The Second
Twente Data Management Workshop
TDM'06

on

Uncertainty in Databases

edited by
Ander de Keijzer and Maurice van Keulen
University of Twente

CTIT Workshop Proceedings Series WP06-01
University of Twente

Sponsors

Centre for Telematics and Information Technology (CTIT),
University of Twente

Dutch Research School for Information and Knowledge Systems (SIKS)

Publication Details

Proceedings of the second Twente Data Management Workshop
Edited by Ander de Keijzer and Maurice van Keulen
Published by the Centre for Telematics and Information Technology (CTIT), University of Twente
CTIT Workshop Proceedings Series WP06-01
ISSN 1574-0846

An electronic version of the proceedings is available at `http://www.cs.utwente.nl/~tdm`
June 2006, Enschede, The Netherlands

Preface

Welcome to the second Twente Data Management Workshop. The topic of this second edition is Uncertainty in Databases. This topic has gained more and more attention these last years. This is clearly visible by the increasing number of workshops on, or related to this topic.

We are grateful that Dan Suciu from the university of Washington accepted our invitation to give the keynote speech. We also wish to thank the members of the Program Committee for reviewing the submitted papers. The Program Committee consists of eight people from four different countries and five different institutions.

We received eight submissions by 17 authors from 7 different countries. One paper was reviewed twice, the other seven papers where reviewed three times. We accepted six papers, resulting in an acceptance rate of 75%.

We would like to thank the CTIT for sponsoring the workshop and in particular Anja Annink for helping with the proceedings. We would also like to thank SIKS for sponsoring the participation of SIKS PhD-Students.

Finally, we hope that this workshop will contribute to the research and cooperation on Uncertainty in Databases.

To an inspiring workshop!

Enschede, May 22nd 2006

Ander de Keijzer and Maurice van Keulen

TDM'06 Organizing Committee

Ander de Keijzer (University of Twente)
Maurice van Keulen (University of Twente)

TDM'06 Program Committee

Henk Ernst Blok (University of Twente, The Netherlands)
Patrick Bosc (IRISA/ENSSAT, France)
Sunil Choenni (Research & Documentation Centre, Dutch Ministry of Justice, The Netherlands)
Maarten Fokkinga (University of Twente, The Netherlands)
Ander de Keijzer (University of Twente, The Netherlands)
Maurice van Keulen (University of Twente, The Netherlands)
Giuseppe Psaila (University of Bergamo, Italy)
Guy De Tré (Ghent University, Belgium)

Workshop Program

Tuesday, June 6th, 2006
Logica, University of Twente, Enschede, The Netherlands

09:30 Registration
10:00 Opening
Ander de Keijzer & Maurice van Keulen

Keynote speech

10:15 Managing Imprecisions with Probabilistic Databases
Dan Suciu

11:15 Coffee break

Session I: Querying with Uncertainty
11:30 About Generalized YES/NO Queries in the Possibilistic and Probabilistic Database Contexts
Patrick Bosc, Nadia Liétard, Olivier Pivert
12:00 Structured Queries for Semistructured Probabilistic Data
Alex Dekhtyar, Krol Kevin Mathias, Praveen Gutti

12:30 Lunch

Session II: Uncertainty and its Representations
14:00 Z-style notation for Probabilities
Maarten M. Fokkinga
14:30 Modeling Uncertain Context Information via Event Probabilities
Arthur H. van Bunningen, Ling Feng, Peter M.G. Apers

15:00 Poster session & coffee break

Session III: Uncertainty - uncertain
15:45 Ignorance in the Relational Model
Maarten M. Fokkinga
16:15 Integrity Checking for Uncertain Data
Hendrik Decker, Davide Martinenghi

16:45 Closing

17:00 Reception

Table of Contents

Managing Imprecisions with Probabilistic Databases

Dan Suciu
University of Washington

This talk describes research done at the University of Washington on the SQL query evaluation problem on probabilistic databases. The motivation comes from managing imprecisions in data: fuzzy object matching, information extracted from text, constraint violations. There are three dimensions to the query evaluation problem: the probabilistic data model, the complexity of the SQL queries, and whether output probabilities are exact or approximated.

In the simplest probabilistic data model every tuple t is an independent probabilistic event, whose probability p represents the probability that t belongs to the database. For example, in information extraction every fact (tuple t) extracted from the text has a probability p of being correct, and for any two tuples t, t' their probabilities are independent. Single block SQL queries without duplicate elimination can be evaluated simply by multiplying probabilities during join operations. But when duplicate elimination or other forms of aggregations are present, then the story is more complex. For some queries we can find a query plan such that independence still holds at each projection/duplicate-elimination operator, and thus evaluate the query efficiently. But other queries are #P-hard, and it is unlikely that they can be evaluated efficiently, and there is a simple criterion to distinguish between these two kinds of queries.

Moving to a slightly more complex data model, we consider the case when tuples are either independent or exclusive (disjoint). For example, in fuzzy object matching an object ``Washington U.'' in one database matches both ``University of Washington'' with probability 0.4 and ``Washington University in St. Louis'' with probability 0.3 in a second database. This can be represented by two tuples t, t' with probabilities 0.4 and 0.3, which are exclusive events. Here, too, there is a crisp separation of queries that can be evaluated efficiently and those that are #P-hard.

Finally, we considered a slightly different query semantics: rank the query's answers by their probabilities, and return only the top k answers. Thus, the exact output probabilities are not important, only their ranking, and only for the top k answers. This is justified in applications of imprecise data, where probabilities have little semantics and only the top answers are meaningful. We have found that a combination of Monte Carlo simulation with in-engine SQL query evaluation scales both with the data size and the query complexity.

About Generalized Yes/No Queries in the Possibilistic and Probabilistic Database Contexts

Patrick Bosc
IRISA-ENSSAT
Technopole Anticipa – BP 80518
22305 Lannion Cedex France
+33 2 96 46 90 00
Patrick.Bosc@enssat.fr

Nadia Liétard
IRISA-ENSSAT
Technopole Anticipa – BP 80518
22305 Lannion Cedex France
+33 2 96 46 90 00
Nadia.Liétard@enssat.fr

Olivier Pivert
IRISA-ENSSAT
Technopole Anticipa – BP 80518
22305 Lannion Cedex France
+33 2 96 46 90 00
Olivier.Pivert@enssat.fr

ABSTRACT

This paper is concerned with the handling of imprecise information in relational databases. The need for dealing with imprecise data is more and more acknowledged in order to cope with real data, even if commercial systems are most of the time unable to manage them. Here, the possibilistic and probabilistic settings are considered. In these frameworks, any imprecise piece of information is modeled as a distribution intended for constraining the more or less acceptable values. Such an imprecise database has a natural interpretation in terms of a set of regular databases which provides the basic gateway to interpret queries. However, if this approach is sound it is not realistic and it is necessary to consider restricted queries for which a calculus grounded on the possibilistic (or probabilistic) database, i.e., where the operators work directly on imprecise relations, is feasible. Extended yes/no queries are dealt with here, i.e., yes/no queries in the presence of imprecise data. Their general form is: "to what extent is it possible and certain that the answer to Q fulfills property P", where Q is an algebraic relational query. A strategy for processing such queries efficiently is proposed under some assumptions as to the operators appearing in Q. The impact of the model of uncertainty (possibilistic or probabilistic) on the complexity of the evaluation is given a special attention.

1. INTRODUCTION

The need for imperfect data has been recognized for a while in many areas, even if available tools are not always able to deal with imprecision/uncertainty and commercial database management systems illustrate the gap between needs and products. In the database domain, imperfect information can appear in diverse situations such as data warehouses, forecasts, archives (where some information has been lost or damaged and can only be partly recovered), systems managing information issued from automated recognition mechanisms (where, sometimes, several candidate objects remain).

Different formalisms can be used to represent imprecise information (see for instance [8]), but whatever this choice, in this context querying turns out to be much more complicated in terms of complexity. Indeed, an imprecise information is represented as a (possibly infinite) set of acceptable candidates and then a database with imprecise information can be seen as a set of regular databases, called worlds, associated with a choice for each attribute value. Then, a natural way of processing a query, which is semantically founded, is to evaluate it against each world. Such an approach is intractable in case of an infinite number of worlds, but also because even when it is finite, it is generally huge. This state of fact leads to consider only specific families of queries which can be processed in a "compact" fashion, while delivering a result equivalent to the one defined in terms of worlds.

In this paper, we are interested in a type of queries known as "yes/no" queries, whose general form (in a regular database context) is:

"is it true that the answer to Q fulfills condition C?"

where Q is a relational algebraic query. An example is "membership-based queries", whose form is: "is it true that tuple t belongs to the answer to Q", where t is a tuple specified by the user. In the regular case (i.e., when data are precise), such queries are of interest because they fit users' needs when the question is about the existence of a given fact (or type of fact).

The extension to such queries when null values (in the sense of unknown) are present in the database, has been considered in [1]. In this case, the answer to a yes/no query becomes uncertain; for instance, in the case of a membership-based query, it can be either:

- *yes*, i.e., it is *certain* that tuple t belongs to the result of Q, whatever the actual value taken by attributes whose value is known as *null*;
- *no*, i.e., it is *certain* that tuple t does not belong to the result of Q, and there is actually no choice which allows the construction of t;
- *maybe*, i.e., it is possible that tuple t appears or not in the answer to Q depending on the choice made for attributes whose value is *null*.

