, we have established the equivalences in Table 4 that involve the join and cartesian product operators.

Equivalence	Condition
$\sigma_{c \wedge c'}(e_1) \equiv \sigma_c(\sigma_{c'}(e_1))$	c and c' are SCs.
$\sigma_c(\sigma_{c'}(e_1)) \equiv \sigma_{c'}(\sigma_c(e_1))$	c and c' are SCs.
$\pi_{f \cap f'}(e_1) \equiv \pi_f(\pi_{f'}(e_1))$	f and f' are PLs.
$\pi_f(\pi_{f'}(e_1)) \equiv \pi_{f'}(\pi_f(e_1))$	f and f' are PLs.
$\mu_{d \wedge d'}(e_1) \equiv \mu_d(\mu_{d'}(e_1))$	d and d' are CEs.
$\mu_d(\mu_{d'}(e_1)) \equiv \mu_{d'}(\mu_d(e_1))$	d and d' are CEs.
$e_1 \times e_2 \equiv e_2 \times e_1$	e_1 and e_2 are cp-compatible.
$(e_1 \times e_2) \times e_3 \equiv e_1 \times (e_2 \times e_3)$	e_1, e_2, e_3 are cp-compatible.

Table 3: Query Equivalences for SP-Algebra operations.

Equivalence	Condition
$e_1 \ltimes (e_2 \times e_3) \equiv (e_1 \ltimes e_2) \times e_3$	e_1, e_2 are join compatible and e_2, e_3 are cp-compatible.
$e_1 \rtimes (e_2 \times e_3) \equiv (e_1 \rtimes e_2) \times e_3$	e_1, e_2 are join compatible and e_2, e_3 are cp-compatible.
$e_1 \times (e_2 \ltimes e_3) \equiv (e_1 \times e_2) \ltimes e_3$	e_1, e_2 are cp-compatible and e_2, e_3 are join compatible.
$e_1 \times (e_2 \rtimes e_3) \equiv (e_1 \times e_2) \rtimes e_3$	e_1, e_2 are cp-compatible and e_2, e_3 are join compatible.
$e_1 \ltimes e_2 \equiv e_2 \rtimes e_1$	e_1 and e_2 are join-compatible P(X, Y)* P($Z\|Y$) = P($X\|Y$)* P(Y, Z), where X, Y are variables from e_1 and Y, Z are variables from e_2.

Table 4: Query Equivalences for SP-Algebra operations.

No less important than the equivalences that hold *are those that do not*. In particular, we note that projection, conditionalization and cartesian product/join/mix operations change the probability distribution stored in the probability table of the resulting SPO. Thus, selections on probabilities cannot be, in general, commuted with other operations. For example, the result of SP-Algebra expression $\pi_{var.WR}(\sigma_{tbl.prob<0.5}(S4))$ is not the same as $\sigma_{tbl.prob<0.5}(\pi_{var.WR}(S4))$ as shown in Figure 2. This fact has significant effect on the query translation mechanism from SPOQL to SP-Algebra, presented in the next section.

2.4 Implementation

SPDBMS is implemented on top of an RDBMS in Java. Figure 3 depicts the overall architecture of the system [20]. The SPDBMS application server processes query requests like standard database management instructions and SPOQL queries from a variety of client applications.

The application server provides a JDBC-like API, through which client applications can send standard database management instructions, such as CREATE DATABASE, DROP DATABASE, CRE-

ω: $\pi_f(\sigma_c(S4))$	
city: Lexington	
WR	P
high	0.24
low	0.2
WH = good	

ω: $\sigma_c(\pi_f(S4))$	
city: Lexington	
WR	P
low	0.2
WH = good	

Figure 2: SPOs resulting from two SP-Algebra expressions, where selection condition is $c : tbl.prob < 0.5$ and projection condition is $f : var.WR$ and $S4$ is given in Figure 1(f).

ATE SP-RELATION, DROP SP-RELATION, INSERT INTO SP-RELATION, DELETE FROM SP-RELATION, as well as SP-Algebra queries to the server. Our SPOQL implementation has been integrated into the architecture shown in Figure 3.

