Probabilistic Object Bases

THOMAS EITER
Technische Universität Wien
JAMES J. LU
Bucknell University
THOMAS LUKASIEWICZ
Technische Universität Wien
and
V. S. SUBRAHMANIAN
University of Maryland

Although there are many applications where an object-oriented data model is a good way of representing and querying data, current object database systems are unable to handle objects whose attributes are uncertain. In this article, we extend previous work by Kornatzky and Shimony to develop an algebra to handle object bases with uncertainty. We propose concepts of consistency for such object bases, together with an NP-completeness result, and classes of probabilistic object bases for which consistency is polynomially checkable. In addition, as certain operations involve conjunctions and disjunctions of events, and as the probability of conjunctive and disjunctive events depends both on the probabilities of the primitive events involved as well as on what is known (if anything) about the relationship between the events, we show how all our algebraic operations may be performed under arbitrary probabilistic conjunction and disjunction strategies. We also develop a host of equivalence results in our algebra, which may be used as rewrite rules for query optimization. Last but not least, we have developed a prototype probabilistic object base server on top of ObjectStore. We describe experiments to assess the efficiency of different possible rewrite rules.

Categories and Subject Descriptors: H.2.1 [**Database Management**]: Logical Design—*data models*; H.2.3 [**Database Management**]: Languages—*query languages*; H.2.4 [**Database Management**]: Systems—*object-oriented databases*; I.2.4 [**Artificial Intelligence**]: Knowledge Representation Formalisms and Methods

General Terms: Algorithms, Languages, Performance, Theory

This work was supported by a TASC/DARPA grant J09301S98061, by the Army Research Laboratory under contract number DAAL01-97-K0135, by an NSF Young Investigator award IRI-93-57756, by an NSF grant CCR9731893, by a DFG grant, and by the Austrian Science Fund under project N Z29-INF.

Authors' addresses: T. Eiter and T. Lukasiewicz, Institut und Ludwig Wittgenstein Labor für Informationssysteme, Technische Universität Wien, Favoritenstraße 9-11, A-1040 Wien, Austria; email: {eiter,lukasiew}@kr.tuwien.ac.at; J. J. Lu, Department of Computer Science, Bucknell University, Lewisburg, PA 17837; email: jameslu@bucknell.edu; V. S. Subrahmanian, Institute for Advanced Computer Studies, Institute for Systems Research and Department of Computer Science, University of Maryland, College Park, MD 20742; email: vs@cs.umd.edu.

Additional Key Words and Phrases: Consistency, object-oriented database, probabilistic object algebra, probabilistic object base, probability, query language, query optimization

1. INTRODUCTION

The concept of an *object base* is gaining numerous adherents because it allows data to be organized in an application-specific manner for scalability, while still supporting a common query language. However, there are many applications where *probabilistic* data need to be stored. For instance, image interpretation programs are uncertain in their identification of features in images and such image databases are typically stored using object databases [Grosky et al. 1997]. Similarly, an application tracking a set of mobile objects using an object database may only know that an object is at one of a given set of points right now, but the precise location may be unknown. Likewise, an application that forecasts stock movements or the weather needs to represent uncertainty in the forecast. When the application data (stocks, weather) are in an object repository, methods to represent uncertain aspects of these objects need to be developed. In short, the ability to represent probabilistic information in an object base, and to manipulate such "probabilistic object bases" (POBs) efficiently is important for a variety of applications.

To date, there has been only one significant attempt in the database community to merge probability models with object bases, namely, that by Kornatzky and Shimony [1994], who proposed a probabilistic object calculus. Building upon their influential work, we make the following contributions.

1. First and foremost, we propose a notion of a probabilistic schema and formally define a logical model theory for it. We define what *consistent schemas* are and prove that consistency checking is NP-complete. We identify special classes of schemas for which consistency may be polynomially checked. Previous work on probabilistic object bases had no associated concept of consistency.
2. We then propose an algebra for probabilistic object bases in which the classical relational algebra operators are extended to apply to probabilistic object bases. It is well known [Lakshmanan et al. 1997] that the probabilities of conjunctive and disjunctive events are computed in different ways depending upon the dependencies between the events involved. Our algebraic operators are parameterized by the user's knowledge (or lack thereof) of such dependencies; hence, the user can ask queries of the form, "Find the join of ... under ignorance," which describes a join assuming no knowledge about the dependencies between the events involved. Previous work on probabilistic object bases assumed that all events involved were independent. To our knowledge, this is the first (extension of the) relational algebra for POBs.
3. We then prove a host of equivalence results in our algebra. These equivalence results may be used as the set of rewrite rules that a database query optimizer uses for query rewriting.

4. We have implemented a distributed POB system in C++ on top of Object Store. This implementation allowed us to conduct experiments across the network to evaluate the performance of our system and also to see how to rewrite queries.

This article is structured as follows. In the next section, we consider a motivating database application. Section 3 describes the architecture of a POB system. After some basic definitions of probability concepts in Section 4, we develop our POB model in Sections 5 and 6. A query algebra is then presented in Section 7, and equivalence results in this algebra are derived in Section 8. We report on an implementation of POBs in Section 9, and discuss related work in Section 10. Detailed proofs of all results may be found in Eiter et al. [1999].

2. A MOTIVATING EXAMPLE

Consider the task of building an extensive database describing the types of vegetation found in the Amazon rainforest. The creation of such a database is a formidable task. Individuals need to exhaustively examine the vegetables, herbs, and other kinds of plants growing in these forests, and provide information describing soil conditions, climactic conditions, and so on.

When describing the plants growing in such rainforests, there are several possible causes of *uncertainty*. First, some plant species may not be uniquely identifiable by the surveyor in the field. He may classify a particular herb as either being Silver Thyme or French Thyme (two different species of thyme), without being able to specify exactly to which species the plant in question belongs. By the same token, if he were slightly more expert, he might be able to say that he is not sure whether the herb is Silver Thyme or French Thyme, but he rates the probability that it is Silver Thyme twice as high as that it is French Thyme.

Figure 1 shows a very simple class hierarchy that describes plants as either being perennials or annuals, and being vegetables, herbs, or flowers. Clearly, the classes perennials and annuals are disjoint (i.e., a plant cannot be both an annual and a perennial), as are the classes vegetables, herbs, and flowers. Mutually disjoint classes are connected by a "**d**" in Figure 1. Note that we can certainly have plants that are annuals and herbs (e.g., Basil). For now, the numbers labeling edges in Figure 1 may be ignored. They are revisited later.

In the rest of this article, we repeatedly use this example to illustrate our definitions. By the end of this article, we describe how to build and query a POB that captures the Plant Database of this example as a special case.

3. ARCHITECTURE OF A PROBABILISTIC OBJECT BASE

In this section, we describe the overall architecture of a probabilistic object base system. Figure 2 presents an architecture for query processing in probabilistic object bases. The architecture consists of the following components.

1. The user expresses queries through a graphical user interface that generates as output a query in a declarative *probabilistic object calculus*

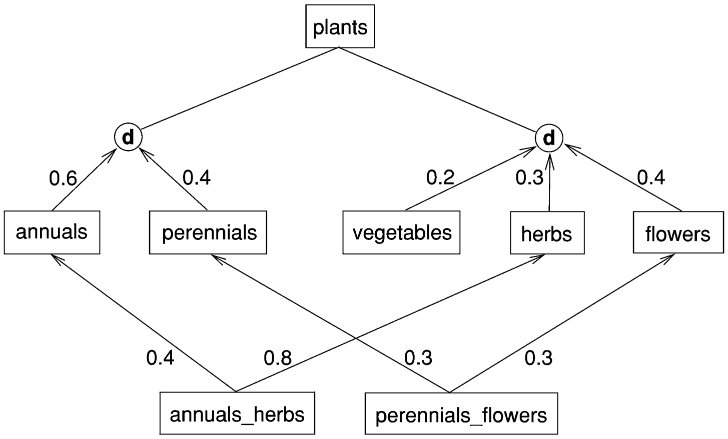


Fig. 1. Plant identification example.

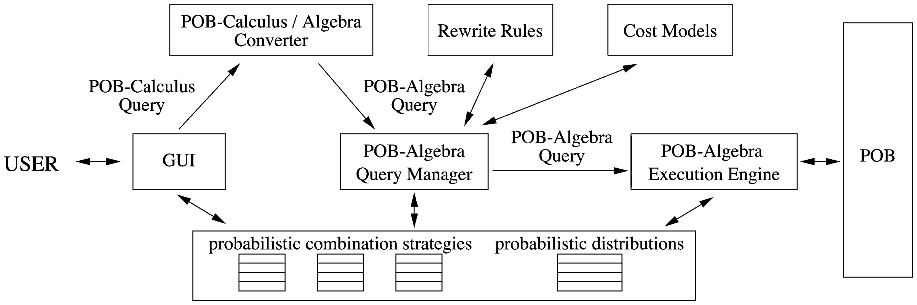

POB-Calculus / Algebra Converter
Rewrite Rules
Cost Models
POB-Calculus Query
POB-Algebra Query
USER
GUI
POB-Algebra Query Manager
POB-Algebra Query
POB-Algebra Execution Engine
POB
probabilistic combination strategies
probabilistic distributions

Fig. 2. Architecture of POB system.

(POB-calculus). Note that queries in this calculus are declarative queries. A pioneering attempt at such a calculus is that of Kornatzky and Shimony [1994].

2. The calculus query generated will be fed into a *Converter* which converts POB-calculus queries into queries in a *probabilistic object algebra* (POB algebra).
3. The algebraic query generated by the converter will be fed into a *Query Manager*, which will take as input a set of *rewrite rules* (reflecting equivalences between different queries in the POB algebra) and a set of *cost models* to perform a query optimization. Given a set of rewrite rules and a set of cost models, the task of finding a rewriting of a query that has minimal expected cost (according to the cost models) is well studied, and good commercial implementations of such code exist (e.g., Graefe's [1995] CASCADES system is presently being used by Microsoft).
4. The "optimized" algebra query thus produced will be physically executed on the probabilistic object base.
5. All the components above use *libraries* consisting of: (i) a set of probabilistic *conjunction*, *disjunction*, *and difference* strategies that allow the user to

express her knowledge of the dependencies between events—this is used in query formulation, query optimization, cost evaluation, and query execution; and (ii) a set of distribution functions that allow a user to specify how probabilities are distributed over a space of possible values for an unknown attribute.

Giving a detailed description of all these components is clearly beyond the scope of a single paper. In previous work, Kornatzky and Shimony [1994] developed a probabilistic object calculus. In this article, we expand the concept of a probabilistic object base used by them and formally define a POB algebra and prove a host of query equivalence results. We report on a prototype implementation of the POB algebra and describe experimental results; given a query equivalence $q_1 = q_2$, these experimental results identify when a query of the form q_1 should be rewritten to a query of the form q_2 and vice versa. To our knowledge, this article is the first to propose a probabilistic object algebra, the first to present results on query equivalences in such an algebra, and the first to implement such an algebra on top of a commercial object database system.

4. BASIC PROBABILITY DEFINITIONS

In this section, we present some basic definitions used to set up a probabilistic extension of object bases. The probabilistic concepts are divided into two parts: probabilistic combination strategies and distribution functions.

4.1 Probabilistic Combination Strategies

Suppose we know the probabilities of events e_1 and e_2. For example, e_1 may be the event, "The photographed plant p_1 (in image I) is French Thyme." Similarly, e_2 may be the event, "The photographed plant p_2 (in image I) is Mint." Assume now that we are interested in the probability of the complex event $(e_1 \wedge e_2)$. The probability of $(e_1 \wedge e_2)$ is computed in different ways based upon our knowledge of the dependencies between e_1 and e_2.

- —*e_1 and e_2 are independent.* This may occur if we know that the plants p_1 and p_2 are growing in the area independently of each other. In this case, $\mathbf{P}(e_1 \wedge e_2) = \mathbf{P}(e_1) \cdot \mathbf{P}(e_2)$ (i.e., the probability of $(e_1 \wedge e_2)$ is the product of the probabilities of e_1 and e_2).
- —*e_1 and e_2 are mutually exclusive.* Suppose, for example, we know that p_1 and p_2 are the same plant. Since the events e_1 and e_2 are mutually exclusive, we can immediately say that $\mathbf{P}(e_1 \wedge e_2) = 0$.
- —*We are ignorant of the relationship between e_1 and e_2.* This case occurs when we do not know anything about the relationship between the plants p_1 and p_2 growing in the same area. As shown by Boole [1854], the best we can say in this case about the probability of $(e_1 \wedge e_2)$ is that it lies in the interval $[\max(0, \mathbf{P}(e_1) + \mathbf{P}(e_2) - 1), \min(\mathbf{P}(e_1), \mathbf{P}(e_2))]$.

Thus, the probability of $(e_1 \wedge e_2)$ depends not only on the probabilities of e_1 and e_2, but also on the relationship between the events e_1 and e_2. A similar situation applies when we consider complex events such as $(e_1 \vee e_2)$. The above are only three examples of different ways of evaluating probabilities of complex events. In general, depending on exactly what is known about the dependencies between the events involved, there is a whole plethora of such probability computations.

In our framework, we use probability intervals instead of point probabilities for two reasons: in many applications, the probability of an event is often not precisely given; and as already shown by Boole [1854], when we do not know the dependencies between two events, all that we can say about the probability of the conjunction/disjunction of two events is that it belongs to an interval.

Definition 4.1 (*Consistent Probabilistic Intervals for Two Events*). Suppose e_1 and e_2 have probabilities in the intervals $I_1 = [L_1, U_1]$ and $I_2 = [L_2, U_2]$, respectively. Such an assignment of probabilistic intervals is called *consistent* if and only if $L_1 \leq U_1$, $L_2 \leq U_2$, and the following conditions hold.

- If $(e_1 \wedge e_2)$ is contradictory,[1] then $L_1 + L_2 \leq 1$.
- If $(e_1 \wedge \neg e_2)$ is contradictory, then $L_1 \leq U_2$.
- If $(\neg e_1 \wedge e_2)$ is contradictory, then $L_2 \leq U_1$.
- If $(\neg e_1 \wedge \neg e_2)$ is contradictory, then $U_1 + U_2 \geq 1$.

In the following, all assignments of probabilistic intervals are implicitly assumed to be consistent unless stated otherwise. Suppose $I_1 = [L_1, U_1]$ and $I_2 = [L_2, U_2]$. We use $I_1 \leq I_2$ as an abbreviation for $L_1 \leq L_2$ and $U_1 \leq U_2$, and $I_1 \subseteq I_2$ to denote that I_1 is contained in I_2; that is, $L_2 \leq L_1$ and $U_1 \leq U_2$.

As many dependencies between events cannot be automatically inferred, it is imperative that the user be able to specify, in his query, what knowledge he has about such relationships. To facilitate this, Lakshmanan et al. [1997] have introduced generic probabilistic conjunction and disjunction strategies. Any function that satisfies the axioms listed in Table I is called a probabilistic conjunction or disjunction strategy, respectively. (Given two events e_1 and e_2 with probabilities in the intervals $I_1 = [L_1, U_1]$ and $I_2 = [L_2, U_2]$, respectively, the notations "$I = I_1 \otimes I_2$" and "$I = I_1 \oplus I_2$" are shorthand for "$(e_1 \wedge e_2, I) = (e_1, I_1) \otimes (e_2, I_2)$" and "$(e_1 \vee e_2, I) = (e_1, I_1) \oplus (e_2, I_2)$", respectively.)

While the notion of conjunction and disjunction strategies are recapitulated from Lakshmanan et al. [1997], the concept of difference strategies below is new.

Definition 4.2 (*Probabilistic Difference Strategy*). Suppose e_1 and e_2 have probabilities in the intervals $I_1 = [L_1, U_1]$ and $I_2 = [L_2, U_2]$, respectively. A *probabilistic difference strategy* is a binary operation $\ominus$ that uses this information to compute a probabilistic interval $I = [L, U]$ for the event $(e_1 \wedge \neg e_2)$. When the events involved are clear from context, we use "$I = I_1 \ominus I_2$" to denote "$(e_1 \wedge \neg e_2, I) = (e_1, I_1) \ominus (e_2, I_2)$". Difference strategies satisfy the following postulates.

Bottomline: $(I_1 \ominus I_2) \leq [\min(L_1, 1 - U_2), \min(U_1, 1 - L_2)]$.
Ignorance: $(I_1 \ominus I_2) \subseteq [\max(0, L_1 - U_2), \min(U_1, 1 - L_2)]$.

[1] Contradictory here merely means "inconsistent in classical propositional logic."

Table I. Axioms for Conjunction and Disjunction Strategies

Axiom Name	Conjunction Strategy
Bottomline	$(I_1 \otimes I_2) \leq [\min(L_1, L_2), \min(U_1, U_2)]$
Ignorance	$(I_1 \otimes I_2) \subseteq [\max(0, L_1 + L_2 - 1), \min(U_1, U_2)]$
Identity*	$(I_1 \otimes [1, 1]) = I_1$
Commutativity	$(I_1 \otimes I_2) = (I_2 \otimes I_1)$
Associativity	$((I_1 \otimes I_2) \otimes I_3) = (I_1 \otimes (I_2 \otimes I_3))$
Monotonicity	$(I_1 \otimes I_2) \leq (I_2 \otimes I_3)$ if $I_2 \leq I_3$

Axiom Name	Disjunction Strategy
Bottomline	$(I_1 \oplus I_2) \geq [\max(L_1, L_2), \max(U_1, U_2)]$
Ignorance	$(I_1 \oplus I_2) \subseteq [\max(L_1, L_2), \min(1, U_1 + U_2)]$
Identity*	$(I_1 \oplus [0, 0]) = I_1$
Commutativity	$(I_1 \oplus I_2) = (I_2 \oplus I_1)$
Associativity	$((I_1 \oplus I_2) \oplus I_3) = (I_1 \oplus (I_2 \oplus I_3))$
Monotonicity	$(I_1 \oplus I_2) \leq (I_1 \oplus I_3)$ if $I_2 \leq I_3$

*The Identity-axioms for probabilistic conjunction (resp., disjunction) strategies assume that $e_1 \wedge e_2$ and $\neg e_1 \wedge e_2$ (resp., $\neg e_1 \wedge \neg e_2$ and $e_1 \wedge \neg e_2$) are not contradictory.

Table II. Examples of Probabilistic Combination Strategies

Strategy	Operators
Ignorance	$([L_1, U_1] \otimes_{ig} [L_2, U_2]) \equiv [\max(0, L_1 + L_2 - 1), \min(U_1, U_2)]$
	$([L_1, U_1] \oplus_{ig} [L_2, U_2]) \equiv [\max(L_1, L_2), \min(1, U_1 + U_2)]$
	$([L_1, U_1] \ominus_{ig} [L_2, U_2]) \equiv [\max(0, L_1 - U_2), \min(U_1, 1 - L_2)]$
Independence	$([L_1, U_1] \otimes_{in} [L_2, U_2]) \equiv [L_1 \cdot L_2, U_1 \cdot U_2]$
	$([L_1, U_1] \oplus_{in} [L_2, U_2]) \equiv [L_1 + L_2 - (L_1 \cdot L_2), U_1 + U_2 - (U_1 \cdot U_2)]$
	$([L_1, U_1] \ominus_{in} [L_2, U_2]) \equiv [L_1 \cdot (1 - U_2), U_1 \cdot (1 - L_2)]$
Positive Correlation (when e_1 implies e_2, or e_2 implies e_1)	$([L_1, U_1] \otimes_{pc} [L_2, U_2]) \equiv [\min(L_1, L_2), \min(U_1, U_2)]$
	$([L_1, U_1] \oplus_{pc} [L_2, U_2]) \equiv [\max(L_1, L_2), \max(U_1, U_2)]$
	$([L_1, U_1] \ominus_{pc} [L_2, U_2]) \equiv [\max(0, L_1 - U_2), \max(0, U_1 - L_2)]$
Mutual Exclusion (when e_1 and e_2 are mutually exclusive)	$([L_1, U_1] \otimes_{me} [L_2, U_2]) \equiv [0, 0]$
	$([L_1, U_1] \oplus_{me} [L_2, U_2]) \equiv [\min(1, L_1 + L_2), \min(1, U_1 + U_2)]$
	$([L_1, U_1] \ominus_{me} [L_2, U_2]) \equiv [L_1, \min(U_1, 1 - L_2)]$

Identity: If $(\neg e_1 \wedge \neg e_2)$ and $(e_1 \wedge \neg e_2)$ are not contradictory,[2] then $(I_1 \ominus [0, 0]) = I_1$.

Examples of probabilistic conjunction, disjunction, and difference strategies are given in Table II. Note that we do not assume any postulates that relate probabilistic conjunction, disjunction, and difference strategies to each other (e.g., postulates that express the distributivity of conjunction and disjunction strategies). Readers may make such assumptions if they wish; however, the results of this article stand even if these assumptions are not made.

[2] Note that the precondition is necessary. For example, if $I_1 = [0, 1]$, and $\neg e_1 \wedge \neg e_2$ (resp., $e_1 \wedge \neg e_2$) is contradictory, then $(I_1 \ominus [0, 0]) = [1, 1] \neq I_1$ (resp., $(I_1 \ominus [0, 0]) = [0, 0] \neq I_1$) by the laws of probability.

4.2 Probability Distribution Functions

Probability distribution functions assign probabilities to elementary events in a coherent way. For example, if we are told that plant p_1 is currently at one of the locations a, b, c with probability 60 to 70%, then a distribution function allows us to assign parts of this probability mass to the events "plant p_1 is at location a," "plant p_1 is at location b," and "plant p_1 is at location c."

Definition 4.3 (*Distribution Function*). Let X be a finite set. A *(probability) distribution function α over X* is a mapping from X to the real interval $[0, 1]$ such that $\sum_{x\in X} \alpha(x) \leq 1$.

We do not require that $\sum_{x\in X} \alpha(x) = 1$ holds; a distribution function α with this property is said to be *complete*. The above definition allows us to assign probabilities to a subset $X \subseteq Y$ of elements, leaving the probabilities of the other elements open.

An important distribution function which we often encounter in practice is the *uniform distribution*. For a finite set X, it is defined by $\mathrm{u}_X(x) = 1/|X|$ for all $x \in X$. We abbreviate u_X by u, whenever X is clear from the context. Many other distribution functions are conceivable; we do not study this further here.

Definition 4.4 (*Probabilistic Triple*). A *probabilistic triple* $\langle X, \alpha, \beta\rangle$ consists of a finite set X, a distribution function α over X, and a function $\beta : X \to [0, 1]$ mapping X to the real interval $[0, 1]$ such that (i) $\alpha(x) \leq \beta(x)$ for all $x \in X$ and (ii) $\sum_{x\in X} \beta(x) \geq 1$ hold.