In other words, the answer to a yes/no query calls on the notions of possible and certain answers (regardless of possibility theory). The answers yes and no require that it is certain that t is or is not

present in the answer to Q, while maybe relies only on the possibility for t to belong to this answer, i.e., in (at least) one world associated with the imprecise database.

However, null (unknown) values correspond to the complete absence of knowledge as to the somewhat acceptable values which is a limit case, useful from a technical point of view, but perhaps seldom and then unrealistic in practice. Fortunately, such nulls can be refined in terms of regular subsets (or-sets or intervals) or weighted sets (probability or possibility distributions). In the rest of the paper, we first consider possibility theory for the representation of imprecise data, before studying what changes when probability theory is used instead. Possibility theory provides an ordinal framework for the management of uncertainty able to deal with linguistic descriptions of imprecise data. In addition, the values appearing in this framework have no connection with frequency of events as it is the case with probability theory. In the framework of possibilistic databases, the general format of yes/no queries becomes:

> "to what extent is it possible and certain that the answer to Q fulfills condition C ?"

where Q is a relational algebraic query (we will see in section 2 that some constraints exist as to the operators that can be present in Q). Of course, in a probabilistic setting, the answer of a yes/no query is characterized by a single degree, which expresses the probability that the answer to Q fulfills the specified condition C.

The structure of the paper is the following. Some basic notions on possibility theory are recalled in section 2, where possibilistic relational databases are introduced as well. In that same section, we also describe the principles for processing some algebraic queries in a "compact" way (i.e., applying directly to the possibilistic database, without making the worlds explicit). In particular, the concept of a strong representation system and its characteristic property ensuring the correctness of a "compact" calculus is pointed out. Then, in section 3, the family of queries considered in the paper, namely generalized yes/no queries, is presented. Several types of such queries are identified and the algorithms that permit to obtain the two desired degrees (possibility and necessity) are outlined for each of these types of queries. Section 4 studies what would be the impact on the algorithms/performances if the probabilistic model of uncertainty was used instead of the possibilistic one. Finally, section 5 is dedicated to some conclusions on the results achieved as well as to some lines of future research.

2. A POSSIBILISTIC DATABASE MODEL

2.1 Possibility theory: some reminders

Possibility theory [9] provides an ordinal model for uncertainty where imprecision is represented by means of a preference relation encoded by a total order over the possible situations. This framework is strongly linked to fuzzy sets since the idea is to constrain the values that can be taken by a variable, by means of a normalized fuzzy set. A possibility distribution is a mapping π from a given domain into the unit interval [0, 1] and $\pi(a)$ gives the possibility degree associated with the fact that the actual value of the variable is a. The normalization condition imposes that at least one value (a_0) is completely possible, i.e., $\pi(a_0) = 1$. This formalism is particularly suited to the expression of subjective uncertainties described by means of linguistic terms such as big, young, rather small, etc. When the domain is finite, a possibility distribution is denoted by: $\{\pi_1/a_1 + \ldots + \pi_n/a_n\}$ where a_i is a candidate value and π_i is its possibility degree. This approach provides a unified framework for representing precise values, as well as imprecise ones (regular sets) or vague ones (fuzzy sets), and various null value situations (see [6] for more details). Any event E defined on the powerset of X (the set of all the elementary events) is characterized by two measures Π and N. The axioms related to the measure of possibility Π are the following:

$$\Pi(X) = 1 \text{ (which requires the normalization condition)},$$
$$\Pi(\varnothing) = 0,$$
$$\Pi(E1 \cup E2) = \max(\Pi(E1), \Pi(E2)) \quad (1)$$

and the measure of possibility of the event E is derived from the possibility distribution associated with the concerned variable in the following way:

$$\Pi(E) = \max_{x \in E} \pi(x).$$

The possibility of the conjunction of two events is given by:

$$\Pi(E1 \cap E2) \leq \min(\Pi(E1), \Pi(E2)),$$

but if E1 and E2 are non interactive events:

$$\Pi(E1 \cap E2) = \min(\Pi(E1), \Pi(E2)). \quad (2)$$

The only relationship between the possibility of E^c (the opposite event of E) and that of E is:

$$\max(\Pi(E), \Pi(E^c)) = 1,$$

which entails that if $\Pi(E) = 1$, nothing can be said for $\Pi(E^c)$, which can range from 0 to 1. In order to have a better characterization of the event E, the measure of certainty (or necessity) N has been also introduced:

$$N(E) = 1 - \Pi(E^c). \quad (3)$$

In other words, the less possible E^c, the more certain E. Due to the duality between these two measures, the following formulas hold in the general case:

$$N(E1 \cap E2) = \min(N(E1), N(E2))$$

$$N(E1 \cup E2) \geq \max(N(E1), N(E2)).$$

In addition, if E1 and E2 are two non interactive events:

$$N(E1 \cup E2) = \max(N(E1), N(E2)). \quad (4)$$

As far as regular (i.e., non-fuzzy) events are concerned, it can be proven that:

$$\Pi(E) < 1 \Rightarrow N(E) = 0. \quad (5)$$

Then, these two measures provide a total order over the set of events which can be ordered according to Π for those who are not at all certain and according to N for those which are completely possible.

2.2 Possibilistic databases and worlds

In contrast to a regular database, a possibilistic database D may have some attributes which take imprecise values. In such a case, a possibility distribution is used to represent all the more or less

acceptable candidates for the attribute value. In the rest of this paper, only finite possibility distributions are taken into account.

The first version of a possibilistic database model was introduced by Prade in the mid 80s [8]. From a semantic point of view, a possibilistic database D can be interpreted as a set of usual databases (also called worlds), denoted by rep(D), each of which being more or less possible (one of them is supposed to correspond to the actual state of the universe modeled). Any world W_i is obtained by choosing a candidate value in each possibility distribution appearing in D and its degree of possibility is the minimum of those of the candidates taken (according to formula 2 since choices are assumed to be independent).

Example 1. Let us consider the possibilistic database D involving two relations: im and pl whose respective schemas are IM(#i, ap, date, place) and PL(ap, lg, msp). Relation im describes satellite images of airplanes and each image, identified by a number (#i), taken on a certain location (place) a given day (date) is supposed to include a single airplane whose type ap is possibly ill-known, due to the imprecision in the recognition process. Relation pl gives the length (lg) and maximal speed (msp) of each airplane and is a regular (precise) relation. With the extension of im:

im	#i	ap	date	place
	i_1	a_1	$\{1/d_1 + 0.7/d_3\}$	c_1
	i_3	$\{1/a_3 + 0.3/a_4\}$	d1	c_2

four worlds can be drawn, since there are two candidates for date (resp. ap) in the first (resp. second) tuple of im. Each of these worlds involves relation pl which is precise and one of the four regular relations issued from the possibilistic relation im. ♦

2.3 The extended model

As mentioned before, a calculus based on the processing of the query Q against worlds is intractable and a compact approach to the calculus of the answer to Q must be found out. It is then necessary to be provided with both a data model and operations which have good properties: i) the data model must be closed for the considered operations, and ii) it must be possible to process any query in a compact way. In addition, its result must be a compact representation of the results obtained if the query were applied to all the worlds drawn from the possibilistic database D, i.e.: rep(Qc(D)) = Q(rep(D)) where rep(D) denotes the set of worlds associated with D and Qc stands for the query obtained by replacing the operators of Q by their compact versions. This property characterizes data models called strong representation systems. A data model complying with this property is briefly described hereafter (see [2] for more details).

Because some operations (e.g. selection) filter candidate values, there is a need at the compact level for expressing that some tuples can have no representative in some worlds. A simple solution is to introduce a new attribute, denoted by N (valued in [0, 1]). The value of N associated with a tuple t expresses the certainty of the presence of a representative of t in any world. A tuple is denoted by a pair N/t where N equals 1 for tuples of initial possibilistic relations as well as when no candidate value has been discarded.

Example 2. Let us consider the following extension of the possibilistic relation im:

im	#i	ap	date	place
	i_1	B-727	d_1	c_1
	i_2	ATR-72	d_1	c_2
	i_3	{1/B-727+ 0.7/ATR-42}	d_2	c_4
	i_4	{1/B-727 + 1/B-747}	d_2	c_2

The selection based on the condition "ap = B-727" leads to discard the candidates which are different from this desired value. Thanks to the introduction of attribute N, the result of the selection is:

res	#i	ap	date	place	N
	i_1	B-727	d_1	c_1	1
	i_3	B-727	d_2	c_4	0.3
	i_4	B-727	d_2	c_2	0

In the second tuple N is equal to 0.3, i.e, 1 minus the possibility degree attached to the most possible alternative that has been discarded.

This resulting relation has four interpretations (worlds), including that made of the single tuple $<i_1$, B-727, d_1, $c_1>$ whose degree of possibility is: min(1, 1 – 0.3, 1 – 0) = 0.7. ♦

Another aspect of the model concerns the fact that one sometimes has to express dependencies between candidate values coming from different attributes in a same tuple. This makes it necessary to represent attribute values defined as possibility distributions over several domains, which is feasible in the relational framework thanks to the concept of a nested relation. In such relations, exclusive candidates are represented as weighted tuples. Therefore, level-one relations keep their conjunctive meaning, whereas nested relations have a disjunctive interpretation.