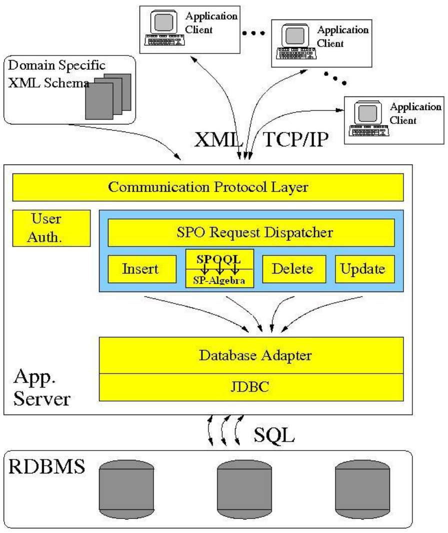

Figure 3: The overall architecture of SPDBMS.

3. SPOQL

Since its deployment, SPDBMS has been used as a back end for a number of programs designed to build and manipulate Bayesian networks. One of the lessons learned during this time was that SP-Algebra syntax was not the most convenient way for programmers, unfamiliar with the internals of SPDBMS to express queries. To alleviate this problem we have investigated replacing SP-Algebra with declarative syntax that would look familiar to programmers with some SQL experience. SPOQL, thus was born.

3.1 Syntax of SPOQL

The basic form of an **SPOQL** query is as follows:

```
SELECT <selectlist>
FROM <fromlist>
[WHERE <condition>]
[CONDITIONAL <conditionlist>]
```

Intuitively, a SPOQL query corresponds to an SP-Algbra expression involving selections, projections, conditionalizations, and combining operations (mix, cartesian products, join). Every SPOQL query returns an SP-relation (collection of SPOs).

Every SPOQL query must have a **SELECT** clause, which specifies the variables (context, conditional, random) of an SP-relation to be retained in the result, and a **FROM** clause, which specifies the list of participating SP-relations along with combining operations. The optional **WHERE** clause specifies selection conditions on the SP-relations mentioned in the **FROM** clause and join conditions on the combining operations mentioned in the **FROM** clause. The optional **CONDITIONAL** clause specifies the conditionalization expressions on the SP-relations mentioned in the **FROM** clause.

Thus, the $selectlist$ is a sequence of random variables, context variables, and conditionals that are involved in the projection operation. Every element of the $selectlist$ addresses an SP-relation from the $fromlist$. * is used in the absence of the projection operation when the entire object has to be returned.

The $fromlist$ is a sequence of SP-relations separated by combining operations “TIMES”, “JOIN” and “,”. The “,” stands for the mix operator, “TIMES” for cartesian product and “JOIN” for the join (left join is the default). An SP-relation in the $fromlist$ can also be an SPOQL query. This provides for nesting in an SPOQL queries. In addition, parentheses can be used to specify associativity of combination operations, and *SP-relation aliases*, similar to SQL’s table aliases are allowed.

The *condition* is a sequence of selection and join conditions separated by the keyword “AND”. Every selection condition addresses an SP-relation from the $fromlist$. Likewise every join condition addresses a combining operation from the $fromlist$ based on the SP-relations provided by it.

The *conditionlist* is a sequence of conditionalization expressions separated by the keyword “AND”. Every conditionalization expression addresses an SP-relation/alias from the $fromlist$.

3.2 SPOQL Semantics By Example

Let us consider a few simple SPOQL queries. SPOs in Figure 1(d), Figure 1(e), Figure 1(f) and Figure 1(g) can be obtained[3] using the following SPOQL queries.

1. ```
   SELECT * FROM S1
   WHERE S1.tbl.VOP = 'success'
   AND S1.tbl.prob > 0.7
   ```
2. ```
   SELECT S1.cnt.city, S1.cnd.A FROM S1
   ```
3. ```
   SELECT * FROM S2 JOIN S3
   ```
4. ```
   SELECT * FROM S4
   CONDITIONAL S4.var.WR='high'
   ```

Each of the above queries represents a single SP-Algebra operation. More complex SPOQL queries can represent multiple SP-Algebra operations. Let us consider an SPOQL query Q:

```
SELECT * FROM S2, S3
WHERE S2.tbl.WR = 'high' AND
S3.tbl.prob < 0.7
```

[3] $S1$, $S2$ and $S3$ in Figure 1 are ids of individual SPOs, not SP-relations, however, without loss of generality, we assume here that $S1$, $S2$ and $S3$ are also names of singleton SP-relations that contain the respective SPOs.