Informally, a probabilistic triple assigns to each element $x \in X$ a probability interval $[\alpha(x), \beta(x)]$. This assignment is consistent in the sense that we can assign each element in X a probability $p(x)$ from $[\alpha(x), \beta(x)]$ such that the sum of all $p(x)$ adds up to 1. In the following, we implicitly assume that all probabilistic triples $\langle X, \alpha, \beta\rangle$ are *tight*; that is, for each $x \in X$, the bounds $\alpha(x)$ and $\beta(x)$ are the minimum and maximum, respectively, of $p(x)$ subject to all complete distribution functions p over X such that $p(x') \in [\alpha(x'), \beta(x')]$ for all $x' \in X$. Thus, any probabilistic triple that is entered by a user or computed by our algebraic operations is implicitly assumed to be transformed into its tight equivalent (which can easily be done).

5. TYPES AND PROBABILISTIC OBJECT BASE SCHEMAS

In this section, we provide some basic definitions underlying a probabilistic object base. We first consider types and values, and then the schema of a POB. The notion of POB schema is more complex than in the context of relational databases, and may lead to inconsistent specifications; we present efficient algorithms for checking schema consistency.

5.1 Types and Values

We start with the definition of types.

Definition 5.1 (*Types*). Let $\mathcal{A}$ be a set of *attributes* and let $\mathcal{T}$ be a set of *atomic types*. We define *types* inductively as follows.

—Every atomic type from $\mathcal{T}$ is a type.

—If τ is a type, then $\{\tau\}$ is a type, which is called the *set type* of τ.

—If $A_1, \ldots, A_k$ are pairwise different attributes from $\mathcal{A}$ and $\tau_1, \ldots, \tau_k$ are types, then $[A_1\colon \tau_1, \ldots, A_k\colon \tau_k]$ is a type. This type is called a *tuple type* over the set of attributes $\{A_1, \ldots, A_k\}$. Given such a type $\tau = [A_1\colon \tau_1, \ldots, A_k\colon \tau_k]$, we use $\tau.A_i$ to denote τ_i. We call $A_1, \ldots, A_k$ the *top-level attributes* of τ.

Example 5.1 (*Plant Example*: *Types*). In the Plant Example, some atomic types from $\mathcal{T}$ are integer, real, string, soiltype, and suntype. The attributes soil, sun (sun-exposure), and rain (daily water) describe conditions needed for a plant to grow. Some other attributes are pname, size, height, and width. Some (nonatomic) types include: {soiltype}, [soil: {soiltype}, sun: suntype, rain: integer] and [pname: string, size: [height: integer, width: integer]].

Definition 5.2 (*Values*). Every atomic type $\tau \in \mathcal{T}$ has an associated *domain* $\mathrm{dom}(\tau)$. We define *values* by induction as follows.

—For all atomic types $\tau \in \mathcal{T}$, every $v \in \mathrm{dom}(\tau)$ is a value of type τ.

—If $v_1, \ldots, v_k$ are values of type τ, then $\{v_1, \ldots, v_k\}$ is a value of type $\{\tau\}$.

—If $A_1, \ldots, A_k$ are pairwise different attributes from $\mathcal{A}$ and $v_1, \ldots, v_k$ are values of types $\tau_1, \ldots, \tau_k$, then $[A_1\colon v_1, \ldots, A_k\colon v_k]$ is a value of type $[A_1\colon \tau_1, \ldots, A_k\colon \tau_k]$.

Example 5.2 (*Plant Example*: *Values*). Let us return to the types of Example 5.1. We assign the usual domains to integer, real, and string. Let soiltype and suntype be enumerated types having the domains {loamy, swampy, sandy} and {mild, medium, heavy}, respectively. The value sets associated with the types of Example 5.1 are as follows.

—soiltype: Any element of {loamy, swampy, sandy} is a value of soiltype. For example, loamy is a value of soiltype. When associated with a particular plant, this value might say that the plant needs loamy soil to flourish.

—{soiltype}: Any set of values of soiltype is a value of this type. For example, if a particular plant can grow well in either loamy or swampy soil, then {loamy, swampy} is an appropriate value of this type that can be associated with this plant.

—[soil: {soiltype}, sun: suntype, rain: integer]: Any triple (v_1, v_2, v_3) is a value of this type, where v_1 is a *set* of values of soiltype, v_2 is a value of suntype, and v_3 is a value of integer. For example, ({loamy, swampy}, mild, 3) is a value of this type. It says that the plant needs either loamy or swampy soil, mild sun, and 3 units of water per day to flourish.

Definition 5.3 (*Probabilistic Tuple Values*). If $A_1, \ldots, A_k$ are pairwise distinct attributes from $\mathcal{A}$ and $(V_1, \alpha_1, \beta_1), \ldots, (V_k, \alpha_k, \beta_k)$ are probabilistic triples where $V_1, \ldots, V_k$ are *sets* of values of types $\tau_1, \ldots, \tau_k$, then the expression $[A_1\colon (V_1, \alpha_1, \beta_1), \ldots, A_k\colon (V_k, \alpha_k, \beta_k)]$ is a *probabilistic tuple value* of type $[A_1\colon \tau_1, \ldots, A_k\colon \tau_k]$ over the set of attributes $\{A_1, \ldots, A_k\}$. For probabilistic tuple values $ptv = [A_1\colon (V_1, \alpha_1, \beta_1), \ldots, A_k\colon (V_k, \alpha_k, \beta_k)]$, we use $ptv.A_i$ to denote (V_i, α_i, β_i).

Note that the order of the $A_i\colon(V_i, \alpha_i, \beta_i)$s in a probabilistic tuple value $ptv = [A_1\colon(V_1, \alpha_1, \beta_1), \ldots, A_k\colon(V_k, \alpha_k, \beta_k)]$ is not important.

Example 5.3 (*Plant Example*: *Probabilistic Tuple Values*). Assume we know that the soil type of a wild forest plant is loamy (presumably, as we can see, the plant is flourishing in the place in which it is currently growing). Moreover, we are sure that this plant is Thyme, but unsure whether it is French Thyme (french), Silver Thyme (silver), or Wooly Thyme (wooly). If we are sure with 20 to 60% probability each that it is French Thyme, Silver Thyme, or Wooly Thyme, then we may encode this knowledge via the probabilistic tuple value of type [soil: soiltype, classification: string] over the set of attributes {soil, classification}:

$$[\text{soil}\colon \langle\{\text{loamy}\}, \text{u}, \text{u}\rangle,\ \text{classification}\colon \langle\{\text{french}, \text{silver}, \text{wooly}\}, 0.6\,\text{u}, 1.8\,\text{u}\rangle]\,.$$

Note that the expressions “0.6 u” and “1.8 u” denote the distribution function α and the function β, respectively, that are defined by $\alpha(x) = 0.6 \cdot 1/3$ and $\beta(x) = 1.8 \cdot 1/3$ for all x from {french, silver, wooly}.

In the above definition, a probabilistic triple (V_i, α_i, β_i) may only assign a probability interval to some values v (viz. those in V_i) for the attribute A_i. Nothing is stated for the (possibly infinitely many) other values that A_i could have according to its type τ_i. We must find a clean and appealing way in which such incomplete knowledge about the probability assignment is handled.

As in relational databases, we adopt a *closed world assumption* (CWA): we assume that every value $v \in \text{dom}(\tau_i) - V_i$ has probability 0; that is, it is implicitly assigned the probability interval $[0, 0]$. Under this convention, “consistency” (which we define formally later) of the probability information given by (V_i, α_i, β_i) is preserved in the larger context of $\text{dom}(\tau_i)$: there exists a probability function p over $\text{dom}(\tau_i)$ that is compatible with (V_i, α_i, β_i) such that all $p(v)$ with $v \in \text{dom}(\tau_i)$ sum up to 1. The probabilistic object base algebra defined in Section 7 is based on this CWA. Notice that an open world view is still possible for particular values. We may, for instance, add v to V and set $\alpha(v) = 0$, $\beta(v) = 1$; this explicitly expresses that the probability of v is unknown.

5.2 Probabilistic Object Base Schema

Informally, a probabilistic object base schema consists of a hierarchy of classes. Membership of an object in an immediate subclass of any class is expressed by a probability value.

Definition 5.4 (*Probabilistic Object Base Schema*). A *probabilistic object base schema* (*POB schema*) is a quintuple $(\mathcal{C}, \tau, \Rightarrow, \text{me}, \wp)$, where:

—*$\mathcal{C}$ is a finite set of* classes. Intuitively, these reflect the classes associated with this probabilistic object base.

—*τ maps each class from $\mathcal{C}$ to a tuple type.* Intuitively, this mapping specifies the data type of each class.

—*$\Rightarrow$ is a binary relation on $\mathcal{C}$ such that $(\mathcal{C}, \Rightarrow)$ is a directed acyclic graph* (*dag*). Intuitively, each node of the directed acyclic graph $(\mathcal{C}, \Rightarrow)$ is a class from $\mathcal{C}$ and each edge $c_1 \Rightarrow c_2$ says that the class c_1 is an *immediate subclass* of c_2.

—me *maps each class* c *to a partition of the set of all immediate subclasses of* c. Intuitively, suppose class c has the five subclasses $c_1, \ldots, c_5$ and suppose $\mathrm{me}(c)$ is given by the partition $\{\{c_1, c_2\}, \{c_3, c_4, c_5\}\}$. Here, $\mathrm{me}(c)$ produces two clusters. An object $o \in c$ can belong to either or both clusters. However, the classes within a cluster are mutually exclusive; that is, o cannot belong to both c_1 and c_2 at the same time.

—$\wp$ *maps each edge in* $(\mathcal{C}, \Rightarrow)$ *to a positive rational*[3] *number in the unit interval* $[0, 1]$ *such that for all classes* c *and all clusters* $\mathcal{P} \in \mathrm{me}(c)$*, it holds that* $\Sigma_{d \in \mathcal{P}} \wp(d, c) \leq 1$. Intuitively, if $c_1 \Rightarrow c_2$, then $\wp(c_1, c_2)$ specifies the conditional probability that an arbitrary object belongs to the subclass c_1 given that it belongs to the superclass c_2. The summation condition says that the sum of the probabilities of edges within a mutually exclusive set of subclasses must sum up to less than or equal to 1.

A *directed path* in the directed acyclic graph $(\mathcal{C}, \Rightarrow)$ is a sequence of classes $c_1, c_2, \ldots, c_k$ such that $c_1 \Rightarrow c_2 \Rightarrow \cdots \Rightarrow c_k$ and $k \geq 1$. We use $\Rightarrow^\star$ to denote the reflexive and transitive closure of $\Rightarrow$. Note that $\Rightarrow^\star$ induces a natural partial order $\leq$ on $\mathcal{C}$ by $c \leq d$ if and only if $c \Rightarrow^\star d$ for all $c, d \in \mathcal{C}$.

We use $\mathrm{S}(c) = \{d \in \mathcal{C} \mid d \Rightarrow c\}$ to denote the set of all *immediate subclasses* of $c \in \mathcal{C}$, and $\mathrm{S}^\star(c) = \{d \in \mathcal{C} \mid d \Rightarrow^* c\}$ to denote the set of *subclasses* of $c \in \mathcal{C}$. A class d is a *subclass* of a partition cluster $\mathcal{P}$ if and only if d is a subclass of some $c \in \mathcal{P}$.

We represent the above structure (excluding the type assignment τ) in a graphical way as shown in Figure 1, where the edges are labeled by conditional probabilities.

Example 5.4 (*Plant Example*: *Probabilistic Object Base Schema*). A POB schema for the Plant Example may consist of the following components.

—$\mathcal{C} = \{$plants, annuals, perennials, vegetables, herbs, flowers, annuals_herbs, perennials_flowers$\}$.

—τ is given by Table III.

—$(\mathcal{C}, \Rightarrow)$ is the graph obtained from Figure 1 by contracting the d-nodes to plants and ignoring probabilities.

—me is the partitioning of edges shown in Figure 1.

—$\wp$ is the probability assignment in Table IV.

For example, annuals and annual_herbs are subclasses of plants, and annuals is an immediate subclass of plants while annual_herbs is not; annual_herbs is a subclass of the cluster {annuals, perennials}.

The POB schemas defined thus far may be inconsistent; that is, it may not always be possible to find a set of objects that satisfies the taxonomic and

[3] Note that we assume rational numbers here, since we adopt a probabilistic semantics of class hierarchies that is based on relative cardinalities of sets of objects. We can easily generalize our model to real numbers, if we assume a more general probabilistic semantics that is based on real-valued probability functions over a set of possible worlds. All the results of this subsection, except for the NP-membership result in Theorem 5.2, carry over to this more general setting.

Table III. Type Assignment τ

c	$\tau(c)$
plants	[pname: string, soil: soiltype, rain: integer]
annuals	[pname: string, soil: soiltype, rain: integer, sun: suntype]
perennials	[pname: string, soil: soiltype, rain: integer, sun: suntype, expyears: integer]
vegetables	[pname: string, soil: soiltype, rain: integer, sun: suntype, expyears: integer]
herbs	[pname: string, soil: soiltype, rain: integer, sun: suntype, expyears: integer, classification: string]
flowers	[pname: string, soil: soiltype, rain: integer, sun: suntype, expyears: integer, classification: string]
annuals_herbs	[pname: string, soil: soiltype, rain: integer, sun: suntype, expyears: integer, classification: string]
perennials_flowers	[pname: string, soil: soiltype, rain: integer, sun: suntype, expyears: integer, classification: string]

Table IV. Probability assignment $\wp$

Edge	Probability
annuals ⇒ plants	0.6
perennials ⇒ plants	0.4
vegetables ⇒ plants	0.2
herbs ⇒ plants	0.3
flowers ⇒ plants	0.4
annuals_herbs ⇒ annuals	0.4
annuals_herbs ⇒ herbs	0.8
perennials_flowers ⇒ perennials	0.3
perennials_flowers ⇒ flowers	0.3

probabilistic knowledge expressed by the directed acyclic graph, the partitioning of edges, and the probability assignment. The formal definition of consistency of a POB schema is given below.

Definition 5.5 (*Consistent POB Schema*). Let $\mathbf{S} = (\mathcal{C}, \tau, \Rightarrow, \text{me}, \wp)$ be a POB schema. An *interpretation* of $\mathbf{S}$ is any mapping ε from $\mathcal{C}$ to the set of all finite subsets of a set $\mathcal{O}$. An interpretation ε of $\mathbf{S}$ is called a *taxonomic model* of $\mathbf{S}$ if and only if.

C1 $\varepsilon(c) \neq \emptyset$, for all classes $c \in \mathcal{C}$;

C2 $\varepsilon(c) \subseteq \varepsilon(d)$, for all classes $c, d \in \mathcal{C}$ with $c \Rightarrow d$;

C3 $\varepsilon(c) \cap \varepsilon(d) = \emptyset$, for all distinct classes $c, d \in \mathcal{C}$ that belong to the same cluster $\mathcal{P} \in \cup\, \text{me}(\mathcal{C})$.

Two classes $c, d \in \mathcal{C}$ are *taxonomically disjoint* (*t-disjoint*) if and only if $\varepsilon(c) \cap \varepsilon(d) = \emptyset$ for all taxonomic models ε of $\mathbf{S}$. ε is a *taxonomic and probabilistic model* (or simply *model*) of $\mathbf{S}$ if and only if it is a taxonomic model of $\mathbf{S}$ and it satisfies the condition:

C4 $|\varepsilon(c)| = \wp(c, d) \cdot |\varepsilon(d)|$ for all classes $c, d \in \mathcal{C}$ with $c \Rightarrow d$.

We say $\mathbf{S}$ is *consistent* if and only if it has a model.

Let us illustrate this definition within the Plant Example.

Table V. Interpretation ε of Schema $\mathbf{S}$

c	$\varepsilon(c)$	$\lvert\varepsilon(c)\rvert$
plants	$\mathcal{O}_1 \cup \cdots \cup \mathcal{O}_{10}$	800
annuals	$\mathcal{O}_1 \cup \cdots \cup \mathcal{O}_5$	480
perennials	$\mathcal{O}_6 \cup \cdots \cup \mathcal{O}_{10}$	320
vegetables	$\mathcal{O}_1 \cup \mathcal{O}_9$	160
herbs	$\mathcal{O}_2 \cup \mathcal{O}_5 \cup \mathcal{O}_6$	240
flowers	$\mathcal{O}_3 \cup \mathcal{O}_7 \cup \mathcal{O}_{10}$	320
annuals_herbs	$\mathcal{O}_5$	192
perennials_flowers	$\mathcal{O}_{10}$	96

Example 5.5 (*Plant Example*: *Consistent POB Schema*). Let $\mathbf{S} = (\mathcal{C}, \tau, \Rightarrow, \text{me}, \wp)$ be the POB schema given in Example 5.4. Let $\mathcal{O}$ be a set of cardinality 800, which is partitioned into pairwise disjoint subsets $\mathcal{O}_1, \mathcal{O}_2, \ldots, \mathcal{O}_{10}$ having cardinalities 90, 27, 126, 45, 192, 21, 98, 35, 70, and 96, respectively. An interpretation ε of $\mathbf{S}$ is given in Table V. It is easy to see that ε is also a model of $\mathbf{S}$. For example, $\varepsilon(\text{plants}) \neq \emptyset$, $\varepsilon(\text{annuals}) \subseteq \varepsilon(\text{plants})$, $\varepsilon(\text{annuals}) \cap \varepsilon(\text{perennials}) = \emptyset$, and $|\varepsilon(\text{annuals})| = 0.6 \cdot |\varepsilon(\text{plants})|$. Hence, $\mathbf{S}$ is consistent.

It would be nice to have an efficient algorithm for deciding the consistency of a given POB Schema. For this purpose, we need a suitable characterization of consistency. The following condition is a natural candidate.

Definition 5.6 (*Pseudoconsistent POB Schema*). The POB schema $\mathbf{S} = (\mathcal{C}, \tau, \Rightarrow, \text{me}, \wp)$ is *pseudoconsistent* if and only if the following conditions hold.

P1 For any two different classes $c_1, c_2 \in \mathcal{C}$ with $c_1 \Rightarrow^\star c_2$, the product of the edge probabilities is the same on all paths from c_1 up to c_2.

P2 For all clusters $\mathcal{P} \in \cup\, \text{me}(\mathcal{C})$, no two distinct classes $c_1, c_2 \in \mathcal{P}$ have a common subclass.

Example 5.6 (*Plant Example*: *Pseudoconsistent POB Schema*). It is easy to see that the POB Schema $\mathbf{S} = (\mathcal{C}, \tau, \Rightarrow, \text{me}, \wp)$ shown in Example 5.4 is pseudoconsistent.

- —The two paths from annuals_herbs up to plants and from perennials_flowers up to plants have both 0.24 and 0.12, respectively, as the product of the edge probabilities.
- —Neither annuals_herbs nor perennials_flowers is a subclass of two t-disjoint classes.

Indeed, it is straightforward to show that pseudoconsistency is a necessary condition for consistency.

THEOREM 5.1. *Every consistent POB schema is pseudoconsistent.*

However, pseudoconsistency is not a sufficient condition for consistency. Even worse, deciding the consistency of a pseudoconsistent POB Schema is intractable. We have the following result.

THEOREM 5.2. *The problem of deciding whether a given POB Schema* **S** *is consistent is NP-complete. Hardness holds even if* **S** *is pseudoconsistent.*

PROOF. The problem is in NP, since it polynomially reduces to the NP-complete problem of deciding whether a weight formula is satisfiable in a measurable probability structure [Fagin et al. 1990]. Notice that the proof of NP-membership of the latter problem heavily relies on results from the theory of linear programming (the main idea is to prove a *small model theorem*, which says that a weight formula is satisfiable in a measurable probability structure if and only if it is satisfiable in a measurable probability structure of polynomial size; see Fagin et al. [1990]).

More precisely, *weight formulas* are defined as Boolean combinations of *basic weight formulas*, which are expressions of the form $a_1 \cdot w(\phi_1) + \cdots + a_k \cdot w(\phi_k) \geq a$ with integers $a_1, \ldots, a_k, a$ and propositional formulas $\phi_1, \ldots, \phi_k$. A *measurable probability structure* can be identified with a probability function on the finite set of all truth assignments to the primitive propositions, which is extended in a natural way to propositional formulas, basic weight formulas, and weight formulas.

It can now easily be shown that a POB schema $\mathbf{S} = (\mathcal{C}, \tau, \Rightarrow, \text{me}, \wp)$ is consistent if and only if the conjunction of the following weight formulas, which capture C1 to C4 in Definition 5.5, is satisfiable.

C1 $\neg((-1) \cdot w(c) \geq 0)$ for all classes $c \in \mathcal{C}$.

C2 $(w(c \wedge \neg d) \geq 0) \wedge ((-1) \cdot w(c \wedge \neg d) \geq 0)$ for all classes $c, d \in \mathcal{C}$ with $c \Rightarrow d$.

C3 $(w(c \wedge d) \geq 0) \wedge ((-1) \cdot w(c \wedge d) \geq 0)$ for all distinct classes $c, d \in \mathcal{C}$ of the same cluster.

C4 $(n \cdot w(c) + (-m) \cdot w(d) \geq 0) \wedge ((-n) \cdot w(c) + m \cdot w(d) \geq 0)$ for all classes $c, d \in \mathcal{C}$ with $c \Rightarrow d$, where m and n are natural numbers such that $\wp(c, d) = m/n$.

The proof of NP-hardness is given in Eiter et al. [1999]. □

Nonetheless, polynomial algorithms for deciding the consistency of a POB schema in relevant special cases may be possible. Well-structured POB schemas, which we introduce next, enjoy this property.

Definition 5.7 (*Well-Structured POB Schema*). The POB schema $\mathbf{S} = (\mathcal{C}, \tau, \Rightarrow, \text{me}, \wp)$ is *well structured* if and only if the following conditions hold.

W1 There exists a class $c \in \mathcal{C}$ such that every class $d \in \mathcal{C}$ is a subclass of c (i.e., the graph $(\mathcal{C}, \Rightarrow)$ has a *top element*).

W2 For every class $c \in \mathcal{C}$ and distinct $c_1, c_2 \in \text{S}(c)$, the set $S := \text{S}^\star(c_1) \cap \text{S}^\star(c_2)$ is either empty or has a unique element $d_m \neq c_1, c_2$ such that $d \Rightarrow^\star d_m$ for all $d \in S$ (i.e., for every class $c \in \mathcal{C}$, any two distinct immediate subclasses c_1 and c_2 of c either have no common subclass or a greatest common subclass d_m, which is different from them).