Example 3. Let us consider the following intermediate relation int-r involving the nested attribute X(date, place):

int-r	#i	ap	X		N
			date	place	
	i_1	B-727	$\{1/<d_1, c_1> + 0.7/<d_1, c_2> + 0.4/<d_3, c_2>\}$		1
	i_3	B-727	$<d_1, c_2>$		0.3
	i_4	B-727	$\{0.3/<d_3, c_2>\}$		0

This relation is associated with 12 worlds since the first tuple admits 3 interpretations, the second and third ones have 2 interpretations among which Ø (no representative). ♦

2.4 The operators

In order to meet the objective of a compact processing of algebraic queries, the operators must be adapted so as to accept possibilistic relations (as defined in the previous section) both as inputs and outputs. It turns out that the only operations such that

an input tuple participates in the production of at most one element of the result, can be expected to admit a compact version (see [2, 4]). As a consequence, the intersection, the difference and the Cartesian product (then the join in the general case) are discarded and the four acceptable operators are now dealt with. Considering the specific objective of this paper, we will not give a detailed presentation of these operators (whose definitions can be found in [2, 4]). We limit ourselves to a brief introduction and the behavior of the operators is then illustrated by an example.

The selection removes the unsatisfactory candidate values (and possibly tuples) and updates the value N of each retained tuple (cf. example 2).

The originality of the projection in this context is twofold: i) it does not remove duplicates and ii) it updates the possibility degrees attached to the remaining values according to the possibility distributions that were present in the suppressed attributes.

Although there is no hope for defining a compact version of the join (due to the constraint expressed above), it turns out that a specific join, namely the fk-join, is acceptable. The fk-join allows for the composition of a possibilistic relation r of schema R(W, Z), where W and Z may take imprecise values, and a regular relation s of schema S(W, Y) where the functional dependency W $\rightarrow$ Y holds. So, W is the key of s and then a foreign key of r. The fk-join consists in completing tuples of r by adding the image of the W-component. By definition, this leads to a resulting relation involving the nested relation X(W, Y), which "connects" the pairs of candidates over W and Y. The degree of possibility of any structured candidate value is that attached to the value that has been completed. Similarly to the selection, N is updated to keep track of the most possible candidate value for which no match occurred.

Lastly, the union of two relations whose schemas are compatible keeps all the tuples issued from the two input relations without any duplicate removal.

Example 4. This example is intended for illustrating the overall functioning of the procedure designed for answering algebraic queries addressed to possibilistic databases. Let us consider the possibilistic database composed of the relations im1(IM), im2(IM) and pl(PL) where IM and PL are the schemas introduced in example 1. The two relations im1 and im2 are assumed to contain images of airplanes taken by two distinct sources (e.g., satellites). Let us consider the query looking for the existence of images of airplanes whose maximal speed is over 900 km/h and taken by either of the two satellites at a date different from d_3 and d_4, which corresponds to the algebraic query Q:

fk-join(union(select(im1, date $\notin$ {d_3, d_4}), select(im2, date $\notin$ {d_3, d_4})), select(pl, msp > 900), {ap}, {ap}).

With the following extensions:

pl

ap	lg	msp
a_1	20	1000
a_2	25	800
a_3	18	600
a_4	20	1200
a_5	20	1000

im1

#i	ap	date	place	N
i_1	a_3	{$1/d_1 + 0.7/d_3$}	c_1	1
i_2	{$1/a_2 + 0.7/a_1$}	d_1	c_2	1

im2

#i	ap	date	place	N
i_3	{$1/a_4 + 1/a_5$}	{$0.6/d_4 + 1/d_1$}	c_3	1

we obtain the resulting relation res hereafter:

res

#i	X			date	place	N
	ap	lg	msp			
i_2	$0.7/<a_1, 20, 1000>$			d_1	c_2	0
i_3	{$1/<a_4, 20, 1200> + 1/<a_5, 20, 1000>$}			d_1	c_3	0.4

3. GENERALIZED YES/NO QUERIES

3.1 Introduction

In this paper, we consider the following types of generalized yes/no queries (but this typology is clearly not exhaustive):

- vacuity-based yes/no queries: to what extent is it possible and certain that the answer to Q is non-empty?
- membership-based yes/no queries: to what extent is it possible and certain that tuple t belongs to the answer to Q?
- cardinality-based yes/no queries: to what extent is it possible and certain that the answer to Q contains at least (resp. at most, exactly) k items?
- inclusion-based yes/no queries: to what extent is it possible and certain that the answer to Q contains the set of tuples {t_1, ..., t_k}?

For each of these queries, the processing obeys the following three step scheme:

- pre-processing in order to eliminate the unnecessary attributes (and, for membership-based queries, to remove from the operand relations the tuples that cannot generate the target tuple);
- evaluation of Q, which yields a resulting possibilistic table res of schema (A_1, ..., A_n);
- post-processing aiming at computing the final possibility and certainty degrees Π and N.

In the following, we only focus on the post-processing part (last step). Concerning the evaluation of Q, the reader can refer to [4]. We will see in the next subsection that the four previous types of queries can be clustered into two categories: those which require only a sequential scan of the result of Q (vacuity and membership-based queries, denoted by type 1); and those for which it is necessary to use a “trial and error” type of algorithm (cardinality and inclusion-based queries, denoted by type 2). In any case, let us notice that formula 5 allows to reduce the computation effort in certain situations. If the possibility degree is computed first and if

it is strictly less than 1, then there is no need for computing the necessity degree since one knows that it equals zero. Reciprocally, if the necessity degree is computed first and is strictly positive, then one knows that Π equals 1.

3.2 Principles of the algorithms

3.2.1 Vacuity-based yes/no queries

The degrees of possibility and necessity, which constitute the answer to the yes/no query, can be deduced from relation res the following way:

- if res is empty, then the possibility degree equals 0. Due to formula 5, N also equals 0. In other words, it is *certain* that the answer to Q is empty;
- if res contains at least one tuple whose N-value is strictly positive, the degree assessing the necessity that the answer to Q is non-empty is given by: $\max_{t \in res} N(t)$. Indeed, the event under consideration is E = "the first tuple of res exists **or** … **or** the last tuple of res exists". According to possibility theory, the necessity of E is the maximum of the necessity of each of the non-interactive elementary events "the j^{th} tuple of res exists" (cf. formula 4). The contraposition of formula 5 ($N > 0 \Rightarrow \Pi = 1$) allows to state that the possibility degree of E is 1. In other words, it is *completely possible* that the answer to Q is non-empty (and it may be certain in the special case where N itself is 1);
- if, for every tuple t_i of relation res, the N-value equals 0, the certainty of the event "the answer to Q is non-empty" is 0 and its possibility is given by: $\max_{t \in res} \pi(t)$ according to formula 1.

The algorithm for the calculus of the degrees is given hereafter:

Input: relation res whose generic tuple is:
u = nc/<{A_1[1], ..., A_1[k_1]}, ... {A_n[1], ..., A_n[k_n]}> where A_i[j] is a pair <possibility degree (pi), candidate value (val)>;

Output: degrees of possibility poss and necessity nec that the answer to Q is non-empty;

Body of the algorithm:

```
lp ← false; poss ← 1; nec ← 0;
while not end(res) do
  read next tuple u of res;
  lp ← true;
  poss-u ← 1;
  for j from 1 to n do
     pi ← 0;
     for q from 1 to u.k_j do
        pi ← max(pi, u.A_j[q].pi)
     enddo;
     poss-u ← min(poss-u, pi)
  enddo;
  if u.nc > 0 then
     nec ← max(nec, u.nc);
     poss ← 1
  else poss ← max(poss, poss-u) endif;
enddo;
if not lp then poss ←0; nec ← 0 endif.
```

Clearly, this algorithm has a complexity which is linear with respect to the number of tuples of the input relation res. Then, the performances of the overall treatment of a generalized yes/no query is not significantly increased by this additional step.

3.2.2 Membership-based yes/no queries

One may observe that, in the context of a regular database, the initial membership-based yes/no query:

"is it true that tuple t belongs to the answer to Q?"

can be transformed into the vacuity-based yes/no query:

"is the result of query Q' non empty?"

where Q' is derived from Q by adding selection conditions corresponding to the values of t (= <a_1, …, a_n>) on the underlying attributes A_1, …, A_n, i.e., $A_1 = a_1$ **and** … **and** $A_n = a_n$., followed by a projection onto the attributes A_1, ..., A_n. The interest for such a transformation is twofold :

- selections are performed and then the performances can only be improved since the size of intermediate relations decrease,
- the final decision (yes or no) is easier to make, since the result is either empty and the answer is no, or not (it contains one or several occurrences of t, but this does not matter) and the answer is yes.

It has been proven in [5] that such a transformation is also valid in an imprecise database framework inasmuch as the model used is a strong representation system (which is the case here). Let us notice that nested relations may be induced by the various intermediate operations preceding the final projection. However, it is certain that any tuple resulting from Q', whatever the schema of the input relation of the final projection, is only made of the values a_1 to a_n. This observation implies that it is possible to produce a final relation whose schema is (A_1, … , A_n) without any nested relation. The algorithm aiming at computing the final possibility and necessity degrees is a straightforward adaptation of that presented in the previous section, cf. [5] for more detail.