This query raises a number of questions concerning its SP-Algebra translation. We can see that it involves one mix operation and two selection operations, but in what order should these operations be performed? In case of classical relational algebra, selection and cartesian product/join operations commute and the exact order of operations produced by SQL query parsers is not very relevant, as the query tree/execution plan will be finalized at the query optimization stage. In SP-Algebra, however, selections on probabilities and join/mix/product operations do not commute, and therefore, we need to declare that *intent* of the query upfront, and make certain that the SPOQL query parser interprets it correctly. In the case of query Q, our choice is to translate it using the $\sigma_c(.)\otimes\sigma'_c(.)$ structure or using the $\sigma_c(.\otimes.)$ structure. We note that selection conditions in Q specify precisely the SP-relations on which they have to be performed, thus making $\sigma_{\mathsf{tbl.WR}='\mathsf{high}'}(S2)\otimes\sigma_{\mathsf{tbl.prob}<0.7}(S3)$ (see also Figure 4) i.e., the former structure, the natural choice.

This example suggests that we must be more careful in our translation of SPOQL queries into SP-Algebra, than SQL query parsers translating into relational algebra. It becomes important for us to define a precedence list for the SP-Algebra operations in order to achieve consistent query evaluation. Consistent with our reasoning in the previous example, we use the following order of precedence for every SP-relation belonging to the $fromlist$:

1. conditionalization operation using $conditionlist$;
2. selection operation using selection conditions from $condition$;
3. projection operation using $selectlist$;
4. join/times/mix operation using combining operations from $fromlist$ and join conditions from $condition$.

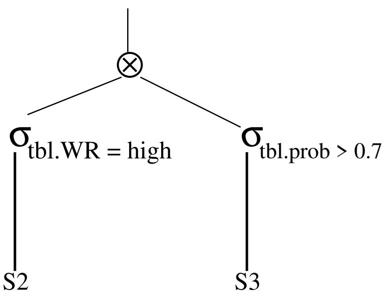

Figure 4: The query tree for SPOQL query Q.

To illustrate the effect of this decision, we show the SPOQL query representing $\sigma_{\mathsf{tbl.WR}='\mathsf{high}'\wedge\mathsf{tbl.prob}<0.7}(S2\otimes S3)^4$:

```
SELECT * FROM S2, S3
WHERE tbl.WR='high' AND tbl.prob < 0.7
```

There are other translation issues that need to be handled. Nesting and aliasing is allowed in SPOQL query to provide flexibility or override the precedence order. To ensure proper query translation/evaluation, we define a separate scope for each level of nesting within an SPOQL query, and enforce a scoping rule that does not permit any elements from the $selectlist$, $condition$, and $conditionlist$ to address SP-relations from the $fromlist$ not belonging to the same query level.

[4] var, cnt, cnd, and tbl without $Name$ of SP-relation, refer to the SP-relation resulting from the usage of combining operations on individual SP-relations from the $fromlist$.

The next issue to consider relates to the order in which SP-relations from the $fromlist$ are combined. In the absence of join conditions, this does not matter, and the traditional left-to-right evaluation can be performed. However, as the following example illustrates, join conditions require special handling.

Let $S5$, $S6$, and $S7$ represent the SP-relations that describe the distribution for the client's characteristics - Aptitude, Goals, and Confidence respectively. We are interested in knowing the joint distribution of these client characteristics in the year 1999 for Goals, and similar age groups for Aptitude and Confidence. A straightforward way to express it in SPOQL is the query Q' below:

```
SELECT * FROM S5 TIMES S6 TIMES S7
WHERE S6.cnt.year = 1999 AND
S5.cnt.age-group = S7.cnt.age-group
```

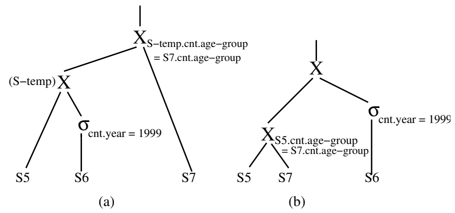

Figure 5: The query trees for SPOQL query Q' based on only left-to-right evaluation (a) and left-to-right evaluation with join condition(s) priority (b).

If left-to-right evaluation of the $fromlist$ is used, the resulting query tree for Q' is the one in Figure 5(a). The actual expression we wanted to represent with Q' is shown in Figure 5(b). In it, $S5$ and $S7$ are combined first, because of the join condition present in the query. Thus, we must ensure that the SPOQL query parser/translator, will override the default order of combining SP-relations in the $fromlist$ for Q' and similar queries. Informally, we can verbalize this idea as *materialize the combinations of SP-relations under join conditions first*.