W3 For every class $c \in \mathcal{C}$, the undirected graph $G_{\mathbf{S}}(c) = (\mathcal{V}, \mathcal{E})$ that is defined by $\mathcal{V} = \text{me}(c)$ and $\mathcal{E} = \{\{\mathcal{P}_1, \mathcal{P}_2\} \in \mathcal{V} \times \mathcal{V} \mid \mathcal{P}_1 \neq \mathcal{P}_2, \cup\, \text{S}^\star(\mathcal{P}_1) \cap \cup\, \text{S}^\star(\mathcal{P}_2) \neq \emptyset\}$ is acyclic (i.e., for every class $c \in \mathcal{C}$, the partition clusters in me(c) are not cyclically connected through common subclasses. Roughly speaking, multiple inheritance does not cyclically connect partition clusters).

W4 For every class $c \in \mathcal{C}$: if the graph $G_{\mathbf{S}}(c)$ has an edge (i.e., two distinct clusters $\mathcal{P}_1, \mathcal{P}_2 \in \mathrm{me}(c)$ have a common subclass), then every path from a subclass of c to the top element of $(\mathcal{C}, \Rightarrow)$ goes through c (i.e., multiple inheritance can be *locally isolated* in the graph $(\mathcal{C}, \Rightarrow)$).

Informally, these conditions restrict multiple inheritance so that a model for the schema **S** can be built bottom-up from models of subschemas. Specifically, W2 and W3 ensure that a model for a subschema under a class $c \in \mathcal{C}$ can be constructed from models of smaller subschemas that are related to the partition clusters in me(c). Moreover, W4 and W1 ensure that these local constructions do not influence each other, and that they yield a model for the whole schema **S**, respectively. We feel that well-structuredness is a reasonable restriction on multiple inheritance.[4] In particular, W1 and W2 can always be satisfied by simply adding missing top elements to certain sets of classes (during the design of a POB schema, before specifying the probability assignment $\wp$). Let us now reconsider the Plant Example.

Example 5.7 (*Plant Example*: *Well-Structured POB Schema*). The POB schema **S** given in Example 5.4 is well structured.

—Every class is a subclass of plants.

—The classes annuals_herbs and perennials_flowers are t-disjoint.

—There are no cyclically connected partition clusters.

—The multiple inheritance at the classes annuals_herbs and perennials_flowers is locally isolated under the class plants.

As far as well-structured POB schemas are concerned, we have the nice result that pseudoconsistency is a necessary and sufficient condition for consistency. However, the proof of this result is highly nontrivial; see Eiter et al. [1999].

THEOREM 5.3. *Every pseudoconsistent and well-structured POB schema* **S** *is consistent.*

It is easily seen that any $\mathbf{S} = (\mathcal{C}, \tau, \Rightarrow, \mathrm{me}, \wp)$ without multiple inheritance (i.e., $|\{d \in \mathcal{C} \mid c \Rightarrow d\}| \leq 1$ for each class $c \in \mathcal{C}$) satisfies W2 to W4. We obtain the following corollary to Theorem 5.3.

COROLLARY 5.4. *Every POB schema with top element and without multiple inheritance is consistent.*

It now remains to show that pseudoconsistency and well-structuredness of a POB schema can be decided efficiently. We first show, via Algorithm 5.2 (which used Algorithm 5.1; please see Figures 3 and 4) how to check pseudoconsistency.

[4] Note that multiple inheritance was considered an optional feature of object-oriented database systems in the "manifesto" [Atkinson et al. 1989].

```
Algorithm 5.1: reachability(S)
Input: POB-schema $\mathbf{S} = (\mathcal{C}, \tau, \Rightarrow, \mathrm{me}, \wp)$.
Output: If $\mathbf{S}$ does not satisfy P1, then nil is returned. Otherwise, a mapping $w\colon \mathcal{C} \times \mathcal{C} \to [0,1]$
        is returned such that $w(c,d)$ is the product of the edge probabilities on all paths
        from $c$ up to $d$ if such a path exists and $w(c,d)$ is 0 otherwise.
 1. for each $c, d \in \mathcal{C}$ do
 2.   if $c = d$ then $w(c,d) := 1$
 3.     else if $c \Rightarrow d$ then $w(c,d) := \wp(c,d)$
 4.       else $w(c,d) := 0$;
 5. for each $c \in \mathcal{C}$ do begin
 6.   $\mathrm{S}(c) := \{d \in \mathcal{C} \mid d \Rightarrow c\}$;
 7.   $\delta(c) := |\{d \in \mathcal{C} \mid c \Rightarrow d\}|$
 8. end;
 9. $N := \{c \in \mathcal{C} \mid \delta(c) = 0\}$;
10. while $N \neq \emptyset$ do begin
11.   take any $c \in N$;
12.   for each $d \in \mathrm{S}(c)$ do begin
13.     $\delta(d) := \delta(d) - 1$;
14.     if $\delta(d) = 0$ then $N := N \cup \{d\}$;
15.     for each $e \in \mathcal{C}$ with $w(c,e) > 0$ do
16.       if $w(d,e) > 0$ and $w(d,e) \neq w(d,c) \cdot w(c,e)$ then return nil
17.         else $w(d,e) := w(d,c) \cdot w(c,e)$
18.   end;
19.   $N := N - \{c\}$;
20. end;
21. return $w$.
```

Fig. 3. Algorithm 5.1.

```
Algorithm 5.2: pseudo-consistent(S)
Input: POB-schema $\mathbf{S} = (\mathcal{C}, \tau, \Rightarrow, \mathrm{me}, \wp)$.
Output: true if $\mathbf{S}$ is pseudo-consistent and false otherwise.
1. $w :=$ reachability($\mathbf{S}$);
2. if $w = nil$ then return false;  ($\mathbf{S}$ does not satisfy P1)
3. for each $c \in \mathcal{C}$ do
4.   for each $\mathcal{P} \in \bigcup \mathrm{me}(\mathcal{C})$ do
5.     if $|\{e \in \mathcal{P} \mid w(c,e) > 0\}| > 1$ then return false;  ($\mathbf{S}$ does not satisfy P2)
6. return true.  ($\mathbf{S}$ is pseudo-consistent)
```

Fig. 4. Algorithm 5.2.

THEOREM 5.5. *The problem of deciding whether a given POB schema* $\mathbf{S} = (\mathcal{C}, \tau, \Rightarrow, \mathrm{me}, \wp)$ *is pseudoconsistent can be done using Algorithm* 5.2 *in time* $O(n(e+n))$, *where* $n = |\mathcal{C}|$ *and* e *is the number of directed edges in* $(\mathcal{C}, \Rightarrow)$.

PROOF. Algorithm 5.2 decides the pseudoconsistency of $\mathbf{S}$. It uses Algorithm 5.1, which computes the reachability relation of the graph $(\mathcal{C}, \Rightarrow)$.

Algorithm 5.1 works as follows. Steps 1 to 4 initialize the reachability relation. In Steps 5 to 8, for each class c, the set $S(c)$ of all its subclasses, and the number $\delta(c)$ of all its immediate superclasses are computed. Step 9 calculates the set of all classes c with $\delta(c) = 0$. The **while**-loop in 10 to 20 then computes the reachability relation. Every time Step 10 is entered, the product of the edge probabilities is correctly computed along all paths that involve only

```
Algorithm 5.3: well-structured(S)
Input: Pseudo-consistent POB-schema $\mathbf{S} = (\mathcal{C}, \tau, \Rightarrow, \mathrm{me}, \wp)$.
Output: true if $\mathbf{S}$ is well-structured and false otherwise.
Notation: We use $top(\mathbf{S})$ to denote the top element of $(\mathcal{C}, \Rightarrow)$. For classes $c \in \mathcal{C}$, the
          expression $\mathbf{S} - c$ denotes the POB-schema that is obtained from $\mathbf{S}$ by removing $c$.
          $\max(\mathcal{D})$, where $\mathcal{D} \subseteq \mathcal{C}$, is the set of all maximal members in $\mathcal{D}$ w.r.t. $\Rightarrow^\star$.
 1. for each $c \in \mathcal{C}$ do
 2.   $\delta(c) := |\{d \in \mathcal{C} \mid c \Rightarrow d\}|$;
 3. if $|\{c \in \mathcal{C} \mid \delta(c) = 0\}| > 1$ then return false;  ($\mathbf{S}$ does not satisfy W1)
 4. $w := reachability(\mathbf{S})$;
 5. $\mathcal{E} := \emptyset$;
 6. $M := \emptyset$;
 7. for each $c \in \mathcal{C}$ do
 8.   for each distinct $\mathcal{P}_1, \mathcal{P}_2 \in \mathrm{me}(c)$ do
 9.     for each $(c_1, c_2) \in \mathcal{P}_1 \times \mathcal{P}_2$ do begin
10.       $\mathcal{D} := \{d \in \mathcal{C} \mid w(d, c_1) > 0$ and $w(d, c_2) > 0\}$;
11.       if $|\max(\mathcal{D})| > 1$ or $\mathcal{D} \cap \{c_1, c_2\} \neq \emptyset$ then return false  ($\mathbf{S}$ does not satisfy W2)
12.         else if $|\max(\mathcal{D})| = 1$ then begin
13.           $\mathcal{E} := \mathcal{E} \cup \{\{\mathcal{P}_1, \mathcal{P}_2\}\}$;
14.           $M := M \cup \{c\}$
15.         end
16.     end;
17. if $(\bigcup \mathrm{me}(\mathcal{C}), \mathcal{E})$ contains a cycle then return false;  ($\mathbf{S}$ does not satisfy W3)
18. for each $c \in M$ do begin
19.   $v := reachability(\mathbf{S} - c)$;
20.   for each $d \in \mathcal{C}$ with $w(d, c) > 0$ do
21.     if $v(d, top(\mathbf{S})) > 0$ then return false;  ($\mathbf{S}$ does not satisfy W4)
22. end;
23. return true.  ($\mathbf{S}$ is well-structured)
```

Fig. 5. Algorithm 5.3.

edges $a \Rightarrow b$ with classes b such that $\delta(b) = 0$ and $b \notin N$. Given this, in 11 to 18, we take some $c \in N$, and we correctly compute the edge probabilities along all paths $d \Rightarrow c \Rightarrow^\star e$, where $c \Rightarrow^\star e$ involves only edges $a \Rightarrow b$ with classes b such that $\delta(b) = 0$ and $b \notin N$. In Step 16, we return *nil* when two distinct products are computed between two classes d and e.

In Algorithm 5.2, we first check in Steps 1 and 2 whether the schema **S** violates P1. We then check in Steps 3 to 5 whether a class c exists that is a subclass of two distinct classes in the same cluster, that is, whether **S** violates P2.

We now show that Algorithm 5.1 runs in time $O(n(e+n))$: The initialization Steps 1 to 4, 5 to 8, and 9 run in time $O(n^2)$, $O(ne)$, and $O(n)$, respectively. Next, it is easy to see that the **for**-loop in 15 to 17 is performed as many times as there are edges in $(\mathcal{C}, \Rightarrow)$, and each execution takes $O(n)$ time. Thus, the whole **while**-loop in 10 to 20 runs in time $O(ne)$.

Hence, Algorithm 5.2 also runs in time $O(n(e+n))$: The Steps 1 to 2 run in time $O(n(e+n))$. The **for**-loop in 4 to 5 runs in linear time in the input size of me (i.e., in e). Thus, the whole **for**-loop in 3 to 5 runs in time $O(ne)$. □

We now focus on deciding well-structuredness via Algorithm 5.3 (please see Figure 5).

THEOREM 5.6. *The problem of deciding whether a pseudoconsistent POB schema* $\mathbf{S} = (\mathcal{C}, \tau, \Rightarrow, \mathrm{me}, \wp)$ *is well structured can be solved using Algorithm* 5.3 *in time* $O(n^2 e)$*, where* $n = |\mathcal{C}|$ *and* e *is the number of directed edges in* $(\mathcal{C}, \Rightarrow)$.

PROOF. Algorithm 5.3 decides the well-structuredness of $\mathbf{S}$. Steps 1 to 3 check whether $\mathbf{S}$ satisfies W1. In 4 to 16, it is then checked whether $\mathbf{S}$ satisfies W2. Moreover, the union of all undirected graphs $G_{\mathbf{S}}(c)$ with $c \in \mathcal{C}$ and the set of all classes with multiple inheritance at subclasses are computed. Step 17 checks that all the graphs $G_{\mathbf{S}}(c)$ with $c \in \mathcal{C}$ are acyclic, that is, that $\mathbf{S}$ satisfies W3. In Steps 18 to 22, it is finally checked whether $\mathbf{S}$ satisfies W4.

We now show that Algorithm 5.3 runs in time $O(n^2 e)$. It is easy to see that Steps 1 to 2, 3, and 4 run in time $O(ne)$, $O(n)$, and $O(n(e+n)) = O(ne)$, respectively (note that W1 ensures $e \geq n - 1$). Step 10 is done one time for each edge in $(\mathcal{C}, \Rightarrow)$ and each class in a set of classes limited by $\mathcal{C}$. The set $\mathcal{D}$ there can be computed in time $O(n)$. The tests in Steps 11 and 12 can be done, using a simple algorithm, in time $O(n)$. Hence, Steps 5 to 16 run in time $O(n^2 e)$. In Step 17, the number of clusters in $\bigcup \mathrm{me}(\mathcal{C})$ is in the worst case equal to e. Thus, Step 17 can be performed in time $O(e)$ using standard algorithms for checking acyclicity. Finally, it is easy to see that Steps 18 to 22 run in time $O(n^2 e)$. □

6. INHERITANCE AND PROBABILISTIC OBJECT BASE INSTANCES

Thus far, we have not addressed inheritance of attributes that may arise through subclass relationships in a POB schema $\mathbf{S}$. For example, if c is a subclass of d, and d's type has a top-level attribute A, then class c should inherit this attribute, unless c already has such an attribute. The issue of inheritance has been extensively discussed in the literature, for example, Bertino and Martino [1993]. We now incorporate inheritance in our framework, and define instances of a POB schema.

6.1 Inheritance Completion and Fully Inherited Schemas

The main idea behind the inheritance of attributes is that every class should inherit all top-level attributes of all its superclasses. In order to handle cases in which the same attribute is inherited from more than one superclass, we make use of inheritance strategies.

Let us assume that any schema $\mathbf{S} = (\mathcal{C}, \tau, \Rightarrow, \mathrm{me}, \wp)$ has an associated *inheritance strategy* $\mathsf{inh}_{\mathbf{S}}$ that determines from which superclass d a class c inherits a top-level attribute A. More formally, let $\mathbf{A}$ denote the set of all top-level attributes of $\mathbf{S}$. For each pair $(c, A) \in \mathcal{C} \times \mathbf{A}$, let c_A be defined as

$$c_A = \{d \in \mathcal{C} \mid c \Rightarrow^{\star} d,\ A \text{ is a top-level attribute of } \tau(d)\}.$$

We now define $\mathsf{inh}_{\mathbf{S}}\colon \mathcal{C} \times \mathbf{A} \to \mathcal{C}$ as a partial mapping that assigns each pair $(c, A) \in \mathcal{C} \times \mathbf{A}$ with $c_A \neq \emptyset$ a minimal element in c_A under the partial order $\Rightarrow^{\star}$ (the value of $\mathsf{inh}_{\mathbf{S}}(c, A)$ is undefined if $c_A = \emptyset$). In particular, $\mathsf{inh}_{\mathbf{S}}(c, A) = c$ if A is a top-level attribute of $\tau(c)$.

This notion of inheritance strategy covers strategies (such as an ordering on classes) that are commonly used to resolve multiple inheritance in practice. Similarly, if we wish to use the strategy of the O_2 system [Bancilhon et al.

Table VI. Redesigned type assignment τ'

c	$\tau'(c)$
plants	[pname: string, soil: soiltype, rain: integer]
annuals	[sun: suntype]
perennials	[sun: suntype, expyears: integer]
vegetables	[sun: suntype, expyears: integer]
herbs	[sun: suntype, expyears: integer, classification: string]
flowers	[sun: suntype, expyears: integer, classification: string]
annuals_herbs	[ ]
perennials_flowers	[ ]

1991] where renamed inheritance of the same attribute with distinct origins is desired, we could generalize $\mathsf{inh}_{\mathbf{S}}(c, A)$ to return all pairs (d, A') with classes d from which attribute A, renamed A', is inherited.

Applying $\mathsf{inh}_{\mathbf{S}}$ on a POB schema $\mathbf{S} = (\mathcal{C}, \tau, \Rightarrow, \mathrm{me}, \wp)$ induces another POB schema $\mathbf{S}^\star = (\mathcal{C}, \tau^\star, \Rightarrow, \mathrm{me}, \wp)$, which only differs from $\mathbf{S}$ in its type assignment $\tau^\star$. More precisely, for each $c \in \mathcal{C}$, we define $\tau^\star(c) = [A_1\colon \tau(d_1).A_1, \ldots, A_k\colon \tau(d_k).A_k]$, where $A_1, \ldots, A_k$ are the top-level attributes that are inherited by c via $\mathsf{inh}_{\mathbf{S}}$ from the classes $d_1, \ldots, d_k$, respectively. We call $\mathbf{S}^\star$ the *inheritance completion* of $\mathbf{S}$. A POB schema $\mathbf{S}$ is *fully inherited* if and only if $\mathbf{S} = \mathbf{S}^\star$.

Example 6.1 (*Plant Example*: *Probabilistic Object Base Schema*). Let us consider the POB schema $\mathbf{S} = (\mathcal{C}, \tau, \Rightarrow, \mathrm{me}, \wp)$ for the Plant Example defined in Example 5.4. It is easily checked that for every subclass c of any class d, each top-level attribute of $\tau(d)$ is a top-level attribute of $\tau(c)$; that is, all attributes in d are already present in c. Thus, no attributes are inherited from proper superclasses, which means that $\mathbf{S}$ is fully inherited. The type assignment τ in $\mathbf{S}$ may be considered ill-designed, however, as it does not reflect natural inheritance relationships.

Consider now the redesigned schema $\mathbf{S}' = (\mathcal{C}, \tau', \Rightarrow, \mathrm{me}, \wp)$ with the redesigned type assignment τ' shown in Table VI, and adopt an inheritance strategy $\mathsf{inh}_{\mathbf{S}'}$ that resolves multiple inheritance by ordering "left-to-right" in Figure 1, that is, orders annuals before herbs and perennials before flowers.[5] Then, the original schema $\mathbf{S}$ is given by the inheritance completion of $\mathbf{S}'$.

In the rest of this article, we implicitly assume that all POB schemas $\mathbf{S}$ are consistent (see Definition 5.5), and that they are all fully inherited. In particular, POB instances in the next subsection, and operations in our POB algebra in Section 7 are defined with respect to fully inherited POB schemas. Note that these definitions can easily be extended to POB schemas $\mathbf{S}$ that are not fully inherited.

6.2 Probabilistic Object Base Instance

We are now ready to define a probabilistic object base instance (POB instance). The following assumption is common in object-oriented databases [Kim 1990].

Assumption. In the rest of this article, we assume that there is a (countably) infinite set $\mathcal{O}$ of *object identifiers* (*oids*).

[5] No renaming is assumed here for the same attribute with distinct origins.

Table VII. Mappings π and $\pi^\star$

c	$\pi(c)$	$\pi^\star(c)$
plants	$\{o_1\}$	$\{o_1, o_2, o_3, o_4, o_5, o_6, o_7\}$
annuals	$\{\}$	$\{o_2, o_3, o_5, o_6, o_7\}$
perennials	$\{\}$	$\{o_4\}$
vegetables	$\{\}$	$\{\}$
herbs	$\{\}$	$\{o_2, o_3, o_5, o_6, o_7\}$
flowers	$\{\}$	$\{o_4\}$
annuals_herbs	$\{o_2, o_3, o_5, o_6, o_7\}$	$\{o_2, o_3, o_5, o_6, o_7\}$
perennials_flowers	$\{o_4\}$	$\{o_4\}$

Each object, represented by an oid, is associated with a value. The objects populate a POB instance as follows.

Definition 6.1 (*Probabilistic Object Base Instance*). Let $\mathbf{S} = (\mathcal{C}, \tau, \Rightarrow, \text{me}, \wp)$ be a consistent POB schema. A *probabilistic object base instance (POB instance)* over $\mathbf{S}$ is a pair (π, ν), where:

—$\pi : \mathcal{C} \to 2^{\mathcal{O}}$ maps each class c to a finite subset of $\mathcal{O}$, such that $\pi(c_1) \cap \pi(c_2) = \emptyset$ for different $c_1, c_2 \in \mathcal{C}$. That is, the classes in $\mathcal{C}$ are mapped to pairwise disjoint sets of oids. We use $\pi(\mathcal{C})$ to abbreviate $\bigcup\{\pi(c) \mid c \in \mathcal{C}\}$. We define the mapping $\pi^\star : \mathcal{C} \to 2^{\mathcal{O}}$ by $\pi^\star(c) = \bigcup\{\pi(c') \mid c' \in \mathcal{C}, c' \Rightarrow^\star c\}$.

Intuitively, $\pi(c)$ denotes the ids of all objects that are *defined in* the class c, while $\pi^\star(c)$ denotes the ids of all objects that *belong to* the class c (i.e., that are defined in c or in one of its proper subclasses).

—ν maps each oid $o \in \pi(\mathcal{C})$ to a probabilistic tuple value of the appropriate type, that is, type $\tau(c)$ for the class c such that $o \in \pi(c)$.

Let us provide a POB instance for the POB schema of Example 5.4.

Example 6.2 (*Plant Example*: *Probabilistic Object Base Instance*). A POB instance over the POB schema shown in Example 5.4 is given as follows.

—π and $\pi^\star$ are the mappings shown in Table VII. Clearly, this is a very simple probabilistic object base (it contains only seven distinct objects).

—ν is the mapping shown in Table VIII.

In classical object bases, the extent of a class c consists of all oids belonging to c. The probabilistic extent of c specifies the probability that an oid belongs to c.

Definition 6.2 (*Probabilistic Extent*). Let $\mathbf{I} = (\pi, \nu)$ be a POB instance over the consistent POB schema $\mathbf{S} = (\mathcal{C}, \tau, \Rightarrow, \text{me}, \wp)$. For all classes $c \in \mathcal{C}$, the *probabilistic extent* of c, denoted ext(c), maps each oid $o \in \pi(\mathcal{C})$ to a *set* of rational numbers in [0, 1] as follows.

1. If $o \in \pi^\star(c)$, then $\text{ext(c)}(o) = \{1\}$.
2. If $o \in \pi^\star(c')$ with a class $c' \in \mathcal{C}$ that is t-disjoint from c (i.e., for all models ε of $\mathbf{S}$, the sets $\varepsilon(c')$ and $\varepsilon(c)$ are disjoint), then $\text{ext(c)}(o) = \{0\}$.
3. Otherwise, $\text{ext(c)}(o) = \{p \mid p$ is the product of the edge probabilities on a path from c up to a class $c' \in \mathcal{C}$, where c' is minimal with $o \in \pi^\star(c')$ and $c \Rightarrow^\star c'\}$.