3.2.3 Cardinality-based yes/no queries

The post-processing of the compact result of Q entails determining the possibility attached to worlds involving a certain number of elements in order to compute the degrees Π/N of the event "to what extent is it possible and certain that the answer to Q contains at least (at most, exactly, …) k distinct elements?". In fact, if we are able to compute the possibility in each case ("at least", "at most", "exactly", …), we can also compute the necessity. Indeed, the necessity that the answer to Q contains at least (respectively at most) k distinct elements is 1 – the possibility that the answer to Q contains less than (respectively more than) k distinct elements. Moreover, the necessity that the answer to Q contains exactly k distinct elements is 1 minus the possibility that the answer to Q contains less than or more than k distinct elements.

The problem is that some tuples of the resulting relation res can produce representatives which are indeed duplicates. According to the case considered ("at least", "at most" or "exactly"), duplicates do not have the same impact. It turns out that, in all cases, the procedure attached to the post-processing must rely on a "trial and error" technique. Let us illustrate this through two

examples, one for the case "at least", then another for the case “at most”).

Example 5. Let us consider the following relation res which is assumed to be the result of the compact processing of an algebraic query Q:

res	A	B	N
	$\{1/a_1 + 0.6/a_2\}$	b	0.3
	a_1	b	1

The degree of possibility that the answer to Q contains at least 2 different tuples cannot be obtained by taking the most possible representative of the two tuples of res because they are identical ($<a_1, b>$). Therefore, the representatives which are duplicates must be identified in order to compute the exact cardinality. ♦

Example 6. Let us consider the following relation res which is assumed to be the result of the compact processing of an algebraic query Q:

res	A	B	N
	a_2	$\{1 / b_3 + 0.9 / b_2\}$	1
	a_2	b_3	1
	$\{1 / a_2 + 0.3 / a_1\}$	b_3	0.3
	$\{0.9 / a_4 + 0.8 / a_5\}$	$\{0.5 / b_1\}$	0

and the cardinality-based query: “to what extent is it possible that the answer to Q has at most 1 element?”.

One may think that the possibility degree delivered to the user would be 0 because each world contains at least a representative of the 2 tuples with N = 1. But in fact, there may be some duplicates among these 2 tuples (it is indeed the case here), that is why the procedure attached to the post processing must rely on a “trial and error” technique. The question is whether the procedure should only concern the tuples with N = 1 or all the tuples somewhat certain. The world made only of the most possible representatives of the first two tuples has the possibility degree $\min(1, 1, 1 - 0.3, 1 - 0) = 0.7$ while the one containing also the best representative of the third tuple (which is (a_2, b_3)) has the possibility degree $\min(1, 1, 1, 1 - 0) = 1$. This latter world contains one tuple because the three representatives are duplicates and it is more possible than the first one. The "trial and error" procedure should therefore consider all the tuples somewhat certain. ♦

The algorithm proposed hereafter aims at computing the possibility degree π of the most satisfactory world with respect to the desired cardinality. Such an algorithm calculates a series of vectors $V = (x_1, x_2, \ldots, x_n)$ (n being the number of tuples in relation res) where each component x_i takes its values in a finite set E_i. Ultimately, it delivers the best (in terms of possibility degree) vector V.

A solution is a vector V which represents a world (a candidate regular relation for the possibilistic relation res resulting from the compact evaluation of Q). Its dimension is the number n of tuples in relation res. The components of vector V are precise tuples (the i^{th} position of V is the representative tuple produced by the i^{th} tuple of relation res). Some positions of V may be empty, which occurs when the value N in relation res is different from 1 (the corresponding tuple may have no representative in a given world). The algorithm is the following:

```
procedure OptimalSolution (i integer)
  begin
    compute E_i;
    for x_j in E_i do
      if satisfactory(x_j) then
        memorize(x_j);
        if solutionFound then
          if better then keepSol endif
        else if stillPossible then
          optimalSolution(i + 1) endif
        endif;
        undo(x_j);
      endif;
    endfor;
end;
```

The different components of this algorithm when the comparator is “at least” are:

E_i: list of the precise tuples corresponding to the possible representatives of the i^{th} tuple from res (including Ø if N < 1);

Π_i: list of the respective possibility degrees π_j of each x_j in E_i (it is equal to 1 – N if x_j is Ø);

V: represents a world of res (a regular relation that is a possible answer to Q);

Pos: vector of the same dimension as V, it contains the possibility degrees of the tuples of V (the possibility degree associated to V being the minimum over Pos);

Card: the cardinality of vector V (i.e., the number of tuples in V different from Ø);

BestΠ: the possibility degree of the most possible world found so far;

satisfactory(x_j): $\pi_j >$ BestΠ (it is possible for the current solution to be better than the best one already found only if the possibility of the current candidate tuple is over BestΠ, since the overall possibility degree of a world is computed by means of a minimum, cf. keepSol below);

memorize(x_j):

$V[i] \leftarrow x_j$;
$Pos[i] \leftarrow \pi_j$;
$b \leftarrow (x_j \neq \text{Ø}$ and the same tuple was not already in V);
if b then Card ← Card + 1;

solutionFound: (i = n) and Card ≥ k;

better: $\min_{q \text{ in } [1, n]} Pos[q] >$ BestΠ;

keepSol: Best$\Pi \leftarrow \min_{q \text{ in } [1, n]} Pos[q]$;

stillPossible: Card + (n – i) ≥ k (there is no hope to construct a solution containing at least k tuples from the current vector if its cardinality is not at least equal to k minus the number of tuples that can still be chosen);

undo(x_j): Pos[i] ← 0; if b then Card ← Card – 1;

This algorithm can be easily adapted to the case where another comparator than "at least" is used, by modifying the components solutionFound and stillPossible. The necessity degree which is the answer to the query "to what extent is it certain that the answer to Q has at least (resp. at most, exactly, …) k distinct elements?" is 1 minus the possibility degree obtained by the algorithm above for the query "to what extent is it possible that the answer to Q has less than (resp. more than, less than or more than, …) k distinct elements?".

Remarks. In order to reduce the number of the worlds to be computed, some improvements can be brought to the algorithm above:

1) When BestΠ is equal to 1 the processing can be stopped.
2) The sets E_i can be ranked in decreasing order on the possibility degrees. In that case, once an unsatisfactory x_j is found, the loop can be stopped (because the following x-values would be unsatisfactory too).
3) In the cases "at least k" and "exactly k", one can take advantage of the number n of tuples in relation res. For instance, if the user wants 5 responses (k = 5) while the relation res contains only 3 tuples, the result is obviously 0.
4) For the case "at most k", the algorithm can be evaluated only on the tuples somewhat certain. Let t be an imprecise tuple whose necessity degree is 0, if we add a representative of t to the current solution, the possibility degree associated to the solution can only decrease (or stay the same). Since the criterion is of the form "at most k", it is thus a better idea not to take a representative of t, this choice being possible at degree 1. Furthermore, if relation res contains at most k tuples with N > 0, the procedure based on a "trial and error" technique is not needed any more and the result is $\Pi = 1$ (because the possibility distributions are normalized for tuples somewhat certain).

3.2.4 Inclusion-based yes/no queries

Inclusion-based queries are similar to cardinality-based queries where the condition on the cardinality is "equals k", except that one has to deal with an additional constraint about the values of the k tuples (since we have a specified set of target tuples). As to the possibility degree, the computation can be made thanks to an adaptation of the preceding algorithm. The elements that change are: memorize(x_j), solutionFound, and undo(x_j):

memorize(x_j):

```
V[i] ← x_j; Pos[i] ← π_j;
if x_j = t_1 then
  B[1] ← true;  -- B is a vector of Booleans of size k
  b ← x_j was not already present in V;
  if b then Nbtuple ← Nbtuple + 1;
endif;
. . .
if x_j = t_k then
  B[k] ← true;
  b ← x_j was not already present in V;
  if b then Nbtuple ← Nbtuple + 1;
endif;
```

where Nbtuple denotes the number of target tuples already found.

solutionFound:

((i = n) and (B[1] and … and B[k]));
the tuples of relation res have been all examined, and all the target tuples belong to this solution (n denotes the cardinality of relation res).

undo(x_j):

```
if it exists q such that x_j = t_q then
  b ← x_j is not present several times in V;
  if b then Nbtuple ← Nbtuple – 1; B[q] ← false
  endif;
endif;
```

As to the necessity degree $N(\{t_1, \ldots, t_k\} \subseteq res(Q))$, according to the axioms of possibility theory, it is equal to:

$$N(t_1 \in res(Q) \text{ and } \ldots \text{ and } t_k \in res(Q))$$
$$= 1 - \Pi(t_1 \notin res(Q) \text{ or } \ldots \text{ or } t_k \notin res(Q))$$
$$= 1 - \max(\Pi(t_1 \notin res(Q)), \ldots, \Pi(t_k \notin res(Q)))$$
$$= 1 - \max(1 - N(t_1 \in res(Q)), \ldots, 1 - N(t_k \in res(Q))).$$

Thus, the computation of the necessity degree of the inclusion-based query amounts to running k times the algorithm for membership-based queries (cf. section 3.2.2), with a different target tuple each time. Let us notice that, considering that this computation has a linear complexity in terms of the cardinality of res, an interesting solution is to compute N first, because if N is strictly positive, then one can conclude that Π equals 1 without running the trial and error procedure.