We note, that we can also explicitly override the order combining SP-relations by using parentheses in the $fromlist$. The following query is equivalent to Q':

```
SELECT * FROM S6 TIMES (S5 TIMES S7) WHERE
S6.cnt.year = 1999 AND
S5.cnt.age-group = S7.cnt.age-group
```

The query tree for this explicit ordering query is the same as in Figure 5(b). But there is a small issue of identifying identical pairs of SP-relations in both the orderings. We resolve this issue by applying any join conditions that refer to the same explicit ordering pair of SP-relations. As a result, duplicate pairs are eliminated. Thus, the SPOQL evaluation algorithm implicitly builds a tree structure by applying the precedence rule and then determining the order of join, cartesian or mix operations with the SP-relations as the leaves.

The algorithm that evaluates an SPOQL query is shown in Figure 6. The input to the evaluation algorithm is the SPOQL query Q and its output is the query tree describing the SP-Algebra expression, that represents the (operational) semantics of Q. In this algorithm, **Build-path()** is the function that builds a conditionalization-selection-projection subtree for each SP-relation/SP-relation alias. **Build-subtree()** is the function that produces a query subtree for a

Evaluate(SPOQL query Q)

1. Validate Q to check for scoping consistency and non duplicate SP-relations in the $fromlist$.
2. if valid, then next step, else "Semantic error".
3. Extract $fromlist$, $selectionlist$, $condlist$, $projlist$ from Q.
4. Let $fromlist = (E_1, \cdots, E_k)$
5. for i = 1 to k do
   T_i = Build-subtree(E_i , $selectionlist, condlist, projlist$)
6. $T = T_1$
   for i = 2 to k do
   T = Combine(T, T_i)
7. return Build-path(T, $selectionlist, condlist, projlist$)

Build-subtree($fromExp, selectionlist, condlist, projlist$)

1. if $fromExp$ is a SPOQL query,
   return Evaluate($fromExp$)
2. else
   if $fromExp$ is an SP-relation,
   return Build-path($fromExp, selectionlist, condlist, projlist$)
   else
   Let $fromExp = (E_1\ Op\ E_2)Name$
   T_1 = Build-subtree(E_1 , $selectionlist, condlist, projlist$)
   T_2 = Build-subtree(E_2 , $selectionlist, condlist, projlist$)
   $T = Op(T_1, T_2)$
   if $Name$ == "" return T
   else return Build-path(T, $selectionlist, condlist, projlist$)

Build-path($fromExp, selectionlist, condlist, projlist$)

1. Perform conditionalization operation on $fromexp$ using $condlist$.
2. Perform selection operation on resultant $fromexp$ using $selectionlist$.
3. Perform projection operation on resultant $fromexp$ using $projlist$.
4. return resultant SP-relation tree T'.

Figure 6: Algorithm for translating SPOQL queries into SP-Algebra expressions.

single $fromlist$ entry (either an SP-relation, or an nested SPOQL query, or a nested join/product/mix expression).

Let us look at an example to understand the working of the **Evaluate** algorithm. Consider an SPOQL query Q'' of the form:

```
SELECT S7.cnt.age-group FROM
S5 TIMES S6 TIMES S7 TIMES S8 WHERE
S6.cnt.year = 1999 AND
S5.cnt.age-group = S7.cnt.age-group AND
S5.cnt.age-group ='19-20'AND
S6.cnt.year = S8.cnt.year
```

given as input to the algorithm. Here, $S5$, $S6$, $S7$, and $S8$ represent the SP-relations that describe the distribution for the welfare-to-work client's characteristics - Aptitude, Goals, Confidence, and Work-history respectively.

The SP-Algebra query tree constructed by the algorithm is shown in Figure 7. The SPOQL query is evaluated for its syntactic and semantic validity. According to the build-path algorithm, the selection operation on context is performed on SP-relations $S6$ and $S5$ resulting in SP-relations $S6'$ and $S5'$ respectively. The projection operation on context is performed on SP-relation $S7$ resulting in SP-relation $S7'$. According to the build-subtree algorithm, SP-relations $S5'$ and $S7'$ are combined by applying the respective join condition to the cartesian product resulting in SP-relation $S5''$. Similarly SP-relations $S6'$ and $S8$ are combined resulting in SP-relation $S6''$. The resulting SP-relation S''' is the cartesian product of SP-relations $S5''$ and $S6''$.