Table VIII. Value Assignment ν

oid	$\nu(oid)$	*oid*	$\nu(oid)$
o_1	[pname: ⟨{Lady-Fern, Ostrich-Fern}, u, u⟩, soil: ⟨{loamy}, u, u⟩, rain: ⟨{25, . . . , 30}, u, u⟩]	o_5	[pname: ⟨{Thyme}, u, u⟩, soil: ⟨{loamy}, u, u⟩, rain: ⟨{20, . . . , 25}, u, u⟩, sun: ⟨{mild, medium}, 0.8 u, 1.2 u⟩, expyears: ⟨{2, 3}, 0.8 u, 1.2 u⟩, classification: ⟨{french, silver, wooly}, 0.6 u, 1.8 u⟩]
o_2	[pname: ⟨{Cuban-Basil, Lemon-Basil}, u, u⟩, soil: ⟨{loamy, sandy}, 0.7 u, 1.3 u⟩, rain: ⟨{20, . . . , 30}, u, u⟩, sun: ⟨{mild, medium}, 0.8 u, 1.2 u⟩, expyears: ⟨{2, 3, 4}, 0.6 u, 1.8 u⟩, classification: ⟨{french, silver, wooly}, 0.6 u, 1.8 u⟩]	o_6	[pname: ⟨{Mint}, u, u⟩, soil: ⟨{loamy}, u, u⟩, rain: ⟨{20}, u, u⟩, sun: ⟨{mild}, u, u⟩, expyears: ⟨{2, 3, 4}, 0.6 u, 1.8 u⟩, classification: ⟨{apple, curly}, 0.6 u, 1.4 u⟩]
o_3	[pname: ⟨{Mint}, u, u⟩, soil: ⟨{loamy}, u, u⟩, rain: ⟨{20}, u, u⟩, sun: ⟨{mild}, u, u⟩, expyears: ⟨{2, 3, 4}, 0.6 u, 1.8 u⟩, classification: ⟨{french, silver, wooly}, 0.6 u, 1.8 u⟩]	o_7	[pname: ⟨{Sage}, u, u⟩, soil: ⟨{sandy}, u, u⟩, rain: ⟨{20, 21}, u, u⟩, sun: ⟨{mild}, u, u⟩, expyears: ⟨{2, 3, 4}, 0.6 u, 1.8 u⟩, classification: ⟨{red, tricolor}, 0.6 u, 1.4 u⟩]
o_4	[pname: ⟨{Aster, Salvia}, u, u⟩, soil: ⟨{loamy, sandy}, 0.6 u, 1.4 u⟩, rain: ⟨{20, . . . , 25}, u, u⟩, sun: ⟨{mild}, u, u⟩, expyears: ⟨{2, 3, 4}, 0.6 u, 1.8 u⟩, classification: ⟨{french, silver, wooly}, 0.6 u, 1.8 u⟩]		

We return to the Plant Example to see what the extents of the various classes are.

Example 6.3 (*Plant Example*: *Probabilistic Extent*). In the Plant Example, the probabilistic extents of annuals_herbs and perennials_flowers are given as follows.

$$\begin{array}{ll}
\text{ext(annuals\_herbs)}(o_1) = \{0.24\} & \text{ext(perennials\_flowers)}(o_1) = \{0.12\} \\
\text{ext(annuals\_herbs)}(o_2) = \{1\} & \text{ext(perennials\_flowers)}(o_2) = \{0\} \\
\text{ext(annuals\_herbs)}(o_3) = \{1\} & \text{ext(perennials\_flowers)}(o_3) = \{0\} \\
\text{ext(annuals\_herbs)}(o_4) = \{0\} & \text{ext(perennials\_flowers)}(o_4) = \{1\} \\
\text{ext(annuals\_herbs)}(o_5) = \{1\} & \text{ext(perennials\_flowers)}(o_5) = \{0\} \\
\text{ext(annuals\_herbs)}(o_6) = \{1\} & \text{ext(perennials\_flowers)}(o_6) = \{0\} \\
\text{ext(annuals\_herbs)}(o_7) = \{1\} & \text{ext(perennials\_flowers)}(o_7) = \{0\}.
\end{array}$$

Definition 6.3 (*Coherent POB Instance*). Let $\mathbf{I} = (\pi, \nu)$ be a POB instance over the consistent POB schema $\mathbf{S} = (\mathcal{C}, \tau, \Rightarrow, \text{me}, \wp)$. The POB-instance $\mathbf{I}$ is *coherent* if and only if for all classes $c \in \mathcal{C}$ and all objects $o \in \pi(\mathcal{C})$, the probabilistic extent $\text{ext(c)}(o)$ contains *at most* one element.

It is easy to see that the Plant Example described thus far is coherent and that testing coherence of a given POB instance **I** of a consistent schema **S** can be done in polynomial time.

7. PROBABILISTIC OBJECT BASES: ALGEBRAIC OPERATIONS

In this section, we formally define the analogues of the classical relational operations on POBs. All standard operations on POBs take POB instances as input, and produce POB instances as output. Recall that all POB schemas of input POB instances are implicitly assumed to be consistent and fully inherited.

The probability computations in our POB algebra are based on probabilistic combination strategies. As shown in Eiter et al. [2000a], all common probabilistic combination strategies can be computed in a constant number of operations from the input intervals. Thus, under these strategies, the probability computations in our POB algebra are all tractable, and it is easy to see that our algebraic operations are all computable in polynomial time in the size of the input POB instances.

7.1 Selection

Intuitively, given a POB instance **I** over the POB schema **S**, the result of a selection operation is another POB instance $\mathbf{I}'$ over **S** such that the objects in the extents of the classes in $\mathbf{I}'$ all satisfy the selection condition of the query. Before describing the selection operation, we formally define the syntax and semantics of selection conditions. We start by defining the syntax of path and selection expressions.

Definition 7.1 (*Path Expression*). Let $\tau = [A_1\colon \tau_1, \ldots, A_k\colon \tau_k]$ be any type. We define *path expressions* by induction as follows. (i) Every A_i is a path expression for τ, and (ii) if P_i is a path expression for τ_i, then $A_i.P_i$ is a path expression for τ, for every $i = 1, \ldots, k$.

We use the Plant Example to demonstrate some path expressions.

Example 7.1 (*Plant Example*: *Path Expression*). In the Plant Example, two path expressions for the type [pname: string, size: [height: integer, width: integer]] are given by pname and size.height.

We now define the syntax of atomic selection expressions.

Definition 7.2 (*Atomic Selection Expression*). Let $\mathbf{S} = (\mathcal{C}, \tau, \Rightarrow, \mathrm{me}, \wp)$ be a POB schema and let $\mathcal{X}$ be a set of *object variables*. An *atomic selection expression* has one of the following forms.

—$x \in c$, where x is an object variable from $\mathcal{X}$, and c is a class from $\mathcal{C}$.

—$x.P\ \theta\ v$, where x is an object variable from $\mathcal{X}$, P is a path expression over attributes from $\mathcal{A}$, θ is a binary predicate from $\{=, \neq, \leq, \geq, <, >, \subseteq, \supseteq, \in, \ni\}$, and v is a value.

—$x.P_1 =_{\otimes} x.P_2$, where x is an object variable from $\mathcal{X}$, P_1 and P_2 are two distinct path expressions over attributes from $\mathcal{A}$, and $\otimes$ is a probabilistic conjunction strategy.

Let us consider some examples of atomic selection expressions.

Example 7.2 (*Plant Example*: *Atomic Selection Expression*). In the Plant Example, some atomic selection expressions are as follows (x is an object variable).

— Find all objects *that are annuals and herbs.* This selection can be represented by the atomic selection expression $x \in \mathsf{annuals\_herbs}$.

— Find all objects *that require a mild sun.* This selection can be represented by the atomic selection expression $x.\mathsf{sun} = \mathsf{mild}$.

— Find all objects *that require over 21 units of rain.* This selection can be represented by the atomic selection expression $x.\mathsf{rain} > 21$.

We now define the syntax of selection expressions.

Definition 7.3 (*Selection Expression*). Let **S** be a POB schema. We define *conjunctive* and *disjunctive selection expressions* by induction as follows.

If ϕ is an atomic selection expression and $\otimes$ is a probabilistic conjunction strategy, then ϕ is a conjunctive selection expression over $\otimes$. If ϕ and ψ are conjunctive selection expressions over the same object variable and the same probabilistic conjunction strategy $\otimes$, then $\phi \otimes \psi$ is a conjunctive selection expression over $\otimes$.

If ϕ is an atomic selection expression and $\oplus$ is a probabilistic disjunction strategy, then ϕ is a disjunctive selection expression over $\oplus$. If ϕ and ψ are disjunctive selection expressions over the same object variable and the same probabilistic disjunction strategy $\oplus$, then $\phi \oplus \psi$ is a disjunctive selection expression over $\oplus$.

A *selection expression* is a conjunctive or disjunctive selection expression.

Let us illustrate this definition via the Plant Example.

Example 7.3 (*Plant Example*: *Selection Expression*). In the Plant Example, some selection expressions are given as follows (x is an object variable).

— The atomic selection expressions $x \in \mathsf{annuals\_herbs}$, $x.\mathsf{sun} = \mathsf{mild}$, and $x.\mathsf{rain} > 21$ given in Example 7.2 are selection expressions.

— Find all objects *that are annuals and herbs and that require a mild sun.* This selection can be represented by the selection expression $x \in \mathsf{annuals\_herbs} \otimes x.\mathsf{sun} = \mathsf{mild}$, where $\otimes$ is a probabilistic conjunction strategy.

— Find all objects *that require a mild sun or over 21 units of rain.* This selection can be represented by the selection expression $x.\mathsf{sun} = \mathsf{mild} \oplus x.\mathsf{rain} > 21$, where $\oplus$ is a probabilistic disjunction strategy.

We are now ready to define the syntax of selection conditions.

Definition 7.4 (*Selection Condition*). Let **S** be a POB schema. (i) If ϕ is a selection expression and L and U are real numbers from $[0, 1]$ with $L \leq U$, then $(\phi)[L, U]$ is a selection condition. (ii) If α and β are selection conditions *over the same object variable*, then $\neg\alpha$, $(\alpha \wedge \beta)$, and $(\alpha \vee \beta)$ are selection conditions.

Let us consider some examples of selection conditions.

Example 7.4 (*Plant Example*: *Selection Condition*). In the Plant Example, some selection conditions are given as follows (x is an object variable).

- The selection of all objects *that require both a mild sun and over 21 units of rain with a probability of 30 to 50%*, can be done by using the selection condition

  $$(x.\mathsf{sun} = \mathsf{mild} \otimes x.\mathsf{rain} > 21)[0.3, 0.5]\,,$$

  where $\otimes$ is a probabilistic conjunction strategy.
- The selection of all objects *that require a mild sun with a probability of at least 40%*, and *over 21 units of rain with a probability of at least 80%*, can be done by using the selection condition:

  $$(x.\mathsf{sun} = \mathsf{mild})[0.4, 1] \wedge (x.\mathsf{rain} > 21)[0.8, 1]\,.$$

- The selection of all objects *that do not require a mild sun with a probability of at least 40%* can be done by using the selection condition:

  $$\neg(x.\mathsf{sun} = \mathsf{mild})[0.4, 1]\,.$$

Note that a selection expression and a selection condition can contain exactly one object variable.

It now remains to define the semantics of selection expressions and conditions. For this purpose, each triple $(\mathbf{S}, \mathbf{I}, o)$ consisting of a POB schema $\mathbf{S} = (\mathcal{C}, \tau, \Rightarrow, \mathrm{me}, \wp)$, a POB instance $\mathbf{I} = (\pi, \nu)$ over $\mathbf{S}$, and an oid $o \in \pi(\mathcal{C})$ in $\mathbf{I}$ is associated with a probabilistic interpretation $\mathrm{prob}_{\mathbf{S},\mathbf{I},o}$, which assigns a probability interval to selection expressions, and a truth value to selection conditions. We first interpret path and selection expressions.

Definition 7.5 (*Interpretation of Path Expressions*). Suppose we are given a tuple type $\tau = [A_1\colon \tau_1, \ldots, A_k\colon \tau_k]$. The *interpretation* of a path expression P for τ under a value $v = [A_1\colon v_1, \ldots, A_k\colon v_k]$ of type τ, denoted $v.P$, is inductively defined by $v.A_i = v_i$ and $v.A_i.P_i = v_i.P_i$, for every $i = 1, \ldots, k$.

The following example shows how path expressions are interpreted.

Example 7.5 (*Plant Example*: *Interpretation of Path Expressions*). In the Plant Example, the interpretation of the path expressions pname and size.height under the value [pname: Thyme, size: [height: 4, width: 12]] is given by the values Thyme and 4, respectively.

We next assign probabilistic intervals to atomic selection expressions.

Definition 7.6 (*Interpretation of Atomic Selection Expressions*). Suppose we are given a POB instance $\mathbf{I} = (\pi, \nu)$ over the POB schema $\mathbf{S} = (\mathcal{C}, \tau, \Rightarrow, \mathrm{me}, \wp)$, and let $o \in \pi(\mathcal{C})$. The *probabilistic interpretation* with respect to $\mathbf{S}$, $\mathbf{I}$, and o, denoted $\mathrm{prob}_{\mathbf{S},\mathbf{I},o}$, is the partial mapping from all atomic selection expressions to the set of all subintervals of [0, 1] that is defined as follows.

- $\mathrm{prob}_{\mathbf{S},\mathbf{I},o}(x \in c) = [\min(\mathrm{ext(c)}(o)), \max(\mathrm{ext(c)}(o))]$. Intuitively, $\mathrm{prob}_{\mathbf{S},\mathbf{I},o}(x \in c)$ describes the interval for the probability that the object o belongs to the class c.

—If $\nu(o).A = \langle V, \alpha, \beta\rangle$, and $P = AP'$ is a path expression for the type of o, where P' is either empty or of the form $.P''$, then:

$$\mathrm{prob}_{\mathbf{S},\mathbf{I},o}(x.P\,\theta\,v) = \begin{cases} [\sum_{u\in W}\alpha(u),\ \min(1, \sum_{u\in W}\beta(u))], & \text{if W} \neq \emptyset\ ; \\ [0,0], & \text{otherwise,} \end{cases}$$

where $W = \{u \in V \mid uP'\,\theta\,v\}$. Note that $\mathrm{prob}_{\mathbf{S},\mathbf{I},o}(x.P\,\theta\,v)$ is undefined,[6] if P is undefined for $\nu(o)$, or if $uP'\,\theta\,v$ is undefined for some $u \in V$. Intuitively, $\mathrm{prob}_{\mathbf{S},\mathbf{I},o}(x.AP'\,\theta\,v)$ describes the interval for the probability that the object o has a value u in attribute A such that $uP'\,\theta\,v$.

—If $\nu(o).A_i = \langle V_i, \alpha_i, \beta_i\rangle$, and $P_i = A_iP_i'$ is a path expression for the type of o, where P_i' is either empty or of the form $.P_i''$, for $i \in \{1, 2\}$, then:

$$\mathrm{prob}_{\mathbf{S},\mathbf{I},o}(x.P_1 =_{\otimes} x.P_2) = \begin{cases} [\sum_{u\in W}\alpha(u),\ \min(1, \sum_{u\in W}\beta(u))], & \text{if } W \neq \emptyset; \\ [0,0], & \text{otherwise,} \end{cases}$$

where $W = \{(u_1, u_2) \in V_1 \times V_2 \mid u_1P_1' = u_2P_2'\}$, and

$$[\alpha(u), \beta(u)] = [\alpha_1(u_1), \beta_1(u_1)] \otimes [\alpha_2(u_2), \beta_2(u_2)] \text{ for all } u = (u_1, u_2) \in W.$$

Note that $\mathrm{prob}_{\mathbf{S},\mathbf{I},o}(x.P_1 =_{\otimes} x.P_2)$ is undefined, if P_1 or P_2 is undefined for $\nu(o)$. Intuitively, $\mathrm{prob}_{\mathbf{S},\mathbf{I},o}(x.A_1P_1' =_{\otimes} x.A_2P_2')$ describes the interval for the probability that the object o has a value u_1 in attribute A_1 *and* a value u_2 in attribute A_2 such that $u_1P_1' = u_2P_2'$. The selected conjunction strategy $\otimes$ reflects the dependencies between the two attributes A_1 and A_2.

Let us give an example to illustrate this definition.

Example 7.6 (*Interpretation of Atomic Selection Expressions*). In the Plant Example, the probabilistic interpretations $\mathrm{prob}_{\mathbf{S},\mathbf{I},o}$ with $o \in \{o_1, o_2, \ldots, o_7\}$ map the atomic selection expressions $x \in \mathsf{annuals\_herbs}$, $x.\mathsf{sun} = \mathsf{mild}$, and $x.\mathsf{rain} > 21$ to the subintervals of $[0, 1]$ shown in Table IX.

We now assign probabilistic intervals to selection expressions.

Definition 7.7 (*Interpretation of Selection Expressions*). Let $\mathbf{I} = (\pi, \nu)$ be a POB instance over the POB schema $\mathbf{S} = (\mathcal{C}, \tau, \Rightarrow, \mathrm{me}, \wp)$ and let $o \in \pi(\mathcal{C})$. We extend $\mathrm{prob}_{\mathbf{S},\mathbf{I},o}$ to a partial mapping from the set of all selection expressions to the set of all closed subintervals of [0,1] as follows.

$$\mathrm{prob}_{\mathbf{S},\mathbf{I},o}(\phi \otimes \psi) = \mathrm{prob}_{\mathbf{S},\mathbf{I},o}(\phi) \otimes \mathrm{prob}_{\mathbf{S},\mathbf{I},o}(\psi).$$
$$\mathrm{prob}_{\mathbf{S},\mathbf{I},o}(\phi \oplus \psi) = \mathrm{prob}_{\mathbf{S},\mathbf{I},o}(\phi) \oplus \mathrm{prob}_{\mathbf{S},\mathbf{I},o}(\psi).$$

Let us illustrate this definition via the Plant Example.

Example 7.7 (*Plant Example*: *Interpretation of Selection Expressions*). In the Plant Example, the two selection expressions $\phi_{st} =$ "$x \in \mathsf{annuals\_herbs} \otimes_{st} x.\mathsf{sun} = \mathsf{mild}$" and $\psi_{st} =$ "$x.\mathsf{sun} = \mathsf{mild} \otimes_{st} x.\mathsf{rain} > 21$" are assigned the subintervals of $[0, 1]$ shown in Table X.

[6] As a consequence, two selections on $\mathbf{I}$ with respect to logically equivalent selection conditions ϕ_1 and ϕ_2 generally produce the same result only when $\mathrm{prob}_{\mathbf{S},\mathbf{I},o}$ is defined for every atomic selection expression in ϕ_1 and ϕ_2, and every object o in $\mathbf{I}$.

Table IX. Interpretation of Atomic Selection Expressions

o	$\mathrm{prob}_{\mathbf{S},\mathbf{I},o}(x \in \textsf{annuals\_herbs})$	$\mathrm{prob}_{\mathbf{S},\mathbf{I},o}(x.\textsf{sun} = \textsf{mild})$	$\mathrm{prob}_{\mathbf{S},\mathbf{I},o}(x.\textsf{rain} > 21)$
o_1	$[0.24, 0.24]$	Undefined	$[1.00, 1.00]$
o_2	$[1.00, 1.00]$	$[0.40, 0.60]$	$[0.82, 0.82]$
o_3	$[1.00, 1.00]$	$[1.00, 1.00]$	$[0.00, 0.00]$
o_4	$[0.00, 0.00]$	$[1.00, 1.00]$	$[0.67, 0.67]$
o_5	$[1.00, 1.00]$	$[0.40, 0.60]$	$[0.67, 0.67]$
o_6	$[1.00, 1.00]$	$[1.00, 1.00]$	$[0.00, 0.00]$
o_7	$[1.00, 1.00]$	$[1.00, 1.00]$	$[0.00, 0.00]$

Table X. Interpretation of Selection Expressions

o	$\mathrm{prob}_{\mathbf{S},\mathbf{I},o}(\phi_{in})$	$\mathrm{prob}_{\mathbf{S},\mathbf{I},o}(\phi_{ig})$	$\mathrm{prob}_{\mathbf{S},\mathbf{I},o}(\psi_{in})$	$\mathrm{prob}_{\mathbf{S},\mathbf{I},o}(\psi_{ig})$
o_1	Undefined	Undefined	Undefined	Undefined
o_2	$[0.40, 0.60]$	$[0.40, 0.60]$	$[0.33, 0.49]$	$[0.22, 0.60]$
o_3	$[1.00, 1.00]$	$[1.00, 1.00]$	$[0.00, 0.00]$	$[0.00, 0.00]$
o_4	$[0.00, 0.00]$	$[0.00, 0.00]$	$[0.67, 0.67]$	$[0.67, 0.67]$
o_5	$[0.40, 0.60]$	$[0.40, 0.60]$	$[0.27, 0.40]$	$[0.07, 0.60]$
o_6	$[1.00, 1.00]$	$[1.00, 1.00]$	$[0.00, 0.00]$	$[0.00, 0.00]$
o_7	$[1.00, 1.00]$	$[1.00, 1.00]$	$[0.00, 0.00]$	$[0.00, 0.00]$

We are now ready to assign truth values to selection conditions.

Definition 7.8 (*Satisfaction of Selection Conditions*). Let $\mathbf{I} = (\pi, \nu)$ be a POB instance over the POB schema $\mathbf{S} = (\mathcal{C}, \tau, \Rightarrow, \mathrm{me}, \wp)$ and let $o \in \pi(\mathcal{C})$. We extend $\mathrm{prob}_{\mathbf{S},\mathbf{I},o}$ to selection conditions as follows.

—$\mathrm{prob}_{\mathbf{S},\mathbf{I},o} \models (\phi)[L, U]$ if and only if $\mathrm{prob}_{\mathbf{S},\mathbf{I},o}(\phi) \subseteq [L, U]$.

—$\mathrm{prob}_{\mathbf{S},\mathbf{I},o} \models \neg\phi$ if and only if it is not the case that $\mathrm{prob}_{\mathbf{S},\mathbf{I},o} \models \phi$.

—$\mathrm{prob}_{\mathbf{S},\mathbf{I},o} \models \phi \wedge \psi$ if and only if $\mathrm{prob}_{\mathbf{S},\mathbf{I},o} \models \phi$ and $\mathrm{prob}_{\mathbf{S},\mathbf{I},o} \models \psi$.