3.2.5 About the complexity of type 2 yes/no queries

The algorithms computing Π for cardinality-based and inclusion-based queries include two pruning conditions: one based on the optimality of the solution (satisfactory(x_j)), the other on the cardinality of the current world under construction (stillPossible). Clearly, the first condition will be very effective (i.e. will cut many branches) if a highly possible satisfactory world is encountered early. If it is not the case (the extreme situation being that the final possibility degree equals zero), only the second pruning condition can have an effect and reduce the number of branches to explore (thus avoiding to construct all the possible worlds of relation res). Let us notice, however, that the number of worlds attached to res is much smaller than the number of worlds attached to the initial possibilistic database, due to the reduction obtained by means of the compact evaluation of query Q. Anyway, in certain cases (when the cardinality of res is too large), due to the combinatorial nature of the algorithms, even when both pruning conditions can be used together, the complexity will still be too high to reach acceptable performances. Therefore, it is important to envisage some ways to overcome this problem. Concerning the computation of Π, an idea could consist in calculating only an approximate answer, i.e., an underestimation of the actual value of Π. This could be done by means of a greedy algorithm scanning relation res sequentially. In the case of inclusion-based queries, for instance, this algorithm could choose, for each imprecise tuple of res, the most possible candidate which is equal to a target tuple not found yet (the scan could be made in both directions, bottom-up and top-down, in order to improve the quality of the estimation). As a matter of fact, even when relation res has a "reasonable" size, such a greedy algorithm could also be used as a preprocessing step in order to obtain – in a non-expensive way – a Π-value that could be used to initialize the

variable BestΠ before running the trial and error procedure, so as to make the first pruning condition (i.e., that based on the optimality of the solution) more effective.

4. PROBABILISTIC DATABASES AND GENERALIZED YES/NO QUERIES

4.1 Processing the Algebraic Query

Concerning the database model, it has been shown in [3] that the model described in section 2.3 constitutes also a strong representation system in a probabilistic framework, for the same set of algebraic operators (selection, projection, fk-join and union). The only difference with the possibilistic case is that we do not need an extra attribute N in order to have available the probability for an imprecise tuple to have a representative in any world. Indeed, in the probabilistic case, we know that some candidates have been discarded from a distribution when the sum of the degrees in this distribution is less than 1. In the case where a single distribution is considered, the probability attached to the situation where the tuple has no representative is equal to 1 minus the sum of the degrees attached to the remaining elements. Since the knowledge necessary to this calculus is available in the tuple, there is no need to explicitly store the degree itself in the relation.

As to the processing of the algebraic query underlying a generalized yes/no query, it differs only slightly from the possibilistic case. Basically, what changes is that in the definitions of the operators, the minimum is replaced by the product, and the maximum by the sum.

4.2 About the Post-Processing

Concerning the first group of generalized yes/no queries (vacuity and membership-based), the post-processing step is a simple adaptation of that described in section 3.2. In the vacuity-based case, for instance, the probability prob that the resulting relation res is non-empty equals 1 minus the probability that it is empty, i.e., that every of its tuples has no representative:

```
empty ← 1;
while not end(res) do
  read next tuple u of res;
  nonrep-u ← 1;
  for j from 1 to n do
     rep ← 0;
     for q from 1 to u.k_j do
        rep ← rep + u.A_i[q].pr
     enddo;
     nonrep-u ← nonrep-u * (1 - rep)
  enddo;
  empty ← empty * nonrep-u;
enddo;
prob ← 1 - empty;
```

Concerning the second group of queries (cardinality and inclusion-based), the main question concerns the pruning conditions: do they still hold or not? The condition that concerns optimality still holds in the probabilistic case: the probabilities are aggregated by means of a product instead of a minimum and it is still possible to discard a current solution when its current probability degree is already below the degree of the optimal solution found so far. The second pruning condition concerns the cardinality of the current solution. Again, this condition still holds since it is completely independent of the degrees (and therefore, of their semantics). Thus, the only change concerns the way the degrees themselves are computed.

5. CONCLUSION

In this paper, we have considered a specific type of queries, namely generalized yes/no queries, addressed to imprecise databases. First, we have recalled the main characteristics of an imprecise database model enabling to process algebraic queries in a "compact" way, under a constraint concerning the algebraic operators that are authorized (selection, projection, fk-join and union). Concerning yes/no queries, the main results are that: i) depending on the type of condition involved, the processing may be based either on a sequential scan of the resulting relation or on a trial and error algorithm. In the latter case, we have pointed out some pruning conditions that can limit the combinatorial growth, but obviously, it may not be sufficient to reach acceptable performances in certain cases. Then a solution could be to use a greedy algorithm in order to obtain an approximation of the degrees; ii) that the model of uncertainty has a very limited impact on the processing of generalized yes/no queries. As to future works, it is obviously desirable to perform some experimentations in order to assess in a more precise way the efficiency of the approach proposed here.

6. REFERENCES

[1] Abiteboul, S., Kanellakis, P., and Grahne, G. On the representation and querying of sets of possible worlds, *Theoretical Computer Science*, vol. 78, 1991, 159-187.

[2] Bosc, P., and Pivert, O. Towards an Algebraic Query Language for Possibilistic Databases. *Proc. of the 12th Conference on Fuzzy Systems (FUZZ-IEEE'03)*, 2003, 671–676.

[3] Bosc, P., and Pivert, O. On a strong representation system for imprecise relational databases, *10th International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems (IPMU'04)*, Perugia, Italy, July 4-9, pp. 1759-1766, 2004.

[4] Bosc, P., and Pivert, O. About Projection-selection-join Queries Addressed to Possibilistic Relational Databases. *IEEE Transactions on Fuzzy Systems*, 31, 2005, 124–139.

[5] Bosc, P., and Pivert, O. About Yes/no Queries against Possibilistic Databases. *International Journal of Intelligent Systems*, to appear.

[6] Bosc, P., and Prade, H. An introduction to the fuzzy set and possibility theory-based treatment of flexible queries and uncertain or imprecise databases, in: *Uncertainty Management in Information Systems*, A. Motro and P. Smets (Eds), Kluwer Academic Publishers, pp. 285-324, 1997.

[7] Imielinski, T., and Lipski, W. Incomplete information in relational databases. *Journal of the ACM*, vol. 31, 1984, 761-791.

[8] Prade, H. Lipski's approach to incomplete information data bases restated and generalized in the setting of Zadeh's possibility theory. *Information Systems*, vol. 9, 1984, 27-42.

[9] Zadeh, L.A. Fuzzy sets as a basis for a theory of possibility. *Fuzzy Sets and Systems*, vol. 1, 1978, 3-28.

Structured Queries for Semistructured Probabilistic Data

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

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

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

ABSTRACT

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

Categories and Subject Descriptors

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

General Terms

Languages, Algorithms

Keywords

probabilistic databases, query langauges

1. INTRODUCTION

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

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

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

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

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

In its current stage, SPDBMS is used by a number of client applications [3, 21] developed for a research project devoted to planning in the presence of uncertain information[13].

This paper describes the newest component of SPDBMS, Semistructured Probabilistic Object Query Language (SPOQL) – an SQL-like declarative language for queries over SPO data. SPOQL is designed to replace SP-Algebra as the interface of choice for communication with the SPDBMS server. While SP-Algebra allows us to study various properties of SPOs in a rigorous manner, it is inconvenient to use as the end-user (or client) query language. SPOQL provides convenient declarative syntax for queries and represents complex queries over SPO data in a straightforward manner.

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

2. SEMISTRUCTURED PROBABILISTIC OBJECTS

2.1 Motivating Example

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

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

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

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

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

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

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

2.2 SPO model and SP-Algebra

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

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

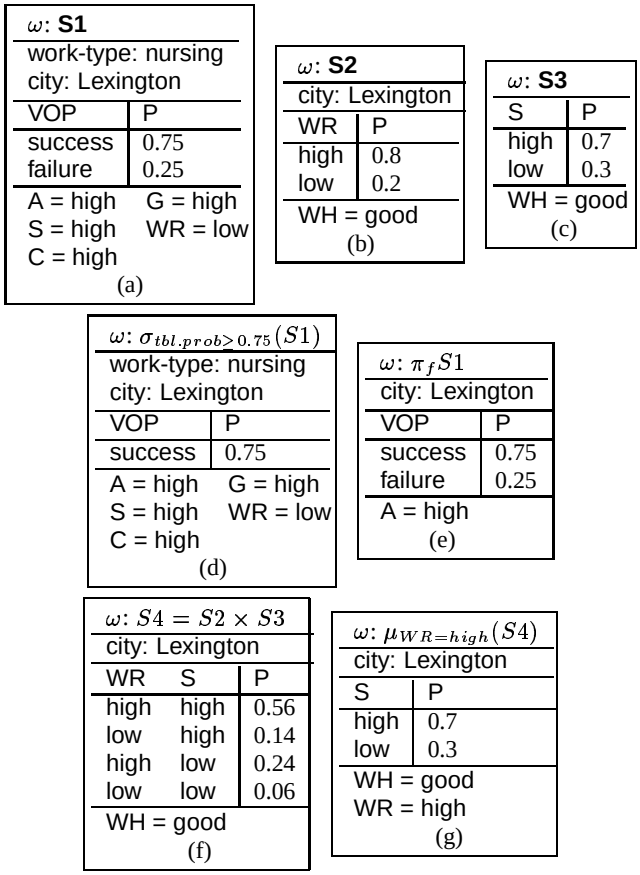


Figure 1: Probability distributions for the random variables in the activity "Volunteer Placement (VOP)", where projection condition f is $cnt.city \wedge cnd.A$.