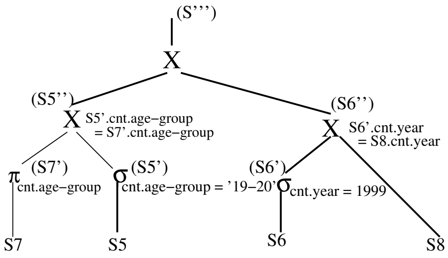

Figure 7: The query tree for SPOQL query Q''.

3.3 SPOQL Implementation

At present, SPOQL is incorporated into the SPDBMS architecture (see Figure 3) by means of the implementation of the translation algorithm described in Figure 6. This algorithm translates SPOQL queries into equivalent SP-Algebra queries that are parsed by the original parser. In addition, to support full functionality of SPOQL as described in this paper, we have added the implementations of the mix operation and of the mix/cartesian product/join with join conditions operations to the SP-Algebra implementation within SPDBMS. New performance tests are currently underway, and future work on the SPDBMS includes a cost-based query optimizer.

4. RELATED WORK

There has been considerable research done in management of probabilistic data. The early relational models, used for storage and processing of probabilistic data [12, 2, 6, 1] have been replaced by the object-oriented [11, 8] and semistructured models [10, 14, 4, 20]. The work of Cavallo and Pittarelli [2] extended the relational model to represent uncertainty due to ambiguity using the well-known probability calculus. A probability measure is assigned to every tuple in a relation. The probability measure indicates the joint probability of all the attribute values in a tuple. Barbara et al. [1] proposed an extension of the relational model using probability theory by adopting a non-lNF probabilistic data model. They redefined the project, select, and join operations using semantics of probability theory. They also introduced a new set of operators to illustrate the various possibilities. Dey and Sarkar [6] provided a probabilistic database framework with relations abiding by first normal form (1NF). The probability measures, assigned to tuples, represented the joint probability distribution of all the non-key attributes in the relation. They provided a closed form query algebra and introduced the conditionalization operation in the context of a probabilistic model. They also proposed a non-procedural probabilistic query language called PSQL [7] as an extension of the SQL language. Lakshmanan et al. [12] proposed axioms characterizing reasonable probabilistic conjunction and disjunction strategies. They first implemented a relational probabilistic database system called ProbView. Ross et al. [18] extended the framework of ProbView to perform aggregate computations over probabilistic data. Cheng et al. [16] provided an uncertainty model, as an extension to the relational model, for handling constantly evolving sensor data. Each tuple in their database is similar to the framework in [6] where probability is attached to a tuple. But instead of having a point probability, they associate an entire distribution. Each tuple in their database is analogous to a non-conditional SPO object in our model. Conditional and joint probabilities are not represented in their framework. The U-DBMS [17] is an implementation of their uncertainty framework.

The work on SPDBMS [20] combines and extends the ideas[5] contained in these papers and applies them to an SPO model. The data stored in the SPDBMS does not conform to a rigid schema.

There are two approaches to semistructured probabilistic data management that are closely related to SPDBMS: the ProTDB [14] and the PXml [10] frameworks [20]. Nierman, et al. [14] extended the XML data model by associating a probability to each element with the modification of regular non-probabilistic DTDs. In ProTDB, independent probabilities are attached to each individual child of an object. The probabilities in an ancestor-descendant chain are related probabilistically, resulting in conditional probabilities in the document. Some drawbacks of ProTDB are overcome by the PXml framework proposed by Hung, Getoor and Subrahmanian [10]. PXml supports arbitrary distributions over sets of children and allows arbitrary acyclic dependency models. They also proved that for any query in their model there is a mapping to an equivalent query in the bayesian network. They also proposed a probabilistic interval XML data model, PIXml [9]. But the PXml and PIXml models do not provide a convenient way to represent joint probability distributions.

5. CONCLUSIONS

The SPO model for management of uncertain data in databases provides flexible means for storing and manipulating large collections of probability distributions. The original query language, SP-Algebra, incorporates all major database operations, and introduces some operations, such as conditionalization, specific to probabilistic database management. The traditional limitations of SP-Algebra — the functional syntax, and unfamiliarity to application programmers, have been alleviated by SPOQL, the structured query language for the SPO model, which provides familiar SQL-like declarative syntax that is easier for the programmers to use. At the same time, as we have shown in this paper, parsing SPOQL queries is a more involved task than parsing SQL queries, due to the fact that important query equivalences do not hold in SP-Algebra, making some potential invariant query translations incompatible. In this paper we have discussed our approach to parsing SPOQL queries, that can be characterized as *eager evaluation* — all operations are applied (in the defined order of precedence) as soon as they can be executed. The new SPDBMS server is in the process of replacing the old one and applications using SPDBMS as the back end for storage of probabilistic data are developed for the Welfare-to-Work modelling project described in part in Section 2.1.