—$\mathrm{prob}_{\mathbf{S},\mathbf{I},o} \models \phi \vee \psi$ if and only if $\mathrm{prob}_{\mathbf{S},\mathbf{I},o} \models \phi$ or $\mathrm{prob}_{\mathbf{S},\mathbf{I},o} \models \psi$.

Let us give an illustrating example.

Example 7.8 (*Plant Example*: *Satisfaction of Selection Conditions*). In the Plant Example, we have:

—$\mathrm{prob}_{\mathbf{S},\mathbf{I},o_2} \models (x.\textsf{sun} = \textsf{mild} \otimes_{in} x.\textsf{rain} > 21)[0.3, 0.5]$ (see Example 7.7);

—$\mathrm{prob}_{\mathbf{S},\mathbf{I},o_2} \not\models (x.\textsf{sun} = \textsf{mild} \otimes_{ig} x.\textsf{rain} > 21)[0.3, 0.5]$ (see Example 7.7);

—$\mathrm{prob}_{\mathbf{S},\mathbf{I},o_2} \models (x.\textsf{sun} = \textsf{mild})[0.4, 1] \wedge (x.\textsf{rain} > 21)[0.8, 1]$ (see Example 7.6);

—$\mathrm{prob}_{\mathbf{S},\mathbf{I},o_3} \not\models (x.\textsf{sun} = \textsf{mild})[0.4, 1] \wedge (x.\textsf{rain} > 21)[0.8, 1]$ (see Example 7.6).

We are now finally ready to define the selection operation.

Definition 7.9 (*Selection on POB Instances*). Let $\mathbf{I} = (\pi, \nu)$ be a POB instance over the POB schema $\mathbf{S} = (\mathcal{C}, \tau, \Rightarrow, \mathrm{me}, \wp)$ and let ϕ be a selection condition over the object variable x. The *selection on* $\mathbf{I}$ *with respect to* ϕ, denoted $\sigma_\phi(\mathbf{I})$, is the POB instance (π', ν') over $\mathbf{S}$, where:

—$\pi'(c) = \{o \in \pi(c) \mid \mathrm{prob}_{\mathbf{S},\mathbf{I},o} \models \phi\}$;

—$\nu' = \nu \mid \pi'(\mathcal{C})$ (i.e., the mapping ν restricted to $\pi'(\mathcal{C})$).

Table XI. π' Resulting from Selection

c	$\pi'(c)$
plants	$\{\}$
annuals	$\{\}$
perennials	$\{\}$
vegetables	$\{\}$
herbs	$\{\}$
flowers	$\{\}$
annuals_herbs	$\{o_2\}$
perennials_flowers	$\{\}$

Table XII. ν' Resulting from Selection

oid	$\nu'(oid)$
o_2	[pname: $\langle\{$Cuban-Basil, Lemon-Basil$\}, \mathrm{u}, \mathrm{u}\rangle$, soil: $\langle\{$loamy, sandy$\}, 0.7\,\mathrm{u}, 1.3\,\mathrm{u}\rangle$, rain: $\langle\{20, \ldots, 30\}, \mathrm{u}, \mathrm{u}\rangle$, sun: $\langle\{$mild, medium$\}, 0.8\,\mathrm{u}, 1.2\,\mathrm{u}\rangle$, expyears: $\langle\{2, 3, 4\}, 0.6\,\mathrm{u}, 1.8\,\mathrm{u}\rangle$, classification: $\langle\{$french, silver, wooly$\}, 0.6\,\mathrm{u}, 1.8\,\mathrm{u}\rangle$]

The following example shows precisely what happens in the Plant Example when we perform selection with respect to selection conditions.

Example 7.9 (*Plant Example*: *Selection*). In the Plant Example, the selection on $\mathbf{I} = (\pi, \nu)$ with respect to the selection condition

$$(x.\mathsf{sun} = \mathsf{mild})[0.4, 1] \wedge (x.\mathsf{rain} > 21)[0.8, 1]$$

is the POB instance (π', ν') over $\mathbf{S}$ (see Example 7.6), where π' and ν' are shown in Tables XI and XII, respectively. This result is also obtained by the selection on $\mathbf{I}$ with respect to $(x.\mathsf{sun} = \mathsf{mild} \otimes_{in} x.\mathsf{rain} > 21)[0.3, 0.5]$ (see Example 7.7).

The selection on $\mathbf{I}$ with respect to $(x.\mathsf{sun} = \mathsf{mild} \otimes_{ig} x.\mathsf{rain} > 21)[0.3, 0.5]$, in contrast, produces the empty POB instance over $\mathbf{S}$ (see Example 7.7).

7.2 Projection and Renaming

In this section, we define the projection of POB instances on arbitrary sets of attributes, and the renaming of (top-level) attributes in POB instances. We start by defining projection on POB instances. We first define the projection of POB schemas on sets of attributes.

Definition 7.10 (*Projection of POB Schemas*). Let $\mathbf{S} = (\mathcal{C}, \tau, \Rightarrow, \mathrm{me}, \wp)$ be a POB schema and let $\mathbf{A}$ be a set of attributes. The *projection of* $\mathbf{S}$ *on* $\mathbf{A}$, denoted $\Pi_A(\mathbf{S})$, is the POB schema $(\mathcal{C}, \tau', \Rightarrow, \mathrm{me}, \wp)$, where the new type $\tau'(c)$ of each class $c \in \mathcal{C}$ is obtained from the old type $\tau(c) = [B_1 \colon \tau_1, \ldots, B_k \colon \tau_k]$ by deleting all $B_j \colon \tau_j$s with $B_j \notin \mathbf{A}$.

Let us consider an example to illustrate the projection of POB schemas.

Example 7.10 (*Plant Example: Projection of POB Schemas*). Consider the POB schema $\mathbf{S}$ described in Example 5.4. Then, the projection of $\mathbf{S}$ on the set of attributes $\mathbf{A} = \{\mathsf{pname}, \mathsf{rain}\}$ has the type assignment τ' shown in Table XIII.

Table XIII. τ' Resulting from Projection

c	$\tau'(c)$
plants	[pname: string, rain: integer]
annuals	[pname: string, rain: integer]
perennials	[pname: string, rain: integer]
vegetables	[pname: string, rain: integer]
herbs	[pname: string, rain: integer]
flowers	[pname: string, rain: integer]
annuals_herbs	[pname: string, rain: integer]
perennials_flowers	[pname: string, rain: integer]

We next define the projection of probabilistic tuple values.

Definition 7.11 (*Projection of Probabilistic Tuple Values*). Let *ptv* be a probabilistic tuple value of the form $[B_1\!:(V_1, \alpha_1, \beta_1), \ldots, B_k\!:(V_k, \alpha_k, \beta_k)]$ and let $\mathbf{A}$ be a set of attributes. The *projection* of *ptv* on $\mathbf{A}$, denoted Π_A (*ptv*), is obtained from $[B_1\!:(V_1, \alpha_1, \beta_1), \ldots, B_k\!:(V_k, \alpha_k, \beta_k)]$ by deleting all $B_j\!:(V_j, \alpha_j, \beta_j)$s with $B_j \notin \mathbf{A}$.

We give a small example to illustrate the projection of probabilistic tuple values.

Example 7.11 (*Plant Example*: *Projection of Probabilistic Tuple Values*). Let the probabilistic tuple value *ptv* be given as follows (note that *ptv* is associated with the object o_2 in Example 6.2).

$$\begin{aligned} ptv = [\ & \text{pname: } \langle\{\text{Cuban-Basil, Lemon-Basil}\}, \mathrm{u}, \mathrm{u}\rangle, \\ & \text{soil: } \langle\{\text{loamy, sandy}\}, 0.7\,\mathrm{u}, 1.3\,\mathrm{u}\rangle, \\ & \text{rain: } \langle\{20, \ldots, 30\}, \mathrm{u}, \mathrm{u}\rangle, \\ & \text{sun: } \langle\{\text{mild, medium}\}, 0.8\,\mathrm{u}, 1.2\,\mathrm{u}\rangle, \\ & \text{expyears: } \langle\{2, 3, 4\}, 0.6\,\mathrm{u}, 1.8\,\mathrm{u}\rangle, \\ & \text{classification: } \langle\{\text{french, silver, wooly}\}, 0.6\,\mathrm{u}, 1.8\,\mathrm{u}\rangle]\,. \end{aligned}$$

The projection of *ptv* on the set of attributes $\mathbf{A}$ = {pname, rain} is given as follows.

$$\Pi_A(ptv) = [\text{pname: } \langle\{\text{Cuban-Basil, Lemon-Basil}\}, \mathrm{u}, \mathrm{u}\rangle,\ \text{rain: } \langle\{20, \ldots, 30\}, \mathrm{u}, \mathrm{u}\rangle]\,.$$

We are now ready to define projection of POB instances.

Definition 7.12 (*Projection of POB Instances*). Let $\mathbf{I} = (\pi, \nu)$ be a POB instance over the POB schema $\mathbf{S} = (\mathcal{C}, \tau, \Rightarrow, \mathrm{me}, \wp)$ and let $\mathbf{A}$ be a set of attributes. The *projection of* $\mathbf{I}$ *on* $\mathbf{A}$, denoted $\Pi_A(\mathbf{I})$, is defined as the POB instance (π', ν') over the POB schema $\Pi_A(\mathbf{S})$, where:

—$\pi'(c) = \pi(c)$ for all classes $c \in \mathcal{C}$;

—$\nu'(o) = \Pi_A(\nu(o))$ for all oids $o \in \pi(\mathcal{C})$.

Let us illustrate this definition within the Plant Example.

Example 7.12 (*Plant Example*: *Projection of POB Instances*). Let us consider the POB instance $\mathbf{I} = (\pi, \nu)$ described in Example 6.2. The projection

Table XIV. ν' Resulting from Projection

oid	$\nu'(oid)$
o_1	[pname: $\langle$\{Lady-Fern, Ostrich-Fern\}, u, u$\rangle$, rain: $\langle\{25, \ldots, 30\}$, u, u$\rangle$]
o_2	[pname: $\langle$\{Cuban-Basil, Lemon-Basil\}, u, u$\rangle$, rain: $\langle\{20, \ldots, 30\}$, u, u$\rangle$]
o_3	[pname: $\langle$\{Mint\}, u, u$\rangle$, rain: $\langle\{20\}$, u, u$\rangle$]
o_4	[pname: $\langle$\{Aster, Salvia\}, u, u$\rangle$, rain: $\langle\{20, \ldots, 25\}$, u, u$\rangle$]
o_5	[pname: $\langle$\{Thyme\}, u, u$\rangle$, rain: $\langle\{20, \ldots, 25\}$, u, u$\rangle$]
o_6	[pname: $\langle$\{Mint\}, u, u$\rangle$, rain: $\langle\{20\}$, u, u$\rangle$]
o_7	[pname: $\langle$\{Sage\}, u, u$\rangle$, rain: $\langle\{20, 21\}$, u, u$\rangle$]

of $\mathbf{I}$ on $\mathbf{A} = \{\text{pname}, \text{rain}\}$ is the POB instance (π', ν'), where π' is the same as π, and ν' is given in Table XIV.

We next define the renaming of (top-level) attributes in POB instances. This operation is especially useful in connection with Cartesian product and join (see Sections 7.3 and 7.4). We first define renaming expressions.

Definition 7.13 (*Renaming Expression*). Let $\mathbf{S} = (\mathcal{C}, \tau, \Rightarrow, \text{me}, \wp)$ be a POB schema and let $\mathbf{A}$ be the set of all top-level attributes of $\mathbf{S}$. A *renaming expression* has the form $\vec{\mathbf{B}} \leftarrow \vec{\mathbf{C}}$, where $\vec{\mathbf{B}} = B_1, B_2, \ldots, B_l$ is a list of distinct attributes from $\mathbf{A}$, and $\vec{\mathbf{C}} = C_1, C_2, \ldots, C_l$ is a list of distinct attributes from $\mathcal{A} - (A - \{B_1, B_2, \ldots, B_l\})$ (this condition ensures that each attribute C_i that belongs to $\mathbf{A}$ must also occur in $\vec{\mathbf{B}}$; that is, each such C_i must itself be renamed).

We now define the renaming of attributes in POB schemas.

Definition 7.14 (*Renaming in POB Schemas*). Let $\mathbf{S} = (\mathcal{C}, \tau, \Rightarrow, \text{me}, \wp)$ be a POB schema and let $N = B_1, B_2, \ldots, B_l \leftarrow C_1, C_2, \ldots, C_l$ be a renaming expression. The *renaming in* $\mathbf{S}$ *with respect to* N, denoted $\delta_N(\mathbf{S})$, is the POB schema $(\mathcal{C}, \tau', \Rightarrow, \text{me}, \wp)$, where the new type $\tau'(c)$ of each class $c \in \mathcal{C}$ is obtained from the old type $\tau(c) = [A_1 \colon \tau_1, \ldots, A_k \colon \tau_k]$ by replacing each attribute A_j with $A_j = B_i$ for some $i \in \{1, \ldots, l\}$ by the new attribute C_i.

Note. Although the above definition does not include renaming of nested attributes, this may be accomplished by a straightforward extension. For the sake of simplicity, we skip this.

Let us give an example to illustrate the renaming of attributes in POB schemas.

Example 7.13 (*Plant Example: Renaming in POB Schemas*). Let us consider again the POB schema $\mathbf{S}$ computed in Example 7.10. The renaming in $\mathbf{S}$ with respect to the renaming expression

$$\text{pname}, \text{rain} \leftarrow \text{pname2}, \text{rain2}$$

has the type assignment τ' shown in Table XV.

We next define the renaming of attributes in probabilistic tuple values.

Definition 7.15 (*Renaming in Probabilistic Tuple Values*). Let *ptv* be a probabilistic tuple value of the form $[A_1 \colon (V_1, \alpha_1, \beta_1), \ldots, A_k \colon (V_k, \alpha_k, \beta_k)]$ and

Table XV. τ' Resulting from Renaming

c	$\tau'(c)$
plants	[pname2: string, rain2: integer]
annuals	[pname2: string, rain2: integer]
perennials	[pname2: string, rain2: integer]
vegetables	[pname2: string, rain2: integer]
herbs	[pname2: string, rain2: integer]
flowers	[pname2: string, rain2: integer]
annuals_herbs	[pname2: string, rain2: integer]
perennials_flowers	[pname2: string, rain2: integer]

let $N = B_1, B_2, \ldots, B_l \leftarrow C_1, C_2, \ldots, C_l$ be a renaming expression. The *renaming in ptv with respect to* N, denoted $\delta_N(ptv)$, is obtained from $[A_1\colon(V_1, \alpha_1, \beta_1), \ldots, A_k\colon(V_k, \alpha_k, \beta_k)]$ by replacing each attribute A_j with $A_j = B_i$ for some $i \in \{1, \ldots, l\}$ by the new attribute C_i.

We are now ready to define the renaming of attributes in POB instances.

Definition 7.16 (*Renaming in POB Instances*). Let $\mathbf{I} = (\pi, \nu)$ be a POB instance over the POB schema $\mathbf{S} = (\mathcal{C}, \tau, \Rightarrow, \text{me}, \wp)$ and let N be a renaming expression. The *renaming in* $\mathbf{I}$ *with respect to* N, denoted $\delta_N(\mathbf{I})$, is defined as the POB instance (π', ν') over the POB schema $\delta_N(\mathbf{S})$, where:

—$\pi'(c) = \pi(c)$ for all classes $c \in \mathcal{C}$;

—$\nu'(o) = \delta_N(\nu(o))$ for all oids $o \in \pi(\mathcal{C})$.

Let us illustrate this definition within the Plant Example.

Example 7.14 (*Plant Example*: *Renaming in POB Instances*). Let us consider the POB instance $\mathbf{I} = (\pi, \nu)$ computed in Example 7.12. The renaming in $\mathbf{I}$ with respect to the renaming expression pname, rain ← pname2, rain2 is the POB instance (π', ν'), where π' is the same as π, and ν' is given in Table XVI.

7.3 Cartesian Product

In relational databases, the Cartesian product of two relations consists of the set of all tuples that can be obtained by concatenating a tuple in the first relation with a tuple in the second relation. If one follows this intuition, the Cartesian product of two POB instances should be obtained by concatenating the property list of any object in the first POB instance with the property list of any object in the second POB instance. This will be the intuition underlying our definition of Cartesian product (a similar idea stands behind Ojoin by Shaw and Zdonik [1990]).

Let us first come back to the Plant Example to show that the Cartesian product is meaningful.

Example 7.15 (*Plant Example*: *Cartesian Product*). Suppose we are interested in pairs of plants that flourish with a certain probability in the same environment (e.g., in pairs of plants that have the same rain requirements with some probability). To obtain this information, we must somehow connect the knowledge tied to each oid with the knowledge tied to other oids.

Table XVI. ν' Resulting from Renaming

oid	$\nu'(oid)$
o_1	[pname2: ⟨{Lady-Fern, Ostrich-Fern}, u, u⟩, rain2: ⟨{25, ..., 30}, u, u⟩]
o_2	[pname2: ⟨{Cuban-Basil, Lemon-Basil}, u, u⟩, rain2: ⟨{20, ..., 30}, u, u⟩]
o_3	[pname2: ⟨{Mint}, u, u⟩, rain2: ⟨{20}, u, u⟩]
o_4	[pname2: ⟨{Aster, Salvia}, u, u⟩, rain2: ⟨{20, ..., 25}, u, u⟩]
o_5	[pname2: ⟨{Thyme}, u, u⟩, rain2: ⟨{20, ..., 25}, u, u⟩]
o_6	[pname2: ⟨{Mint}, u, u⟩, rain2: ⟨{20}, u, u⟩]
o_7	[pname2: ⟨{Sage}, u, u⟩, rain2: ⟨{20, 21}, u, u⟩]

The first challenge in defining the Cartesian product of two POB instances is the following. Suppose we know that the POB schemas of our two POB instances are $\mathbf{S}_1 = (\mathcal{C}_1, \tau_1, \Rightarrow_1, \mathrm{me}_1, \wp_1)$ and $\mathbf{S}_2 = (\mathcal{C}_2, \tau_2, \Rightarrow_2, \mathrm{me}_2, \wp_2)$. Let the POB schema of the Cartesian product instance be denoted by $\mathbf{S} = (\mathcal{C}, \tau, \Rightarrow, \mathrm{me}, \wp)$. What should the relationship among $\mathbf{S}_1$, $\mathbf{S}_2$, and $\mathbf{S}$ be?

Recall first that in classical relational algebra (based on attributes, and not on a numbering of the columns in each relation), the Cartesian product $R_1 \times R_2$ of two relation schemas R_1 and R_2 is in general only defined if they have disjoint sets of attributes. Thus, we define the Cartesian product only for two input schemas $\mathbf{S}_1$ and $\mathbf{S}_2$ that do not have any top-level attributes in common.

Recall next that in classical relational algebra, $R_1 \times R_2$ and $R_2 \times R_1$ yield the same schema. Similarly, we also desire that $\mathbf{S}_1 \times \mathbf{S}_2 = \mathbf{S}_2 \times \mathbf{S}_1$ holds in our POB algebra. Suppose now that the sets of classes of $\mathbf{S}_1 \times \mathbf{S}_2$ and $\mathbf{S}_2 \times \mathbf{S}_1$ are given by $\mathcal{C}_1 \times \mathcal{C}_2$ and $\mathcal{C}_2 \times \mathcal{C}_1$, respectively. Then, the desired relationship $\mathbf{S}_1 \times \mathbf{S}_2 = \mathbf{S}_2 \times \mathbf{S}_1$ implies the condition $\mathcal{C}_1 \times \mathcal{C}_2 = \mathcal{C}_2 \times \mathcal{C}_1$. The latter is achieved by the following technique of assuming that every set of classes of a POB schema is actually a classical relation over a classical relation schema.

Assumption. In the rest of this article, we assume that for each POB schema $\mathbf{S} = (\mathcal{C}, \tau, \Rightarrow, \mathrm{me}, \wp)$, the set of classes $\mathcal{C}$ is a classical relation over a classical relation schema $R(\mathbf{S}) = \{A_1, \dots, A_m\}$ associated with $\mathbf{S}$. That is, each class $c \in \mathcal{C}$ is considered as a tuple over $R(\mathbf{S})$. In particular, for each basic POB schema $\mathbf{S}$, the relation schema $R(\mathbf{S})$ consists of a single distinguished attribute $A_{\mathbf{S}}$.

Thus, as another restriction on the input schemas $\mathbf{S}_1$ and $\mathbf{S}_2$, we also assume that $R(\mathbf{S}_1)$ and $R(\mathbf{S}_2)$ are disjoint. We now summarize which POB schemas $\mathbf{S}_1$ and $\mathbf{S}_2$ can be combined using Cartesian product.

Definition 7.17 (*Cartesian-Product-Compatible POB Schemas*). The POB schemas $\mathbf{S}_1 = (\mathcal{C}_1, \tau_1, \Rightarrow_1, \mathrm{me}_1, \wp_1)$ and $\mathbf{S}_2 = (\mathcal{C}_2, \tau_2, \Rightarrow_2, \mathrm{me}_2, \wp_2)$ are *Cartesian-product-compatible* if and only if $\mathbf{S}_1$ and $\mathbf{S}_2$ do not have any top-level attributes in common, and $R(\mathbf{S}_1)$ and $R(\mathbf{S}_2)$ are disjoint.

Note that any two POB schemas $\mathbf{S}_1$ and $\mathbf{S}_2$ can be made Cartesian-product-compatible by renaming all the common top-level attributes of $\mathbf{S}_1$ and $\mathbf{S}_2$, and all the attributes in $R(\mathbf{S}_1) \cap R(\mathbf{S}_2)$.

We are now ready to define the Cartesian product of two schemas $\mathbf{S}_1$ and $\mathbf{S}_2$.