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

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

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

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

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

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

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

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

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

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

Table 1: Atomic selection conditions

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

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

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

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

Table 2: SP-Algebra expressions

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

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

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

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

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

When two SP-relations S and S' are provided as input, they will have join-compatible SPOs , cp-compatible SPOs and neither join-nor cp-compatible SPOs. The mix operation is the union of the join and the cartesian product S and S'.
Let $S = \langle T, V, P, C, \omega\rangle$ and $S' = \langle T', V', P', C', \omega'\rangle$ are two cp-compatible or join-compatible SPOs. Then, the result of their mix operation, denoted $S \otimes S'$, is defined as follows [2]:
$S \otimes S' = (S \times S') \cup (S \ltimes S')$

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

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

2.3 SP-Algebra equivalences

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

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

Table 3: Query Equivalences for SP-Algebra operations.

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

Table 4: Query Equivalences for SP-Algebra operations.

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

2.4 Implementation

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

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

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

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

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

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

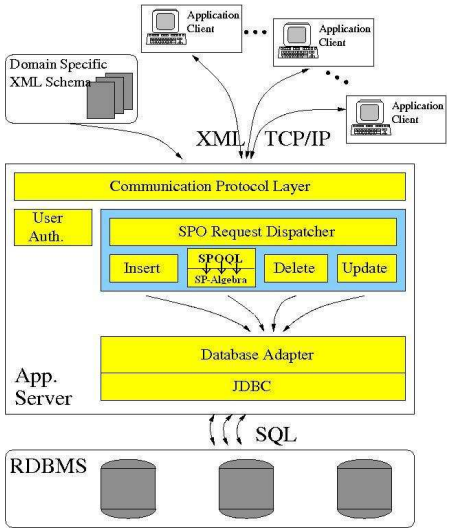


Figure 3: The overall architecture of SPDBMS.

3. SPOQL

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

3.1 Syntax of SPOQL

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

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

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

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

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

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

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

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

3.2 SPOQL Semantics By Example

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

1.
```
SELECT * FROM S1
WHERE S1.tbl.VOP = 'success'
AND S1.tbl.prob > 0.7
```

2.
```
SELECT S1.cnt.city, S1.cnd.A FROM S1
```

3.
```
SELECT * FROM S2 JOIN S3
```

4.
```
SELECT * FROM S4
CONDITIONAL S4.var.WR='high'
```

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

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

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

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

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

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

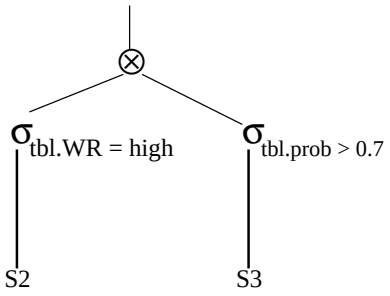


Figure 4: The query tree for SPOQL query Q.

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

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

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

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

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

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

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

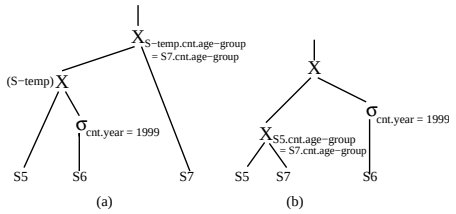


Figure 5: The query trees for SPOQL query Q' based on only left-to-right evaluation (a) and left-to-right evaluation with join condition(s) priority (b).

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

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

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

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

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

Evaluate(SPOQL query Q)

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

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

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

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

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

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

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

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

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

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

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

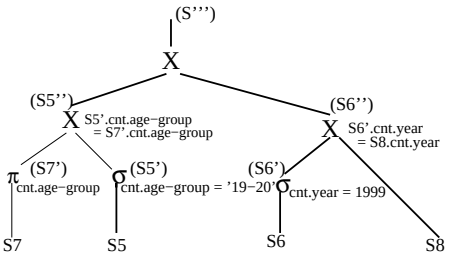


Figure 7: The query tree for SPOQL query Q''.

3.3 SPOQL Implementation

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

4. RELATED WORK

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

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

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

5. CONCLUSIONS

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

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

6. REFERENCES

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

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

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

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

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

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

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

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

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

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

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

[12] V.S. Lakshmanan, N. Leone, R. Ross, and V.S. Subrahmanian. Probview: A flexible probabilistic database system. *ACM Transactions on Database Systems*, 22(3):419–469, 1997.

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

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

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

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

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

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

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

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

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

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

Z style notation for Probabilities

Maarten M. Fokkinga
DB group, dept INF, fac EWI, University of Twente
PO Box 217, NL 7500 AE Enschede, Netherlands
m.m.fokkinga@utwente.nl

Abstract. A notation for probabilities is proposed that differs from the traditional, conventional notation by making explicit the domains and bound variables involved. The notation borrows from the Z notation, and lends itself well to calculational manipulations, with a smooth transition back and forth to set and predicate notation.

1. INTRODUCTION

The notation commonly used in applied probability theory suffers from two drawbacks: the domain of discourse is left implicit, and consequently in predicates the argument is left implicit. To say it in a crude way, the formulas have no meaning without a little verbal story along with them. As a consequence, it is hard to do machine assisted formal calculations (as striven for in, for example, transformational programming [1, 11, 2, 3, 4, 8, 7, 12, 10, 11]); it is simply too hard to feed the machine with the little verbal stories that define the semantics of various sub-expressions. This note presents a possible improvement.

The proposal is not meant to replace existing notation; the current notation has proved its functioning over the years. Rather, the new notation may be beneficial in an educational setting, and every now and then it may help to express one's ideas in a clear and precise way as a stepping-stone to achieve a convenient conventional formulation.

We shall define the notation in the coming two sections, and then illustrate the notation in a series of examples. The examples have the spirit of "theory" (formal calculations to derive some well-known and easy theorems of probability theory) and "application" (showing expressability in the field of information retrieval, coin tossing, and event spaces). We conclude with an appendix about the history of the probability notation.

2. THE NOTATION

The proposal is fully in the style of the Z notation [15], a notation designed for large scale formal specifications, supported by a range of tools [5] (syntax checker, type checker, pretty printer, proof checker, theorem prover). Below we give a list of some notations available in Z — with in the last line the proposed notation for probabilities. The list alone already clearly shows the systematic approach in the choice of the notation; in the subsequent paragraphs we shall show the advantage of this systematic notation in formal manipulations. In the list and the sequel, we shall use letter D for arbitrary *declarations*, letters P, Q for arbitrary *predicates*, and letter E for arbitrary *expressions*. We do not elaborate the syntax of these categories, but instead leave them to the imagination of the reader. The symbols | and • are *no* operator symbols, and have *no* meaning by themselves; they merely separate the three parts (namely D, P, E, and D, P, Q respectively).

Here is the list, followed by a discussion of each line:

notation	semantics
$\{D \mid P \bullet E\}$	the set, for D satisfying P, of values E
$(\lambda\, D \mid P \bullet E)$	the fct that maps D satifying P to E
$(\mu\, D \mid P \bullet E)$	the unique E where D satisfies P
$(\forall\, D \mid P \bullet Q)$	for all D satisfying P, Q holds
$(\exists\, D \mid P \bullet Q)$	there exists D satisfying P such that Q
$(\mathcal{P} D \mid P \bullet Q)$	the probability for D satisfying P that Q

concrete example		value
$\{x, y : \mathbb{N} \mid 2y = x < 5 \bullet x{+}10\}$	$=$	$\{10, 12, 14\}$
$(\lambda\, x, y : \mathbb{N} \mid 2y = x < 5 \bullet x{+}10)$	$=$	$\{(0,0)\mapsto 10,\ (2,1)\mapsto 12,\ (4,2)\mapsto 14\}$
$(\mu\, x, y : \mathbb{N} \mid 0 < 2y = x < 4 \bullet x{+}10)$	$=$	12
$(\forall\, x, y : \mathbb{N} \mid 2y = x < 5 \bullet x{+}10 < 13)$	$=$	*false*
$(\exists\, x, y : \mathbb{N} \mid 2y = x < 5 \bullet x{+}10 < 13)$	$=$	*true*
$(\mathcal{P} x, y : \mathbb{N} \mid 2y = x < 5 \bullet x{+}10 < 13)$	$=$	0.66666...

In the sequel we shall not use (and therefore not explain) the λ-form and the μ-form; they are given here only to demonstrate the variety of forms involving a '$D \mid P \bullet ...$'-part.

The form $\{D \mid P \bullet E\}$ denotes a set; the values of expression E constitute the members of the set, where the variables in E range over their domains as specified in declaration D — but only as far as predicate P is true. A more traditional notation for the example set is "$\{x{+}10 \mid x, y{\in}\mathbb{N} \wedge 2y{=}x{<}5\}$", in which, alas, the *declaration* '$x, y{\in}\mathbb{N}$' is syntactically indistinguishable from the constraining *predicate* '$2y{=}x{<}5$'.

The form $(\forall\, D \mid P \bullet Q)$ is the familiar universal quantification; it denotes the claim that for all conglomerates of variables described by D and satisfying constraint P, predicate Q holds true. The form $(\exists\, D \mid P \bullet Q)$ is its dual: the existential quantification.

The form $(\mathcal{P} D \mid P \bullet Q)$ is the proposed notation for probabilities; it denotes the probability that an arbitrary conglomerate of variables drawn from D satisfies predicate Q, given that the variables already satisfy P. We shall elaborate upon this notation later.