Our ongoing and future work on SPO model is two-fold. First, we are working on a cost-based query optimizer for SPDBMS. The second, current version of SPDBMS had been implemented by special-purpose shredding of XML representing SPOs into relational tables and translating SP-Algebra operations into sequences of SQL statements. We plan on building a new version of SPDBMS on top of a native XML DBMS translating SP-Algebra into XQuery.

[5]Aggregate computations are not available currently in the SPDBMS.

6. REFERENCES

[1] D. Barbará, H. Garcia-Molina, and D. Porter. The management of probabilistic data. *IEEE Trans. on Knowledge and Data Engineering*, 4:487–502, 1992.

[2] R. Cavallo and M. Pittarelli. The theory of probabilistic databases. In *Proc. VLDB'87*, pages 71–81, 1987.

[3] Alex Dekhtyar, Raphael Finkel, Judy Goldsmith, Beth Goldstein, and Cynthia Isenhour. Adaptive decision support for planning under hard and soft constraints. In *Proceedings of the AAAI Spring Symposium on Challenges to Decision Support in a Changing World*, pages 17–22, Stanford University, Palo Alto, CA, 2005.

[4] Alex Dekhtyar, Judy Goldsmith, and Sean R. Hawkes. Semistructured probabilistic databases. In *Proc. Statistical and Scientific Database Management Systems*, pages 36–45, 2001.

[5] Alex Dekhtyar, Judy Goldsmith, Huaizhi Li, and Brett Young. Bayesian advisor project i: Modeling academic advising. Technical Report TR 323-01, University of Kentucky, March 2001.

[6] D. Dey and S. Sarkar. A probabilistic relational model and algebra. *ACM Transactions on Database Systems*, 21(3):339–369, 1996.

[7] Debabrata Dey and Sumit Sarkar. Psql: a query language for probabilistic relational data. *Data Knowl. Eng.*, 28(1):107–120, 1998.

[8] T. Eiter, J. Lu, T. Lukasiewicz, and V.S. Subrahmanian. Probabilistic object bases. *ACM Transactions on Database Systems*, 26(3):264–312, 2001.

[9] Edward Hung, Lise Getoor, and V. S. Subrahmanian. Probabilistic interval xml. In *ICDT*, pages 361–377, 2003.

[10] Edward Hung, Lise Getoor, and V. S. Subrahmanian. Pxml: A probabilistic semistructured data model and algebra. In *ICDE*, 2003.

[11] E. Kornatzky and S.E. Shimony. A probabilistic object data model. *Data and Knowledge Engineering*, 12:143–166, 1994.

[12] 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.

[13] Kevin Krol Mathias, Cynthia Isenhour, Alex Dekhtyar, Judy Goldsmith, and Beth Goldstein. Eliciting and combining influence diagrams: Tying many bowties together. Technical Report TR 453-06, University of Kentucky, March 2006.

[14] Andrew Nierman and H. V. Jagadish. ProTDB: Probabilistic data in XML. In *Proceedings of the 28th VLDB Conference*, 2002.

[15] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 1988.

[16] S. Prabhakar R. Cheng, D. Kalashniov. Evaluating probabilistic queries over imprecise data. In *Proceedings of the ACM SIGMOD*, 2003.

[17] S. Prabhakar R. Cheng, Sarvjeet Singh. U-dbms: A database system for managing consantly-evolving data. In *Proceedings of the 31st VLDB conference*, 2005.

[18] Robert Ross, V. S. Subrahmanian, and John Grant. Aggregate operators in probabilistic databases. *J. ACM*, 52(1):54–101, 2005.

[19] Wenzhong Zhao. *Probabilistic databases and their applications*. PhD thesis, University of Kentucky, 2004.

[20] Wenzhong Zhao, Alex Dekhtyar, and Judy Goldsmith. A framework for management of semistructured probabilistic data. *Journal of Intelligent Information Systems*, 25(3):293–332, 2004.

[21] Wenzhong Zhao, Alex Dekhtyar, Judy Goldsmith, Erik Jessup, and Jiangyu Li. Building bayes nets with semistructured probabilistic dbms. *GI EMISA Forum*, 24(1):29–30, 2004.