Definition 7.18 (*Cartesian Product of POB Schemas*). Let $\mathbf{S}_1 = (\mathcal{C}_1, \tau_1, \Rightarrow_1, \mathrm{me}_1, \wp_1)$ and $\mathbf{S}_2 = (\mathcal{C}_2, \tau_2, \Rightarrow_2, \mathrm{me}_2, \wp_2)$ be two Cartesian-product-compatible POB schemas, and let $R_1 = R(\mathbf{S}_1)$ and $R_2 = R(\mathbf{S}_2)$. The *Cartesian product* of $\mathbf{S}_1$ and $\mathbf{S}_2$, denoted $\mathbf{S}_1 \times \mathbf{S}_2$, is the POB schema $\mathbf{S} = (\mathcal{C}, \tau, \Rightarrow, \mathrm{me}, \wp)$ such that:

- $\mathcal{C} = \mathcal{C}_1 \times \mathcal{C}_2$;
- for all classes $c \in \mathcal{C}$, let $\tau(c[R_1], c[R_2]) = [A_1 \colon \tau_1, \ldots, A_k \colon \tau_k, A_{k+1} \colon \tau_{k+1}, \ldots, A_{k+m} \colon \tau_{k+m}]$, where $\tau_1(c[R_1]) = [A_1 \colon \tau_1, \ldots, A_k \colon \tau_k]$ and $\tau_2(c[R_2]) = [A_{k+1} \colon \tau_{k+1}, \ldots, A_{k+m} \colon \tau_{k+m}]$;[7]
- the directed acyclic graph $(\mathcal{C}, \Rightarrow)$ is defined as follows. For all $c, d \in \mathcal{C}$:

$$c \Rightarrow d \quad \text{if and only if} \quad (c[R_1] \Rightarrow_1 d[R_1] \wedge c[R_2] = d[R_2])$$

or

$$(c[R_1] = d[R_1] \wedge c[R_2] \Rightarrow_2 d[R_2]).$$

- The partitioning me is given as follows. For all $c \in \mathcal{C}$:

$$\mathrm{me}(c) = \{\mathcal{P}_1 \times \{c[R_2]\} \mid \mathcal{P}_1 \in \mathrm{me}_1(c[R_1])\} \cup \{\{c[R_1]\} \times \mathcal{P}_2 \mid \mathcal{P}_2 \in \mathrm{me}_2(c[R_2])\}.$$

- The probability assignment $\wp$ is defined as follows. For all $c \Rightarrow d$:

$$\wp(c, d) = \begin{cases} \wp_1(c[R_1], d[R_1]) & \text{if } c[R_2] = d[R_2] \\ \wp_2(c[R_2], d[R_2]) & \text{if } c[R_1] = d[R_1]. \end{cases}$$

(Note that $\mathcal{C} = \mathcal{C}_1 \times \mathcal{C}_2$ implicitly defines that $R(\mathbf{S}) = R_1 \cup R_2$.)

Let us illustrate this definition within the Plant Example.

Example 7.16 (*Plant Example*: *Cartesian Product of POB Schemas*). Let $\mathbf{S}_1$ be the POB schema computed in Example 7.10, and let $\mathbf{S}_2$ be the POB schema computed in Example 7.13 in which each class c is replaced by c'. The Cartesian product schema $\mathbf{S}_1 \times \mathbf{S}_2 = (\mathcal{C}, \tau, \Rightarrow, \mathrm{me}, \wp)$ is given as follows.

- A partial view on the set of classes $\mathcal{C}$ is given in Figure 6 (note that we use pl, an, pe, ve, he, fl, ah, and pf as abbreviations for plants, annuals, perennials, vegetables, herbs, flowers, annuals_herbs, and perennials_flowers, respectively).
- Each class $c \in \mathcal{C}$ is assigned the following type under τ,

$$\tau(c) = [\text{pname: string, rain: integer, pname2: string, rain2: integer}].$$

- A partial view on the directed acyclic graph $(\mathcal{C}, \Rightarrow)$, the partitioning me, and the probability assignment $\wp$ is also given in Figure 6.

We now define the Cartesian product of probabilistic tuple values.

Definition 7.19 (*Cartesian Product of Probabilistic Tuple Values*). Let ptv_1 and ptv_2 be two probabilistic tuple values over the disjoint sets of attributes $\mathbf{A}_1$ and $\mathbf{A}_2$, respectively. The *Cartesian product* of ptv_1 and ptv_2, denoted $ptv_1 \times ptv_2$, is the probabilistic tuple value ptv over the set of attributes $A_1 \cup A_2$ defined by

[7] As usual, $c[U]$ denotes the restriction of tuple c to the attributes in U.

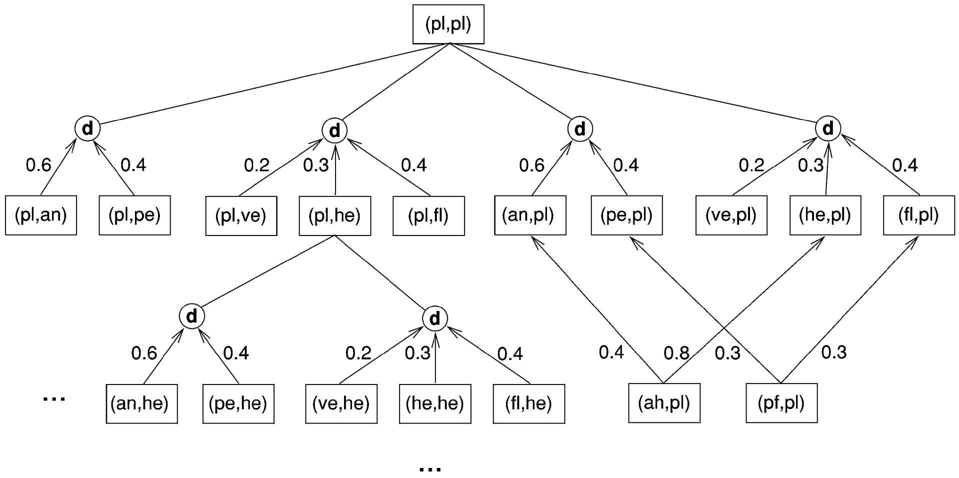


Fig. 6. Some classes in the Cartesian product of the Plant Example.

—$ptv.A = ptv_1.A$ for all attributes $A \in \mathbf{A}_1$;
—$ptv.A = ptv_2.A$ for all attributes $A \in \mathbf{A}_2$.

Note that $ptv_1 \times ptv_2 = ptv_2 \times ptv_1$, since by convention the ordering of attributes in a probabilistic tuple value is immaterial.

Example 7.17 (*Cartesian Product of Probabilistic Tuple Values*). Consider the following two probabilistic tuple values (taken from Examples 7.12 and 7.14, respectively).

$$\begin{aligned} ptv_1 &= [\text{pname: } \langle\{\text{Cuban-Basil, Lemon-Basil}\}, \text{u}, \text{u}\rangle, \text{rain: } \langle\{20, \ldots, 30\}, \text{u}, \text{u}\rangle] \\ ptv_2 &= [\text{pname2: } \langle\{\text{Mint}\}, \text{u}, \text{u}\rangle, \text{rain2: } \langle\{20\}, \text{u}, \text{u}\rangle]\,. \end{aligned}$$

The Cartesian product $ptv_1 \times ptv_2$ of ptv_1 and ptv_2 is given as follows,

$$\begin{aligned} &[\text{pname: } \langle\{\text{Cuban-Basil, Lemon-Basil}\}, \text{u}, \text{u}\rangle, \text{rain: } \langle\{20, \ldots, 30\}, \text{u}, \text{u}\rangle, \\ &\ \text{pname2: } \langle\{\text{Mint}\}, \text{u}, \text{u}\rangle, \text{rain2: } \langle\{20\}, \text{u}, \text{u}\rangle]. \end{aligned}$$

We are finally ready to define the Cartesian product of two POB instances.

Assumption. In the rest of this article, we assume that each oid $o \in \mathcal{O}$ that occurs in a POB instance $\mathbf{I}$ over $\mathbf{S}$ is a tuple over $R(\mathbf{S}) = \{A_1, \ldots, A_m\}$. Each such o may be written as $(o[A_1], \ldots, o[A_m])$.

Roughly speaking, each object in the Cartesian product instance is obtained from two objects in the input instances by first concatenating their two oids, and second collecting all their attribute values. The class in which the new object is defined is obtained by concatenating the classes in which the two input objects are defined.

Definition 7.20 (*Cartesian Product of POB Instances*). Let $\mathbf{I}_1 = (\pi_1, \nu_1)$ and $\mathbf{I}_2 = (\pi_2, \nu_2)$ be two POB instances over the Cartesian-product-compatible POB schemas $\mathbf{S}_1 = (\mathcal{C}_1, \tau_1, \Rightarrow_1, \text{me}_1, \wp_1)$ and $\mathbf{S}_2 = (\mathcal{C}_2, \tau_2, \Rightarrow_2, \text{me}_2, \wp_2)$, respectively, and let $R_1 = R(\mathbf{S}_1)$ and $R_2 = R(\mathbf{S}_2)$. The *Cartesian product* of $\mathbf{I}_1$ and $\mathbf{I}_2$, Table XVII. π Resulting from Cartesian Product (Partial View)

c	$\pi(c)$
(pl, pl)	$\{(o_1, o_1)\}$
(an, pl)	$\{\}$
(ah, pl)	$\{(o_2, o_1), (o_3, o_1), (o_5, o_1), (o_6, o_1), (o_7, o_1)\}$
(pf, pl)	$\{(o_4, o_1)\}$

Table XVIII. ν Resulting from Cartesian Product (Partial View)

oid	$\nu(oid)$
(o_1, o_1)	[pname: $\langle$\{Lady-Fern, Ostrich-Fern\}, u, u$\rangle$, rain: $\langle\{25, \ldots, 30\}$, u, u$\rangle$, pname2: $\langle$\{Lady-Fern, Ostrich-Fern\}, u, u$\rangle$, rain2: $\langle\{25, \ldots, 30\}$, u, u$\rangle$]
(o_2, o_1)	[pname: $\langle$\{Cuban-Basil, Lemon-Basil\}, u, u$\rangle$, rain: $\langle\{20, \ldots, 30\}$, u, u$\rangle$, pname2: $\langle$\{Lady-Fern, Ostrich-Fern\}, u, u$\rangle$, rain2: $\langle\{25, \ldots, 30\}$, u, u$\rangle$]
(o_3, o_1)	[pname: $\langle$\{Mint\}, u, u$\rangle$, rain: $\langle\{20\}$, u, u$\rangle$, pname2: $\langle$\{Lady-Fern, Ostrich-Fern\}, u, u$\rangle$, rain2: $\langle\{25, \ldots, 30\}$, u, u$\rangle$]

denoted $\mathbf{I}_1 \times \mathbf{I}_2$, is defined as the POB instance (π, ν) over the POB schema $\mathbf{S} = \mathbf{S}_1 \times \mathbf{S}_2$, where

—$\pi(c) = \pi_1(c[R_1]) \times \pi_2(c[R_2])$, for all $c \in \mathcal{C}$ (here, $\pi(c) \subseteq \mathcal{O}$ is assumed, for all $c \in \mathcal{C}$);

—$\nu(o) = \nu_1(o[R_1]) \times \nu_2(o[R_2])$, for all $o \in \pi(\mathcal{C})$.

Let us illustrate this definition within the Plant Example.

Example 7.18 (*Plant Example*: *Cartesian Product of POB Instances*). Let $\mathbf{I}_1$ and $\mathbf{I}_2$ be the two POB instances computed in Examples 7.12 and 7.14, respectively. The Cartesian product of $\mathbf{I}_1$ and $\mathbf{I}_2$ is the POB instance (π, ν), where partial views of π and ν are given in Tables XVII and XVIII, respectively.

7.4 Join

In classical relational databases, the join operator is a generalization of the Cartesian product. This is also the case for the join of POB instances, which is defined in this section. We start with the notion of join-compatibility.

Definition 7.21 (*Join-Compatible POB Schemas*). Two POB schemas $\mathbf{S}_1 = (\mathcal{C}_1, \tau_1, \Rightarrow_1, \mathrm{me}_1, \wp_1)$ and $\mathbf{S}_2 = (\mathcal{C}_2, \tau_2, \Rightarrow_2, \mathrm{me}_2, \wp_2)$ are *join-compatible* if and only if $R(\mathbf{S}_1)$ and $R(\mathbf{S}_2)$ are disjoint and $\tau_1(c_1).A = \tau_2(c_2).A$ for all classes $c_1 \in \mathcal{C}_1$, $c_2 \in \mathcal{C}_2$ and attributes A defined for both $\tau_1(c_1)$ and $\tau_2(c_2)$.

We next define the join of two POB schemas.

Definition 7.22 (*Join of POB Schemas*). Let $\mathbf{S}_1 = (\mathcal{C}_1, \tau_1, \Rightarrow_1, \mathrm{me}_1, \wp_1)$ and $\mathbf{S}_2 = (\mathcal{C}_2, \tau_2, \Rightarrow_2, \mathrm{me}_2, \wp_2)$ be two join-compatible POB schemas, and let $R_1 = R(\mathbf{S}_1)$ and $R_2 = R(\mathbf{S}_2)$. The *join* of $\mathbf{S}_1$ and $\mathbf{S}_2$, denoted $\mathbf{S}_1 \bowtie \mathbf{S}_2$, is the POB schema $\mathbf{S} = (\mathcal{C}, \tau, \Rightarrow, \mathrm{me}, \wp)$, where $\mathcal{C}$, $\Rightarrow$, me, and $\wp$ are as in the definition of $\mathbf{S} = \mathbf{S}_1 \times \mathbf{S}_2$ (see Definition 7.18), and τ is defined as follows.

—For all $c \in \mathcal{C}$, the tuple type $\tau(c) = [A_1 \colon \tau_1, \ldots, A_l \colon \tau_l]$ contains exactly all $A_i : \tau_i$ that belong to either the tuple type $\tau_1(c[R_1])$ or the tuple type $\tau_2(c[R_2])$.

For the join of two probabilistic tuple values ptv_1 and ptv_2, we need to combine the two values of a common attribute A_i to a single value for the result. This is done through conjunction of the probabilistic triples representing these values, along the following definition.

Definition 7.23 (*Conjunction Strategies on Probabilistic Triples*). Let $pt_1 = (V', \alpha', \beta')$, $pt_2 = (V'', \alpha'', \beta'')$ be probabilistic triples, and let $\otimes$ be a probabilistic conjunction strategy. Then, $pt_1 \otimes pt_2$ is the probabilistic triple $pt = (V, \alpha, \beta)$ with

—$V = \{v \in V' \cap V'' \mid [\alpha'(v), \beta'(v)] \otimes [\alpha''(v), \beta''(v)] \neq [0, 0]\}$;

—$[\alpha(v), \beta(v)] = [\alpha'(v), \beta'(v)] \otimes [\alpha''(v), \beta''(v)]$ for all $v \in V$.

Note that impossible values v in $V' \cap V''$ (having probability 0) are excluded from V as they are implicitly represented by the CWA. The outcome $pt = pt_1 \otimes pt_2$ is well defined only if pt is consistent, which requires that $\sum_{v \in V} \beta(v) \geq 1$. When an inconsistency arises, we flag an error. Moreover, note that when some intervals $[\alpha'(v), \beta'(v)]$ and $[\alpha''(v), \beta''(v)]$ are inconsistent under the event dependencies associated with $\otimes$ (see Definition 4.1), we also flag an error.

We now define the join of two probabilistic tuple values.

Definition 7.24 (*Join of Probabilistic Tuple Values*). Let ptv_1 and ptv_2 be two probabilistic tuple values over the sets of attributes $\mathbf{A}_1$ and $\mathbf{A}_2$, respectively, such that for all $A \in \mathbf{A}_1 \cap \mathbf{A}_2$, the values $ptv_1.A$ and $ptv_2.A$ are of the same type. Let $\otimes$ be a probabilistic conjunction strategy. The *join* of ptv_1 and ptv_2 under $\otimes$, denoted $ptv_1 \bowtie_\otimes ptv_2$, is the probabilistic tuple value ptv over $\mathbf{A}_1 \cup \mathbf{A}_2$ defined by

—$ptv.A = ptv_1.A$ for all attributes $A \in \mathbf{A}_1 - \mathbf{A}_2$;

—$ptv.A = ptv_2.A$ for all attributes $A \in \mathbf{A}_2 - \mathbf{A}_1$;

—$ptv.A = ptv_1.A \otimes ptv_2.A$ for all attributes $A \in \mathbf{A}_1 \cap \mathbf{A}_2$.

Note that for any probabilistic conjunction strategy $\otimes$, $ptv_1 \bowtie_\otimes ptv_2 = ptv_2 \bowtie_\otimes ptv_1$; that is, the join of probabilistic tuple values is commutative.

Example 7.19. Let us consider the following two probabilistic tuple values,

$$\begin{aligned} ptv_1 &= [A \colon \langle \{a, b\}, 0.6\,\mathrm{u}, 1.4\,\mathrm{u}\rangle,\ B \colon \langle \{a, c\}, 0.7\,\mathrm{u}, 1.3\,\mathrm{u}\rangle]\,, \\ ptv_2 &= [A \colon \langle \{a, b, c\}, 0.3\,\mathrm{u}, 2.4\,\mathrm{u}\rangle,\ C \colon \langle \{c, d\}, 0.4\,\mathrm{u}, 1.6\,\mathrm{u}\rangle]\,. \end{aligned}$$

The *join* $ptv_1 \bowtie_{\otimes_{in}} ptv_2$ of ptv_1 and ptv_2 under independence is given by

$$[A \colon \langle \{a, b\}, 0.06\,\mathrm{u}, 1.12\,\mathrm{u}\rangle,\ B \colon \langle \{a, c\}, 0.7\,\mathrm{u}, 1.3\,\mathrm{u}\rangle,\ C \colon \langle \{c, d\}, 0.4\,\mathrm{u}, 1.6\,\mathrm{u}\rangle]\,.$$

We are now ready to define the join of two POB instances.

Definition 7.25 (*Join of POB Instances*). Let $\mathbf{I}_1 = (\pi_1, \nu_1)$ and $\mathbf{I}_2 = (\pi_2, \nu_2)$ be POB instances over the join-compatible POB schemas $\mathbf{S}_1 = (\mathcal{C}_1, \tau_1, \Rightarrow_1, \mathrm{me}_1, \wp_1)$ and $\mathbf{S}_2 = (\mathcal{C}_2, \tau_2, \Rightarrow_2, \mathrm{me}_2, \wp_2)$, respectively, and let $R_1 = R(\mathbf{S}_1)$ and $R_2 = R(\mathbf{S}_2)$. Let A_1 and A_2 be the sets of top-level attributes of $\mathbf{S}_1$ and $\mathbf{S}_2$, respectively. Let

$\otimes$ be a probabilistic conjunction strategy. The *join* of $\mathbf{I}_1$ and $\mathbf{I}_2$ under $\otimes$, denoted $\mathbf{I}_1 \bowtie_\otimes \mathbf{I}_2$, is the POB instance (π, ν) over the POB schema $\mathbf{S}_1 \bowtie \mathbf{S}_2$, where:

—$\pi(c) = \{(o_1, o_2) \in \pi_1(c[R_1]) \times \pi_2(c[R_2]) \mid$ for all $A \in \mathbf{A}_1 \cap \mathbf{A}_2$: if $(\nu_1(o_1) \bowtie_\otimes \nu_2(o_2)).A = \langle V, \alpha, \beta\rangle$, then $V \neq \emptyset\}$, for all $c \in \mathcal{C}_1 \times \mathcal{C}_2$;

—$\nu(o) = \nu_1(o[R_1]) \bowtie_\otimes \nu_2(o[R_2])$, for all $o \in \pi(\mathcal{C})$.

We remark that the join is the only operation of our algebra in which probabilistic attribute values of *two distinct* objects are combined. The selection operation and the intersection operation (see next subsection), in contrast, only allow the combining of probabilistic attribute values assigned to *one single* object.

7.5 Intersection, Union, and Difference

In this section, we define the classical set operations of *intersection*, *union*, and *difference* for two POB instances over the same schema.

The definition of intersection is intuitive: common objects are selected, and their respective attribute values are combined by conjunction.

Definition 7.26 (*Intersection of Probabilistic Tuple Values*). Let ptv_1 and ptv_2 be two probabilistic tuple values over the same set of attributes $\mathbf{A}$, and let $\otimes$ be a probabilistic conjunction strategy. The *intersection* of ptv_1 and ptv_2 under $\otimes$, denoted $ptv_1 \cap_\otimes ptv_2$, is the probabilistic tuple value ptv over $\mathbf{A}$ defined by $ptv.A = ptv_1.A \otimes ptv_2.A$ for all $A \in \mathbf{A}$.

Definition 7.27 (*Intersection of POB Instances*). Let $\mathbf{I}_1 = (\pi_1, \nu_1)$ and $\mathbf{I}_2 = (\pi_2, \nu_2)$ be two POB instances over the same POB schema $\mathbf{S}$, and let $\otimes$ be a probabilistic conjunction strategy. The *intersection* of $\mathbf{I}_1$ and $\mathbf{I}_2$ under $\otimes$, denoted $\mathbf{I}_1 \cap_\otimes \mathbf{I}_2$, is the POB instance (π, ν) over $\mathbf{S}$, where:

—$\pi(c) = \pi_1(c) \cap \pi_2(c)$;

—$\nu(o) = \nu_1(o) \cap_\otimes \nu_2(o)$.

The union of two POB instances is defined in the same spirit as their intersection.

Definition 7.28 (*Disjunction Strategies on Probabilistic Triples*). Let $pt_1 = (V', \alpha', \beta')$, $pt_2 = (V'', \alpha'', \beta'')$ be probabilistic triples, and let $\oplus$ be a probabilistic disjunction strategy. Then, $pt_1 \oplus pt_2$ is the probabilistic triple $pt = (V, \alpha, \beta)$, where:

—$V = V' \cup V''$;

$$—\alpha(v), \beta(v) = \begin{cases} [\alpha'(v), \beta'(v)] & \text{if } v \in V' - V'' \\ [\alpha''(v), \beta''(v)] & \text{if } v \in V'' - V' \\ [\alpha'(v), \beta'(v)] \oplus [\alpha''(v), \beta''(v)] & \text{if } v \in V' \cap V''. \end{cases}$$

As in the case of conjunction, the outcome pt of $pt_1 \oplus pt_2$ is only defined if pt is consistent, which requires that $\sum_{v \in V} \alpha(v) \leq 1$. A violation of this condition indicates incorrect data or improper application of the disjunction strategy $\oplus$. Again, this is flagged as an error.

Definition 7.29 (*Union of Probabilistic Tuple Values*). Let ptv_1 and ptv_2 be two probabilistic tuple values over the same set of attributes $\mathbf{A}$, and let $\oplus$ be a probabilistic disjunction strategy. The *union* of ptv_1 and ptv_2 under $\oplus$, denoted $ptv_1 \cup_\oplus ptv_2$, is the probabilistic tuple value ptv over $\mathbf{A}$ defined by $ptv.A = ptv_1.A \oplus ptv_2.A$ for all $A \in \mathbf{A}$.