In all forms, when P is *true*, it may be omitted — together with the preceding symbol |. Also, in the set notation, when E is exactly the conglomerate of variables declared by D, it may be omitted together with the preceding symbol $\bullet$. Thus $\{x, y : \mathbb{N} \mid 2y{=}x{<}5\}$ stands for $\{x, y : \mathbb{N} \mid 2y{=}x{<}5 \bullet (x, y)\}$, which equals $\{(1,0), (2,1), (4,2)\}$. This abbreviation could also be done for the λ-form and μ-form, but it is not customary to do so. Similarly, it is not customary to omit '$\bullet\, Q$' when Q is *true*, but there is no formal objection to it. It is customary to omit outer parentheses when no confusion can result.

3. FORMAL MANIPULATIONS

Several laws hold for forms involving '$D \mid P \bullet \ldots$'. These laws facilitate formal manipulations. Not only is it easy for humans to apply those rules, but also a machine can easily support them since all ingredients are available in the notation — there is no need for an informal verbal explanations along with the formulas. By way of illustration we give only a few of these laws. Each line is discussed below:

$$
\begin{array}{lcl}
(\exists\, D \mid P \bullet Q) & = & (\exists\, D \bullet P \wedge Q) \\
(\forall\, D \mid P \bullet Q) & = & (\forall\, D \bullet P \Rightarrow Q) \\
\neg\, (\exists\, D \bullet P) & = & (\forall\, D \bullet \neg\, P) \\
\neg\, (\forall\, D \bullet P) & = & (\exists\, D \bullet \neg\, P) \\
\neg\, (\exists\, D \mid P \bullet Q) & = & (\forall\, D \mid P \bullet \neg\, Q) \\
\neg\, (\forall\, D \mid P \bullet Q) & = & (\exists\, D \mid P \bullet \neg\, Q) \\
\{D \mid (\exists\, D' \mid P' \bullet Q') \wedge P \bullet E\} & = & \\
\qquad \{D;\, D' \mid P' \wedge Q' \wedge P \bullet E\} & & \\
D \mid (\exists\, D' \mid P' \bullet Q') \wedge \ldots & = & \\
\qquad D;\, D' \mid P' \wedge Q' \wedge \ldots & & \text{(Shunting)}
\end{array}
$$

The first two lines show how to eliminate the "constraint" P, and obtain a more traditional form having no constraint part. Note that the elimination gives rise to a *conjunction* in case of existential quantification, and to an *implication* in case of a universal quantification.

> Remark. Newcomers to the field of predicate logic often erroneously write '$\forall\, D \bullet P \wedge Q$' (similarly to their correct use of the form $\exists\, D \bullet P \wedge Q$) when they actually mean $\forall\, D \bullet P \Rightarrow Q$. The reason is that they view P as a constraint upon the the domain of discourse D, and therefore treat P the same way in both quantifications. However, that is not possible with the two-part traditional notation '$D \bullet \ldots$' in contrast to the three-part Z notation '$D \mid P \bullet \ldots$'. Thus the three-part notation '$D \mid P \bullet \ldots$' is practically appealing.

The next two lines show, as a refresher, the familiar duality between universal and existential quantification. No surprise here.

The fifth and sixth line show the beauty of the '$D \mid P \bullet \ldots$' notation: the duality between universal and existential quantification holds even in the presence of a constraint P. In view of the elimination given in the first two lines, this might come as a surprise! In practice, it is often the case that in '$(\exists\, D \bullet P \wedge Q)$' and '$(\forall\, D \bullet P \Rightarrow Q)$' the parts P play the role of an additional constraint on the domain of interest D. By making that role explicit, and writing $(\exists\, D \mid P \bullet Q)$ and $(\forall\, D \mid P \bullet Q)$, respectively, we see that the duality respects those roles!

The one-but-last line shows one example of the interactions between various forms that involve a '$D \mid P \bullet \ldots$'; it assumes that the variables declared in D' do not occur free in P and ... . Thanks to the consistency of the Z-notation the declaration part D' of the existential quantification can be taken over into the declaration part of a set notation — without any change. Apart from this syntactic convenience, the line is also semantically interesting; we urge the reader to check (and understand) the equation. In fact, the rewriting is valid not only in a set context, but also in an arbitrary context:

$$
\begin{array}{lll}
 & D \quad \mid (\exists\, D' \mid P' \bullet Q') \wedge \ldots & \\
= & D;\, D' \mid P' \wedge Q' \wedge \ldots & \text{(Shunting)}
\end{array}
$$

We will apply the rule in a $\mathcal{P}$ context, thus showing the smooth transition between probability and set/predicate notation. There are some more such laws about the interaction between the D-part and P-part; e.g.:

$$x : S \mid x{\in}T \wedge \ldots \quad = \quad x : S{\cap}T \mid \ldots$$

Much more can be said about the Z notation, but this is not the place to do so. The interested reader may consult the Z literature [5] and Dijkstra [6] who has been a co-initiator of the three-part '$D \mid P \bullet \ldots$' notation.

4. EXAMPLE: CALCULATIONS

Recall our proposed notation for probabilities:

$$(\mathcal{P} D \mid P \bullet Q)$$

It denotes the probability that Q holds for a random draw from D that satisfies P. The traditional notation is $\mathcal{P}(Q \mid P)$, thus leaving the domain implicit and making it impossible to refer in P and Q to the variables declared in D. Unfortunately, the places of P and Q are reversed between the new and the traditional notation. Fortunately, the condition P immediately follows the vertical bar |.

In the current paragraph we do some calculations with this notation without referring to concrete examples; in the following paragraphs we'll actually use the notation for concrete examples.

When all sets involved are finite, and the probability distributions are uniform, we may take a frequentist view of probability, and put:

$$(\mathcal{P}\ D \mid P \bullet Q) \quad = \quad \frac{\#\{D \mid P \wedge Q\}}{\#\{D \mid P\}} \qquad \text{(Freq)}$$

Having done so, we are able to derive several theorems that are commonly taken as axioms about $\mathcal{P}$. We are the first to admit that these theorems are worthless when the above equation is false (because the probability distribution is not uniform) or doesn't make sense (because the sets are infinite); in that case we can still take the formulas below as axioms. However, we are mainly interested in the proofs, because *these show how our proposed notation nicely interacts with the set and predicate notation.* Here is a theorem that we shall use later on:

Let P_i, Q_i be predicates that do not use variables from D_j, for $j \neq i$. Then:

$$\begin{array}{ll} & (\mathcal{P}D_1;\, D_2 \mid P_1 \wedge P_2 \bullet Q_1 \wedge Q_2) \\ = & (\mathcal{P}D_1 \mid P_1 \bullet Q_1) \times (\mathcal{P}D_2 \mid P_2 \bullet Q_2) \qquad \text{(Independence)} \end{array}$$

Proof (using $\times$ for both number and cross product):

$$\begin{array}{ll} & (\mathcal{P}D_1;\, D_2 \mid P_1 \wedge P_2 \bullet Q_1 \wedge Q_2) \\ = & \qquad \text{Freq} \\ & \dfrac{\#\{D_1;\, D_2 \mid P_1 \wedge P_2 \wedge Q_1 \wedge Q_2\}}{\#\{D_1;\, D_2 \mid P_1 \wedge P_2\}} \\ = & \qquad \text{set calc.; premise} \\ & \dfrac{\#(\{D_1 \mid P_1 \wedge Q_1\} \times \{D_2 \mid P_2 \wedge Q_2\})}{\#(\{D_1 \mid P_1\} \times \{D_2 \mid P_2\})} \\ = & \qquad \text{set calc.} \\ & \dfrac{(\#\{D_1 \mid P_1 \wedge Q_1\} \times \#\{D_2 \mid P_2 \wedge Q_2\})}{(\#\{D_1 \mid P_1\} \times \#\{D_2 \mid P_2\})} \\ = & \qquad \text{nbr calc.} \\ & \dfrac{(\#\{D_1 \mid P_1 \wedge Q_1\}}{\#\{D_1 \mid P_1\})} \times \dfrac{(\#\{D_2 \mid P_2 \wedge Q_2\}}{\#\{D_2 \mid P_2\})} \\ = & \qquad \text{Freq} \\ & (\mathcal{P}D_1 \mid P_1 \bullet Q_1) \times (\mathcal{P}D_2 \mid P_2 \bullet Q_2) \end{array}$$

Many more properties can be proved in this *algebraic* way: decomposing the expression and composing it in another way while preserving the semantics — and in this case there is also a smooth switch between probability and set notation.

Since they are used in the sequel, we mention two further laws but leave the simple algebraic proofs to the industrious reader:

$$\begin{array}{ll} = & (\mathcal{P}D \mid P \bullet Q_1 \vee Q_2) \qquad \text{(Distribution)} \\ = & (\mathcal{P}D \mid P \bullet Q_1) + (\mathcal{P}D \mid P \bullet Q_2) - (\mathcal{P}D \mid P \bullet Q_1 \wedge Q_2) \\ = & (\mathcal{P}D \mid P \bullet Q_1) + (\mathcal{P}D \mid P \bullet Q_2), \text{ if } \forall\, D | P \bullet \neg\, (Q_1 \wedge Q_2) \end{array}$$

And the divide and conquer law:

$$\begin{array}{l} (\forall\, D \bullet P_1 \neq P_2) \;\Rightarrow \qquad \text{(Divide\&Conquer)} \\ \quad (\mathcal{P}D \bullet Q) \;= \\ \quad (\mathcal{P}D \bullet P_1)(\mathcal{P}D \mid P_1 \bullet Q) + (\mathcal{P}D \bullet P_2)(\mathcal{P}D \mid P_2 \bullet Q) \end{array}$$

Note that $P_1 \neq P_2$ means that P_1, P_2 are each others negation, that is, it is equivalent to $(P_1 \vee P_2) \wedge \neg\, (P_1 \wedge P_2)$, also known as exclusive-or.