Definition 7.30 (*Union of POB Instances*). Let $\mathbf{I}_1 = (\pi_1, \nu_1)$ and $\mathbf{I}_2 = (\pi_2, \nu_2)$ be two POB instances over the same POB-schema $\mathbf{S}$ such that $\pi_1(c_1) \cap \pi_2(c_2) = \emptyset$ for all pairs of distinct classes $c_1, c_2 \in \mathcal{C}$. Let $\oplus$ be a probabilistic disjunction strategy. The *union* of $\mathbf{I}_1$ and $\mathbf{I}_2$ under $\oplus$, denoted $\mathbf{I}_1 \cup_\oplus \mathbf{I}_2$, is defined as the POB instance (π, ν) over $\mathbf{S}$, where:

—$\pi(c) = \pi_1(c) \cup \pi_2(c)$;

$$—\nu(o) = \begin{cases} \nu_1(o) & \text{if } o \in \pi_1(\mathcal{C}) - \pi_2(\mathcal{C}) \\ \nu_2(o) & \text{if } o \in \pi_2(\mathcal{C}) - \pi_1(\mathcal{C}) \\ \nu_1(o) \cup_\oplus \nu_2(o) & \text{if } o \in \pi_1(\mathcal{C}) \cap \pi_2(\mathcal{C}). \end{cases}$$

Finally, we consider the *difference* of two POB instances. For this, we use the notion of a difference strategy for probabilistic tuple values.

Definition 7.31 (*Difference Strategies on Probabilistic Triples*). Let $pt_1 = (V', \alpha', \beta')$, $pt_2 = (V'', \alpha'', \beta'')$ be probabilistic triples, and let $\ominus$ be a probabilistic difference strategy. Then, $pt_1 \ominus pt_2$ is the probabilistic triple $pt = (V, \alpha, \beta)$, where:

—$V = V' - \{v \in V' \cap V'' \mid [\alpha'(v), \beta'(v)] \ominus [\alpha''(v), \beta''(v)] = [0, 0]\}$;

$$—[\alpha(v), \beta(v)] = \begin{cases} [\alpha'(v), \beta'(v)] & \text{if } v \in V - V'' \\ [\alpha'(v), \beta'(v)] \ominus [\alpha''(v), \beta''(v)] & \text{if } v \in V \cap V''. \end{cases}$$

Definition 7.32 (*Difference of Probabilistic Tuple Values*). Let ptv_1 and ptv_2 be two probabilistic tuple values over the same set of attributes $\mathbf{A}$, and let $\ominus$ be a probabilistic difference strategy. The *difference* of ptv_1 and ptv_2 under $\ominus$, denoted $ptv_1 -_\ominus ptv_2$ is the probabilistic tuple value ptv over $\mathbf{A}$ defined by $ptv.A = ptv_1.A \ominus ptv_2.A$ for all $A \in \mathbf{A}$.

Definition 7.33 (*Difference of POB Instances*). Let $\mathbf{I}_1 = (\pi_1, \nu_1)$ and $\mathbf{I}_2 = (\pi_2, \nu_2)$ be POB instances over the same POB schema $\mathbf{S}$, and let $\ominus$ be a probabilistic difference strategy. The *difference* of $\mathbf{I}_1$ and $\mathbf{I}_2$ under $\ominus$, denoted $\mathbf{I}_1 -_\ominus \mathbf{I}_2$, is defined as the POB instance (π, ν) over $\mathbf{S}$, where:

—$\pi(c) = \pi_1(c)$;

$$—\nu(o) = \begin{cases} \nu_1(o) & \text{if } o \in \pi_1(\mathcal{C}) - \pi_2(\mathcal{C}) \\ \nu_1(o) -_\ominus \nu_2(o) & \text{if } o \in \pi_1(\mathcal{C}) \cap \pi_2(\mathcal{C}). \end{cases}$$

7.6 Consistency Preservation

We now prove that all operations of our POB algebra that produce new POB schemas preserve consistency. Specifically, given consistent POB schemas as input, the operations of projection, renaming, Cartesian product, and join always produce a consistent POB schema as output. This is shown by the following theorem.

THEOREM 7.1. *Let* $\mathbf{S}$, $\mathbf{S}_1$, *and* $\mathbf{S}_2$ *be POB schemas. Let* $\mathbf{S}_1$ *and* $\mathbf{S}_2$ *be Cartesian-product-compatible in* (*c*) *and join-compatible in* (*d*). *Let* $\mathbf{A}$ *be a set of attributes, and let* N *be a renaming expression.*

(*a*) *If* $\mathbf{S}$ *is consistent, then* $\mathbf{\Pi_A}(\mathbf{S})$ *is consistent.*

(*b*) *If* $\mathbf{S}$ *is consistent, then* $\delta_N(\mathbf{S})$ *is consistent.*

(*c*) *If* $\mathbf{S}_1$ *and* $\mathbf{S}_2$ *are consistent, then* $\mathbf{S}_1 \times \mathbf{S}_2$ *is consistent.*

(*d*) *If* $\mathbf{S}_1$ *and* $\mathbf{S}_2$ *are consistent, then* $\mathbf{S}_1 \bowtie \mathbf{S}_2$ *is consistent.*

It is worth noting that in the POB algebra, users may express queries that are sometimes "internally inconsistent." For instance, a user may ask a selection query involving the selection condition $(x.\mathsf{soil} = \mathsf{loamy} \otimes_{me} x.\mathsf{soil} = \mathsf{loamy})[0.3, 0.7]$. Here, the user is selecting objects that have loamy soil assuming mutual exclusion of two identical selection expressions ! Clearly, this query does not make sense. Similarly, a query involving the selection condition $(x.\mathsf{rain} < 10 \ \otimes_{in} \ x.\mathsf{rain} > 20)[0.3, 0.7]$ does not make sense—one cannot assume independence of rain being less than 10 and over 20 ! Determining what queries are "safe" with respect to such probabilistic intuitions is a major challenge that will be addressed in a future paper.

8. POB ALGEBRA: EQUIVALENCE RESULTS

In this section, we derive some results on equivalences that hold in our POB algebra. We focus here on equivalences similar to well-known equivalences in the context of classical relational algebra. The list of equivalences is by no means complete, but shows that query optimization in our POB algebra is possible along similar lines as in classical relational algebra [Abiteboul et al. 1995]. Our first result says that selections may be reordered.

THEOREM 8.1. *Let* $\mathbf{I} = (\pi, \nu)$ *be a POB instance over the POB schema* $\mathbf{S}$. *Let* ϕ_1 *and* ϕ_2 *be two selection conditions. Then*

$$\sigma_{\phi_1}(\sigma_{\phi_2}(\mathbf{I})) = \sigma_{\phi_2}(\sigma_{\phi_1}(\mathbf{I})) \ = \ \sigma_{\phi_1 \wedge \phi_2}(\mathbf{I}), \tag{1}$$

where the last expression assumes that ϕ_1 *and* ϕ_2 *have the same object variable.*

Our next result says two things: that projections may be reordered, and that projections may be pushed through selections under appropriate conditions.

THEOREM 8.2. *Let* $\mathbf{I}$ *be a POB instance over the POB schema* $\mathbf{S}$. *Let* $\mathbf{A}$ *and* $\mathbf{B}$ *be sets of attributes, and let* ϕ *be a selection condition in which all path expressions start with attributes from* $\mathbf{A}$. *Then,*

$$\Pi_{\mathbf{A}}(\Pi_{\mathbf{B}}(\mathbf{I})) = \Pi_{\mathbf{B}}(\Pi_{\mathbf{A}}(\mathbf{I})) \tag{2}$$

$$\Pi_{\mathbf{A}}(\sigma_\phi(\mathbf{I})) = \sigma_\phi(\Pi_{\mathbf{A}}(\mathbf{I}))\,. \tag{3}$$

Note that, for example, for $\mathbf{A} \subseteq \mathbf{B}$, Equation (2) reduces to $\Pi_{\mathbf{A}}(\Pi_{\mathbf{B}}(\mathbf{I})) = \Pi_A(\mathbf{I})$, since $\Pi_{\mathbf{B}}(\Pi_{\mathbf{A}}(\mathbf{I})) = \Pi_{\mathbf{A} \cap \mathbf{B}}(\Pi_{\mathbf{A}}(\mathbf{I})) = \Pi_{\mathbf{A}}(\mathbf{I})$ (see Definition 7.10).

The next result, which states that selections and projections can be pushed through the renaming operator, requires some notation. For any renaming expression $N : \vec{\mathbf{B}} \leftarrow \vec{\mathbf{C}}$, the *inverse* of N, denoted by N^{-1}, is the renaming expression $\vec{\mathbf{C}} \leftarrow \vec{\mathbf{B}}$. Furthermore, the notation $\delta_N(X)$ stands for the result of applying the renaming specified by N on the formal object X.

THEOREM 8.3. *Let* $\mathbf{I}$ *be a POB instance over the POB schema* $\mathbf{S}$*, and let* N *be a renaming expression for* $\mathbf{S}$*. Let* ϕ *be a selection condition and let* A *be a set of attributes. Then*

$$\sigma_\phi(\delta_N(\mathbf{I})) = \delta_N(\sigma_{\delta_{N^{-1}}(\phi)}(\mathbf{I})) \tag{4}$$

$$\Pi_{\mathbf{A}}(\delta_N(\mathbf{I})) = \delta_N(\Pi_{\delta_{N^{-1}}(\mathbf{A})}(\mathbf{I})). \tag{5}$$

The following theorem shows that joins are always associative and commutative, regardless of what conjunction strategy is used in the join. In addition, selects may be pushed "through" a join by appropriately splitting the selection condition and the same is true of projections.

THEOREM 8.4. *Let* $\mathbf{S}_1$, $\mathbf{S}_2$, *and* $\mathbf{S}_3$ *be pairwise join-compatible POB schemas and let* $\mathbf{I}_1, \mathbf{I}_2$, *and* $\mathbf{I}_3$ *be POB instances over* $\mathbf{S}_1, \mathbf{S}_2$, *and* $\mathbf{S}_3$, *respectively. Let* $\otimes$ *be a probabilistic conjunction strategy. Let* ϕ_1, ϕ_2, *and* ϕ_3 *be selection conditions such that* ϕ_1 *and* ϕ_2 *involve only attributes from* $\mathbf{A}_1 - \mathbf{A}_2$ *and* $\mathbf{A}_2 - \mathbf{A}_1$, *respectively, where* $\mathbf{A}_1$ *and* $\mathbf{A}_2$ *denote the sets of top-level attributes of* $\mathbf{S}_1$ *and* $\mathbf{S}_2$, *respectively. Let* $\mathbf{B}$ *be a set of attributes and define* $\mathbf{B}_1 = (\mathbf{B} \cup \mathbf{A}_2) \cap \mathbf{A}_1$ *and* $\mathbf{B}_2 = (\mathbf{B} \cup \mathbf{A}_1) \cap \mathbf{A}_2$. *Then*

$$\mathbf{I}_1 \bowtie_\otimes \mathbf{I}_2 = \mathbf{I}_2 \bowtie_\otimes \mathbf{I}_1 \tag{6}$$

$$(\mathbf{I}_1 \bowtie_\otimes \mathbf{I}_2) \bowtie_\otimes \mathbf{I}_3 = \mathbf{I}_1 \bowtie_\otimes (\mathbf{I}_2 \bowtie_\otimes \mathbf{I}_3) \tag{7}$$

$$\sigma_{\phi_1 \wedge \phi_2 \wedge \phi_3}(\mathbf{I}_1 \bowtie_\otimes \mathbf{I}_2) = \sigma_{\phi_3}\big(\sigma_{\phi_1}(\mathbf{I}_1) \bowtie_\otimes \sigma_{\phi_2}(\mathbf{I}_2)\big) \tag{8}$$

$$\Pi_{\mathbf{B}}(\mathbf{I}_1 \bowtie_\otimes \mathbf{I}_2) = \Pi_{\mathbf{B}}\big(\Pi_{\mathbf{B}_1}(\mathbf{I}_1) \bowtie_\otimes \Pi_{\mathbf{B}_2}(\mathbf{I}_2)\big). \tag{9}$$

Note that in classical relational databases, Equivalence (8) remains true if ϕ_1 and ϕ_2 access common attributes of $\mathbf{A}_1$ and $\mathbf{A}_2$. This is no longer guaranteed for POBs, as the join may change the value of common attributes. As the Cartesian product is a special case of join, we obtain the following corollary to Theorem 8.4.

COROLLARY 8.5. *Let* $\mathbf{S}_1$, $\mathbf{S}_2$, *and* $\mathbf{S}_3$ *be pairwise Cartesian-product-compatible POB schemas and let* $\mathbf{I}_1, \mathbf{I}_2$, *and* $\mathbf{I}_3$ *be POB instances over* $\mathbf{S}_1, \mathbf{S}_2$, *and* $\mathbf{S}_3$, *respectively. Let* ϕ_1, ϕ_2, *and* ϕ_3 *be selection conditions such that* ϕ_1 *and* ϕ_2 *involve only attributes from the sets of top-level attributes* $\mathbf{A}_1$ *and* $\mathbf{A}_2$ *of* $\mathbf{S}_1$ *and* $\mathbf{S}_2$*, respectively. Let* $\mathbf{B}$ *be a set of attributes and let* $\mathbf{B}_1 = \mathbf{B} \cap \mathbf{A}_1$ *and* $\mathbf{B}_2 = \mathbf{B} \cap \mathbf{A}_2$. *Then*

$$\mathbf{I}_1 \times \mathbf{I}_2 = \mathbf{I}_2 \times \mathbf{I}_1 \tag{10}$$

$$(\mathbf{I}_1 \times \mathbf{I}_2) \times \mathbf{I}_3 = \mathbf{I}_1 \times (\mathbf{I}_2 \times \mathbf{I}_3) \tag{11}$$

$$\sigma_{\phi_1 \wedge \phi_2 \wedge \phi_3}(\mathbf{I}_1 \times \mathbf{I}_2) = \sigma_{\phi_3}\big(\sigma_{\phi_1}(\mathbf{I}_1) \times \sigma_{\phi_2}(\mathbf{I}_2)\big) \tag{12}$$

$$\Pi_{\mathbf{B}}(\mathbf{I}_1 \times \mathbf{I}_2) = \Pi_{\mathbf{B}_1}(\mathbf{I}_1) \times \Pi_{\mathbf{B}_2}(\mathbf{I}_2). \tag{13}$$

THEOREM 8.6. *Let* $\mathbf{I}_1$*,* $\mathbf{I}_2$*, and* $\mathbf{I}_3$ *be POB instances over the same POB schema* $\mathbf{S}$. *Let* $\otimes/\oplus/\ominus$ *be a probabilistic conjunction/disjunction/difference strategy and let* $\mathbf{A}$ *be a set of attributes. Then,*

$$\mathbf{I}_1 \cap_{\otimes} \mathbf{I}_2 = \mathbf{I}_2 \cap_{\otimes} \mathbf{I}_1 \tag{14}$$

$$(\mathbf{I}_1 \cap_{\otimes} \mathbf{I}_2) \cap_{\otimes} \mathbf{I}_3 = \mathbf{I}_1 \cap_{\otimes} (\mathbf{I}_2 \cap_{\otimes} \mathbf{I}_3) \tag{15}$$

$$\mathbf{I}_1 \cup_{\oplus} \mathbf{I}_2 = \mathbf{I}_2 \cup_{\oplus} \mathbf{I}_1 \tag{16}$$

$$(\mathbf{I}_1 \cup_{\oplus} \mathbf{I}_2) \cup_{\oplus} \mathbf{I}_3 = \mathbf{I}_1 \cup_{\oplus} (\mathbf{I}_2 \cup_{\oplus} \mathbf{I}_3) \tag{17}$$

$$\Pi_{\mathbf{A}}(\mathbf{I}_1 \cap_{\otimes} \mathbf{I}_2) = \Pi_{\mathbf{A}}(\mathbf{I}_1) \cap_{\otimes} \Pi_{\mathbf{A}}(\mathbf{I}_2) \tag{18}$$

$$\Pi_{\mathbf{A}}(\mathbf{I}_1 \cup_{\oplus} \mathbf{I}_2) = \Pi_{\mathbf{A}}(\mathbf{I}_1) \cup_{\oplus} \Pi_{\mathbf{A}}(\mathbf{I}_2) \tag{19}$$

$$\Pi_{\mathbf{A}}(\mathbf{I}_1 -_{\ominus} \mathbf{I}_2) = \Pi_{\mathbf{A}}(\mathbf{I}_1) -_{\ominus} \Pi_{\mathbf{A}}(\mathbf{I}_2). \tag{20}$$

Note that literally taken, Equations (18) and (20) are not true for relational databases. The reason is that we use oids for objects in POBs, while relational databases only contain values.

Finally, we remark that Equations (2), (5), (10), (11), and (13) are actually unrelated to probabilities (since the operations projection, renaming, and Cartesian product are unrelated to probabilities).

9. IMPLEMENTATION

We have implemented a prototype distributed POB system. The server (POB server) runs on top of ObjectStore and is implemented in SUN-C++ A thin client for handling database transactions is implemented using GNU-C++.

9.1 POB Server

The POB server is a collection manager of POB schemas. Each POB schema consists of a set of POB classes and their associated POB object instances. The POB server manages (i) persistent schemas, which correspond to permanent data and (ii) temporary schemas, which maintain intermediate schemas.

The *probability interpreter* contains functions for computing probabilistic conjunction and disjunction strategies. It also contains a library of distribution functions for manipulating probabilistic tuple values associated with objects in the database.

The *POB schema class* maintains an inheritance probability table (the probability assignment $\wp$ in Definition 5.4). The class contains methods to add, remove, and retrieve POB classes and POB objects. In addition, given two classes c_1 and c_2, where c_1 is a subclass of c_2, there is a method that computes the conditional probability that an arbitrary object belongs to c_1 given that it belongs to c_2.

POB classes are objects in the POB schema class. They have a name, a collection of attributes (with associated types), and a collection of parent POB class names along with associated probability assignments. Methods associated with POB classes provide abilities to establish attribute/type information, parent POB class/probability assignments, adding and removing POB objects from the POB class, and various self-replicating functions that are useful for query processing.

POB objects contain an object name, the oid, a collection of probabilistic tuple values, and a POB class pointer which points to the POB class of which it is an instance. The POB class pointer is provided for fast access to class-level information: attributes, types, parents, and the like. Methods associated with POB objects include functions for setting probabilistic tuple values and various self-replicating functions to facilitate query processing.

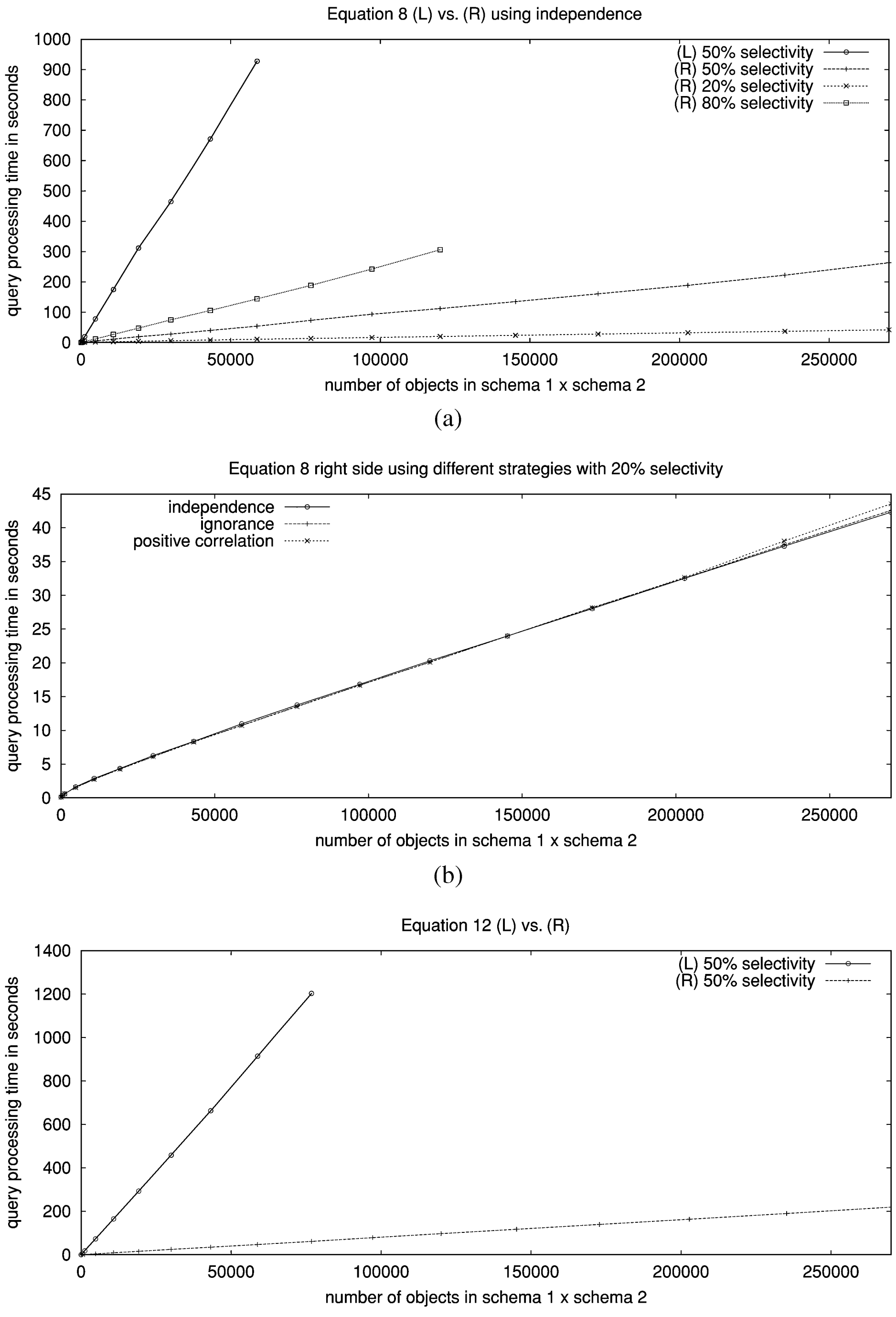
The *POB server* handles client requests. It contains a pointer to an ObjectStore database which provides persistence services. The POB server includes methods for: connecting to a database, disconnecting from a database, creating and removing schemas, creating and removing classes, creating and removing objects, computing the probability that an object is a member of class c_1 given that it is a member of class c_2, computing the probabilistic extent of a class, checking if an object satisfies a given selection condition, executing an arbitrary query in the probabilistic object algebra, and a variety of printing functions.

Note that each method may not correspond to a logical unit of work—in this case a request. In some instances, several requests are handled within one method while in other instances, a single request is handled through a combination of methods.

9.2 Experiments

Using the POB server, we have conducted a set of experiments to assess the various equivalences described in Section 8 as well as to assess the performance of selection. We do not describe all the experiments we conducted (due to space reasons), but only a few sample experiments are listed below. The limiting factor in all experiments was the size of the "largest intermediate schema." This is the number of objects in the largest schema encountered when executing the query. For a selection query, this is just the number of objects in the POB instance on which the selection is performed. In the case of a join/Cartesian product, this is the product of the sizes of the POB instances being joined (or whose Cartesian product is being computed). In the experiments involving Equations (8), (12), and (13) described below, we varied the number of objects in the largest intermediate schema from 0 to 270,000 objects and measured the times taken (on a Sun Ultra 10 workstation) for both the left and right sides of the rewrite rules in question. The idea was to see whether the left side of a rule should be rewritten to the right side or the other way as round.

Effectiveness of Equation (8). Our first experiment evaluated the effectiveness of pushing selections into joins (Theorem 8.4). Figure 7(a) shows what happens if the selectivity is varied using independence. It is easy to see that the



(a)

(b)

(c)

Fig. 7. Experimental results.

right side of this equation pays off in a huge way and that as the selectivity decreases (i.e., fewer and fewer objects are selected), more and more objects can be efficiently handled. For instance, with 20% selectivity, 270,000 objects in the largest intermediate schema can be computed in about 30 seconds.

Figure 7(b) shows the effect of evaluating the right side of Equation (8) with different probabilistic strategies. We see that precisely which strategy is used has very little impact on the computation time.

Effectiveness of Equation (12). We conducted experiments similar to those described above with Equation (12). Figure 7(c) shows the result of testing it shows that pushing selections into a Cartesian product may save up to 80 to 90% of the time and this savings increases as the number of objects increases.

Effectiveness of Equation (13). We conducted experiments similar to those described above with Equation (13). Figure 8(a) shows the result of testing: it shows that pushing projections into a Cartesian product does not help very much. The reason for this is because projection does not reduce the number of objects.

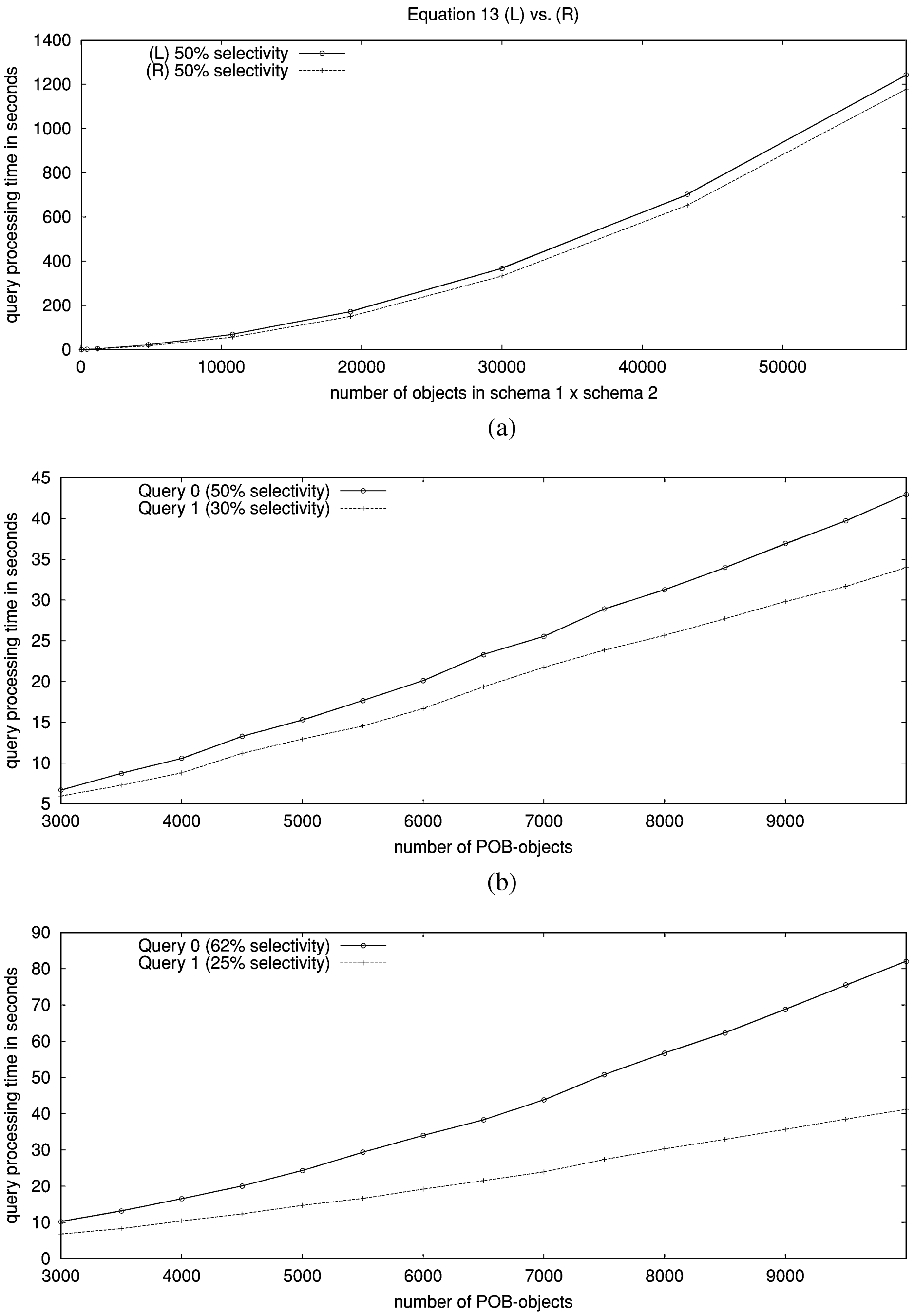
Effectiveness of Selection. We also conducted some experiments on the effectiveness of selection on POB schemas of sizes between 3,000 and 10,000 objects. In the experiment, we executed queries of the form "Select x from schema e where $x.D > val$." Figure 8(b) shows the result when two different selectivities are used: 50% (i.e., half the objects satisfy the selection condition) and 30% (i.e., 30% of the objects satisfy the selection condition). We also tested what happens when we consider *membership* selection queries of the form "Select x from schema e where (x is a member of class C)." Figure 8(c) shows the result of this query with two different selectivities. Note that the queries generally exhibit linear behavior with respect to the number of objects. In addition, membership queries are computationally more expensive than simple inequality queries.

We are part way towards developing a POB query optimizer. While the equivalence results of Section 8 readily serve as rewrite rules, the problem of developing a cost model is a challenge on which we are currently working. Once cost models for POBs are developed, a CASCADES [Graefe 1995] style framework may be readily used for query optimization. We are currently working on this problem.

10. RELATED WORK

Our work has been inspired by the prior work of Kornatzky and Shimony [1994] who describe a probabilistic object-oriented data model in which, as in our approach, uncertainty in the values of attributes and in the class graph may be represented by probabilities. The main differences between Kornatzky and Shimony [1994] and our approach can be briefly summarized as follows.

1. Kornatzky and Shimony introduce an object calculus for extracting objects from probabilistic object-oriented databases. This calculus can thus be compared to our selection operation. It is more restrictive in the sense that it only handles probabilities on atomic formulas (which always evaluate to either true or false), while our selection operation also handles probabilities



(a)

(b)

(c)

Fig. 8. Experimental results.

on conjunctions and disjunctions of atomic formulas, using probabilistic conjunction and disjunction strategies. Specifically, we make no independence assumption (as Kornatzky and Shimony do). On the other hand, their object calculus has quantifiers, which our selection operation does not include. However, it could be easily extended in this direction.

2. We also discuss, in detail, the algebraic operations of projection, renaming, Cartesian product, join, selection, union, intersection, and difference. As they were developing a calculus, Kornatzky and Shimony do not deal with this.
3. We introduce, for the first time, results on query equivalences in probabilistic object bases, and to our knowledge, our system is the first implementation of a probabilistic object base.
4. Kornatzky and Shimony assume that the class graph is a directed tree *without multiple inheritance*. Moreover, incomparable classes are always disjoint. In contrast, in our approach, the class graph may be any directed acyclic graph, thus allowing multiple inheritance. Furthermore, the disjointness of classes can be expressed in a flexible way by grouping them into partition clusters. The consistency of schema declarations is guaranteed for a large subclass extending directed trees.
5. Kornatzky and Shimony assume a precise probability distribution on the set of all possible values of an attribute (including a null value $\perp$ that represents the inapplicability of an attribute). Our approach, in contrast, just requires an interval range for probability distributions. Furthermore, objects occurring as attribute values are given special treatment in Kornatzky and Shimony [1994]; our model can be extended in this respect.
6. In Kornatzky and Shimony [1994], the probabilistic extent of a class is derived from statistical and subjective probabilities. Since, in general, inconsistency may arise, the notion of *cutsets* of classes is introduced there. The probabilistic extent of a class is then given by statistical probabilities in the class hierarchy and by subjective probabilities with respect to a cutset. Our probabilistic extent, in contrast, is just derived from statistical probabilities and classical class membership. We thus avoid all the problems that come along with mixing up statistical and subjective probabilities.

Sadri [1994] describes how to extend object-oriented databases by using the *information source tracking method*, in which every piece of information is assigned a vector of confirming information sources. The formalism is based on a nonprobabilistic lattice structure, but Sadri mentions a possible extension by numerical and especially probabilistic uncertainty. He models uncertainty on the attribute, object, and class level, which roughly relates to our probabilistic attributes, to our interpretation of selection expressions, and to our probabilistic extents, respectively.

In the area of uncertainty in AI, there is related work on object-oriented Bayesian networks by Koller and Pfeffer [1997] and by Laskey and Mahoney [1997], and on constructing Bayesian networks from first-order probabilistic knowledge bases by Haddawy [1994]. The main idea behind object-oriented

Bayesian networks is to use methods from object-oriented programming languages in order to enable flexible and large-scale knowledge representation with Bayesian networks. The objects in this framework are given by Bayesian network fragments. Objects that share common features are grouped together into classes, which are organized along inheritance hierarchies. Query processing in object-oriented Bayesian networks is essentially reduced to a form of Bayesian network inference that exploits some locality aspects of the object-oriented modeling for increased efficiency. Haddawy [1994] aims at a first-order generalization of Bayesian networks. More precisely, he describes how queries to first-order probabilistic knowledge bases satisfying certain constraints can be translated into Bayesian network inference problems.

A step towards the model proposed in the present article is an extension of the relational model allowing complex values [Eiter et al. 2000a,b] with probabilities. However, the model in Eiter et al. [2000a,b] has no class hierarchy and, in particular, inheritance is not addressed. Thus, it has no features of an object-oriented system, and is essentially in the group of probabilistic relational database models, which we discuss next.

ProbView [Lakshmanan et al. 1997] is a probabilistic relational database model that generalizes various approaches (such as Brbara et al. [1992] and Cavallo and Pittarelli [1987]). Cavallo and Pittarelli's [1987] important paper views relations in a (flat) relational database as probability distribution functions, where tuples in the same relation are viewed as pairwise disjoint events whose probabilities sum up to 1. Drawbacks of this approach have been pointed out in Dey and Sarkar [1996]. An extension of the model using probability intervals, which are viewed as constraints on the probabilities, is reviewed in Pittarelli [1994]. Barbara et al. [1992] consider a probabilistic extension to the relational model, in which imprecise attributes are modeled as probability distributions over finite sets of values. No probabilities can be assigned to outmost tuples. Their approach assumes that key attributes are deterministic (have probability 1) and that nonkey attributes in different relations are independent. As pointed out in Barbara et al. [1992], "lossy" joins are possible in this model.

Another important probabilistic database model is that of Dey and Sarkar [1996], which assigns each tuple in a (flat) relational database a probability value in a special attribute. Based on Dey and Sarkar [1996], a probabilistic extension to SQL is developed in Dey and Sarkar [1998]. The classical relational operations are defined in Dey and Sarkar [1996] adopting different assumptions on the relationship between tuples; in particular, join assumes independence, union and difference assume positive correlation, and compaction assumes disjointness or positive correlation. Our model is far more general.

Fuhr and Rölleke [1996] consider a probabilistic version of NF2 relations, extending their approach for flat tuples [Fuhr and Rölleke 1997], and define a relational algebra for this model. Probabilities are assigned to tuples and to values of nested tuples (i.e., set-valued attributes), which are viewed as events that have an associated event expression. The algebraic operators manipulate tuples by combining value and event expressions appropriately. An intensional semantics is developed in Fuhr and Rölleke [1996] in which probabilities are defined through possible worlds. The evaluation method assumes that in nondeterministic relations (i.e., relations with uncertain tuples), joint occurrence of two different values is either always independent or impossible—this is certainly restrictive.

Dyreson and Snodgrass [1998] provide a version of SQL to handle temporal indeterminacy, where there is uncertainty about when an event occurs. They use a relational framework and focus on the important case where the space of values over which uncertainty exists is huge.

Kiessling and his group [1992] developed a framework called DUCK for reasoning with uncertainty. They provide an elegant, logical, axiomatic theory for uncertain reasoning in the presence of rules. In contrast, in our framework, rules are not present; rather, our interest is in extending object database models to handle uncertainty in an algebraic setting.

In an important paper, Lakshmanan and Sadri [1994b] show how selected probabilistic strategies can be used to extend the previous probabilistic models. Lakshmanan and Shiri [1996] show how deductive databases may be parameterized through the use of conjunction and disjunction strategies, an approach also followed by Dekhtyar and Subrahmanian [1997]. We have built in this article upon the important concept of probabilistic conjunction and disjunction strategies, but in an object-oriented instead of a logic programming setting.

11. CONCLUSION

In this article, we proposed an extension of the relational algebra to handle probabilistic modes of uncertainty in object-oriented database systems. More precisely, the main contributions of this article can be briefly summarized as follows.

1. We presented a formal definition of a probabilistic object base, which extends previous definitions given by Kornatzky and Shimony [1994].
2. We gave a formal model-theoretic basis for discussing the consistency of POBs, and showed that consistency checking is NP-complete in general. We then defined classes of POBs for which consistency can be checked in polynomial time, and provided efficient algorithms for this task.
3. We developed an algebra that extends the relational algebra to probabilistic object bases. Specifically, this algebra recognizes that probabilities of complex events depend on existing knowledge about dependencies between events, and hence, it allows users to express algebraic queries under appropriate conjunction, disjunction, and difference strategies (which encode such dependence information).
4. We presented a number of equivalence results that may form a set of rewrite rules to be used in query optimization.
5. Our POB framework has been implemented on top of ObjectStore in C++.
6. Finally, we conducted a set of experiments on the efficacy of our equivalence results for query rewriting (and hence for query optimization).

Several tasks remain for further work. One is the enhancement of the current prototype by a sophisticated POB algebra query manager, which optimizes queries by using cost models and rewrite rules as shown in Figure 2. For the front end of the system, it would be well worth developing a probabilistic version of SQL (similar to, for example, Dey and Sarkar's [1998] language PSQL).

Another important task for further work is to develop a model-theoretic semantics for probabilistic attribute values. Specifically, we are planning to map the probabilistic knowledge in top-level attributes into a language in probabilistic logic, which can be interpreted by probability distributions over a set of possible worlds.

A further topic of future research is to generalize our model. For example, one could think about allowing arbitrary sets of probability distributions [Kyburg, Jr. and Pittarelli 1996] as values of top-level attributes (and not just convex sets of distributions that are specified by probabilistic triples), or about allowing negations besides in selection conditions also, at a lower level, in selection expressions. It would also be interesting to allow objects as attribute values.

Finally, another important topic is to explore how to perform updates on probabilistic attribute values of existing objects in POBs.

ACKNOWLEDGMENTS

We are very grateful to the anonymous referees for their many constructive comments and helpful suggestions for improvements.

REFERENCES

ABITEBOUL, S., HULL, R., AND VIANU, V. 1995. *Foundations of Databases*. Addison-Wesley, Reading, Mass.

ATKINSON, M., DEWITT, D., MAIER, D., BANCILHON, F., DITTRICH, K., AND ZDONIK, S. 1989. The object-oriented database system manifesto. In *Proceedings of DOOD-89*, Elsevier Science, Amsterdam, the Netherlands, 40–57.

BANCILHON, F., DELOBEL, C., AND KANELLAKIS, P. (Eds.) 1991. *Building an Object-Oriented Database System: The Story of O_2*. Morgan Kaufmann, Los Altos, Calif.

BARBARA, D., GARCIA-MOLINA, H., AND PORTER, D. 1992. The management of probabilistic data. *IEEE Trans. on Knowl. Data Eng. 4*, 5, 387–502.

BERTINO, E. AND MARTINO, L. 1993. *Object Oriented Database Systems: Concepts and Architectures*. Addison-Wesley, Wokingham, UK.

BOOLE, G. 1854. *The Laws of Thought*. Macmillan, London.

CAVALLO, R. AND PITTARELLI, M. 1987. The theory of probabilistic databases. In *Proceedings of VLDB-87*, Morgan Kaufmann, Los Altos, Calif., 71–81.

DEKHTYAR, A. AND SUBRAHMANIAN, V. S. 1997. Hybrid probabilistic programs. In *Proceedings of the Fourteenth International Conference on Logic Programming (ICLP '97),* MIT Press, Cambridge, Mass., 391–405.

DEY, D. AND SARKAR, S. 1996. A probabilistic relational model and algebra. *ACM Trans. Database Syst. 21*, 3, 339–369.

DEY, D. AND SARKAR, S. 1998. PSQL: A query language for probabilistic relational data. *Data Knowl. Eng. 28*, 107–120.

DYRESON, C. AND SNODGRASS, R. 1998. Supporting valid-time indeterminacy. *ACM Trans. Database Syst. 23*, 1, 1–57.

EITER, T., LU, J. J., LUKASIEWICZ, T., AND SUBRAHMANIAN, V. S. 1999. Probabilistic object bases. Tech. Rep. INFSYS RR-1843-99-11, Institut für Informationssysteme, Technische Universität Wien.

EITER, T., LUKASIEWICZ, T., AND WALTER, M. 2000a. A data model and algebra for probabilistic complex values. Tech. Rep. INFSYS RR-1843-00-04, Institut für Informationssysteme, Technische Universität Wien.

EITER, T., LUKASIEWICZ, T., AND WALTER, M. 2000b. Extension of the relational algebra to probabilistic complex values. In *Proceedings of the International Symposium on Foundations of*

Information and Knowledge Systems (*FoIKS 2000*), Lecture Notes on Computer Science, vol. 1762, Springer-Verlag, New York, 94–115.

Fagin, R., Halpern, J. Y., and Megiddo, N. 1990. A logic for reasoning about probabilities. *Inf. Comput. 87*, 78–128.

Fuhr, N. and Rölleke, T. 1996. A probabilistic NF2 relational algebra for integrated information retrieval and database systems. In *Proceedings of the Second World Conference on Integrated Design and Process Technology*, Society for Design and Process Science, 17–30.

Fuhr, N. and Rölleke, T. 1997. A probabilistic relational algebra for the integration of information retrieval and database systems. *ACM Trans. Inf. Syst. 15*, 1, 32–66.

Garey, M. R. and Johnson, D. S. 1979. *Computers and Intractability*: *A Guide to the Theory of NP-Completeness*. Freeman, San Francisco.

Graefe, G. 1995. The cascades framework for query optimization. *Data Eng. Bull. 18*, 3, 19–29.

Grosky, W. I., Jain, R., and Mehrotra, R. (Eds.) 1997. *The Handbook of Multimedia Information Management*. Prentice-Hall, Englewoood Cliffs, NJ.

Güntzer, U., Kiessling, W., and Thöne, H. 1991. New directions for uncertainty reasoning in deductive databases. In *Proceedings of ACM SIGMOD '91*, ACM Press, New York, 178–187.

Haddawy, P. 1994. Generating Bayesian networks from probability logic knowledge bases. In *Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence* (*UAI-94*), Morgan Kaufmann, Los Altos, Calif., 262–269.

Halpern, J. Y. 1990. An analysis of first-order logics of probability. *Artif. Intell. 46*, 3, 311–350.

Kiessling, W., Thöne, H., and Güntzer, U. 1992. Database support for problematic knowledge. In *Proceedings of EDBT-92*, Lecture Notes in Computer Science, vol. 580, Springer-Verlag, New York, 421–436.

Kifer, M. and Li, A. 1988. On the semantics of rule-based expert systems with uncertainty. In *Proceedings of ICDT-88*, Lecture Notes in Computer Science, vol. 326, Springer-Verlag, New York, 102–117.

Kim, W. 1990. *Introduction to Object-Oriented Databases*. MIT Press, Cambridge, Mass.

Koller, D. and Pfeffer, A. 1997. Object-oriented Bayesian networks. In *Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence* (*UAI-97*), Morgan Kaufmann, Los Altos, Calif., 302–313.

Kornatzky, Y. and Shimony, S. E. 1994. A probabilistic object-oriented data model. *Data Knowl. Eng. 12*, 143–166.

Kyburg, Jr., H. E. and Pittarelli, M. 1996. Set-based Bayesianism. *IEEE Trans. Syst. Man Cybern. A*: *Syst. Humans 26*, 3, 324–339.

Lakshmanan, L. V. S. and Sadri, F. 1994a. Modeling uncertainty in deductive databases. In *Proceedings of DEXA-94*, vol. 856, Springer-Verlag, New York, 724–733.

Lakshmanan, L. V. S. and Sadri, F. 1994b. Probabilistic deductive databases. In *Proceedings of the 1994 International Logic Programming Symposium* (*ILPS '94*), MIT Press, Cambridge, Mass., 254–268.

Lakshmanan, L. V. S. and Shiri, N. 1996. A parametric approach to deductive databases with uncertainty. In *Proceedings of the International Workshop on Logic in Databases* (*LID '96*), Lecture Notes in Computer Science, vol. 1154, Springer-Verlag, New York, 61–81.

Lakshmanan, L. V. S., Leone, N., Ross, R., and Subrahmanian, V. S. 1997. ProbView: A flexible probabilistic database system. *ACM Trans. Database Syst. 22*, 419–469.

Laskey, K. B. and Mahoney, S. M. 1997. Network fragments: Representing knowledge for constructing probabilistic models. In *Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence* (*UAI-97*), Morgan-Kaufmann, Los Altos, Calif., 334–341.

Pittarelli, M. 1994. An algebra for probabilistic databases. *IEEE Trans. Knowl. Data Eng. 6*, 2, 293–303.

Sadri, F. 1994. Modeling uncertainty in object-oriented databases. In *Proceedings of the Workshop on Incompleteness and Uncertainty in Information Systems* (*IUIS'93*), Workshops in Computing, Springer-Verlag, New York, 56–68.

Shaw, G. M. and Zdonik, S. B. 1990. A query algebra for object-oriented databases. In *Proceedings of ICDE-90*, IEEE Computer Society, Los Alamitos, Calif., 154–161.

Received December 1999; revised September 2000; accepted February 2001