5. EXAMPLE: INFORMATION RETRIEVAL

We set out to reformulate part of Section 2.3.3 of Hiemstra's PhD thesis[9] in our style and notation. We shall also make a comparison between our and his notation.

The scene is information retrieval. Here is a rough introduction. A set *Doc* of documents is given, and a user is in need for some relevant documents. The user poses a query to the system (a query is simply a set of *query terms*), and it is the systems task to rank the documents in order of increasing probability of being relevant (and then show the top-ranked documents to the user). The documents that contain the same query terms should be ranked the same. We bypass the problem of the way in which the set of relevant documents can be made known to the system.

With this introduction in mind, the following is our formalization. First, a set *Doc* of documents is postulated. Next, in order to avoid defining the internal structure of documents and queries, we postulate for each query q an equivalence relation $\approx_q$ on *Doc*, with the following interpretation:

$$\begin{array}{ll} = & d \approx_q d' \\ & \text{“}d \text{ contains the same query terms of } q \text{ as } d' \text{ does”} \end{array}$$

Now, the ranking function $rnk_{q,R}$ related to a query q and a set $R \subseteq Doc$ of “relevant” documents, is defined as follows:

$$rnk_{q,R}(d_0) \;=\; (\mathcal{P}\, d : Doc \mid d \approx_q d_0 \bullet d \in R)$$

In words: document d_0 is ranked with the probability that an arbitrary document with the same terms as d_0, happens to be relevant. (Much more can be said about the alternatives and variations for *rnk*, but this note is not the place to do so.)

Comparison with Hiemstra's formulation.
Hiemstra [9] formulates the ranking value as follows (the main ingredient of equation (2.8) in [9, page 19], see also [9, equation (2.10)]):

$$\mathcal{P}(L{=}1 \mid D_1, \cdots, D_n)$$

This is all the accompanying explanation:

- The domain of discourse is a set of documents; no further formalization is given. A document may be indexed with a query term, meaning that the query term occurs in the document. The query under consideration is supposed to have n query terms.
- Citing from page 19, line 10 from the bottom:
  Let L be the random variable “document is relevant” with a binary sample space {0,1}, 1 indicating relevance and 0 non-relevance.
- Citing again, line 8 from the bottom:
  Let D_k $(1 \leq k \leq n)$ be a random variable indicating “document belongs to the subset indexed with the k-th query term” with a binary sample space $\{0, 1\}$.
- Rephrasing line 6 from the bottom:
  A document satisfying a particular state of $D_1, \ldots, D_n$ is assigned the value given above (with the same state).

Although it is perfectly possible to do *numeric* calculations with such a probability notation, it is hard to take the ingredients of this expression and use them in set or predicate notation: the conventions and semantics of phrases like ‘$L = 1$’

and 'D_k' are just too far away from the conventions and semantics of set and predicate notation. Probability theory and set theory use quite different languages, here, whereas in our opinion that is not at all necessary.

6. RELAXED NOTATION

There is no objection against abbreviations in order to make the notation more compact. For example, when the discussion is about documents from the set *Doc* for pages and pages, then we may convene to omit the indication '$: Doc$' from the declaration part, thus writing:

$$(\mathcal{P}\, d \mid d \approx_q d_0 \bullet d \in R)$$

Going one step further, we may define $D(d) \Leftrightarrow d \approx_q d_0$ and $L(d) \Leftrightarrow d \in R$, and then write:

$$(\mathcal{P}\, d \mid D(d) \bullet L(d))$$

As another convention we might now suppress the bound variable and abbreviate this to:

$$(\mathcal{P} \mid D \bullet L)$$

This comes close to Hiemstra's notation $\mathcal{P}(L{=}\,1 \mid D_{1,\ldots,n})$. The point is that the semantics is still given by an expression of the form $(\mathcal{P}D \mid P \bullet Q)$, and if the need arises we can fall back to that form. Even with the above abbreviations we do not really leave the conventions of set and predicate notation.

7. EXAMPLE: TOSSING

The probability that head *and* tail turn up together with *one* throw of two *fair* coins is 0.5. To formalize the claim, let C be a set consisting of just two distinct symbols H and T, that is, $C = \{H, T\}$; letters C, H, and T are mnemonic for coin, head and tail. The claim then reads:

$$(\mathcal{P}x, y : C \bullet \{x, y\} = \{H, T\}) \quad = \quad 1/2$$

Proof. Fairness means that the probability distribution is uniform:

$$(\mathcal{P}x : C \bullet x{=}H) \;=\; (\mathcal{P}x : C \bullet x{=}T) \;=\; 1/2 \quad \text{(Fairness)}$$

Now:

$$\begin{array}{ll} & (\mathcal{P}x, y : C \bullet \{x, y\} = \{H, T\}) \\ & (\mathcal{P}x, y : C \bullet (x{=}H \wedge y{=}T) \vee (x{=}T \wedge y{=}H)) \\ = & \quad \text{distribution} \\ & (\mathcal{P}x, y : C \bullet x{=}H \wedge y{=}T) + (\mathcal{P}x, y : C \bullet x{=}T \wedge y{=}H) \\ = & \quad \text{Independence} \\ & (\mathcal{P}x : C \bullet x{=}H) \times (\mathcal{P}y : C \bullet y{=}T) \;+ \\ & (\mathcal{P}x : C \bullet x{=}T) \times (\mathcal{P}y : C \bullet y{=}H) \\ = & \quad \text{fairness} \\ & \frac{1}{2}\times\frac{1}{2} \;+\; \frac{1}{2}\times\frac{1}{2} \\ = & 1/2 \end{array}$$

Unfair coins can be dealt with analogously; simply give a different probability distribution.

8. EXAMPLE: EVENT SPACES

The problems and solutions discussed by Robertson [14] form yet another confirmation that our notation works well. Robertson essentially proposes to distinguish the various *event spaces* $\mathcal{E}$ that play a role, and to indicate them somehow in the notation $\mathcal{P}(X \mid Y)$, say as $\mathcal{P}_{\mathcal{E}}(X \mid Y)$. Here we cite the case described by Robertson (the table on the right is ours):

Example: we have stars $\mathcal{S}$, and planets $\mathcal{T}$. Stars either have (X=1) or do not have (X=0) magnetic fields. Planets either have (Y=1) or do not have (Y=0) magnetic fields.

The universe:

star	X	planet	Y
s_1	1	t_{11}	1
		t_{12}	0
s_2	0	t_{21}	0

Star s_2 has x_2=0; it has one planet t_{21} with y_{21}=0. We have a (complete) universe consisting of 2 stars and 3 planets. Star s_1 has x_1=1; it has two planets t_{11} and t_{12} with y_{11}=1 and y_{12}=0. In this universe, the following probabilities may be calculated exactly:

$$\begin{array}{lcl} \mathcal{P}(X{=}1) & = & \frac{1}{2} \\ \mathcal{P}(Y{=}1 \mid X{=}1) & = & \frac{1}{2} \\ \mathcal{P}(Y{=}1 \mid X{=}0) & = & 0 \end{array}$$

From these we would infer that $\mathcal{P}(Y{=}1) = \frac{1}{4}$; the inference uses the following law [a generalization of our Divide&Conquer, but written in conventional notation]:

$$\mathcal{P}(Y) \;=\; \sum_X \mathcal{P}(X)\mathcal{P}(Y \mid X) \qquad \text{(Div 'n Conq)}$$

But we have three planets, one of which has a magnetic field, so actually we have $\mathcal{P}(Y{=}1) = \frac{1}{3}$.

What is the problem here? In short, it is the event space. The laws of probability are written in terms of a single event space with a single probability measure on it; for historical reasons, the standard notation $\mathcal{P}(. \mid .)$ does not provide for the denotation of the event space.

Robertson proposes to denote the probability for a particular event space $\mathcal{E}$ as $\mathcal{P}_{\mathcal{E}}(. \mid .)$. Thus, writing $\mathcal{S}$ and $\mathcal{T}$ for the event space of stars and planets, respectively, he rewrites the calculated probabilities as:

$$\begin{array}{lcl} \mathcal{P}_{\mathcal{S}}(X{=}1) & = & \frac{1}{2} \\ \mathcal{P}_{\mathcal{T}}(Y{=}1 \mid X{=}1) & = & \frac{1}{2} \\ \mathcal{P}_{\mathcal{T}}(Y{=}1 \mid X{=}0) & = & 0 \end{array}$$

He even distinguishes some more event spaces: $\mathcal{S}^+$, $\mathcal{T}^+$, and $\mathcal{ST}$. Thanks to this notational distinction, it is immediately clear that the Div 'n Conq law cannot be applied to these probabilities; they apply to different event spaces, whereas the equation apparently assumes them to apply to the same event space.

In our notation, the event space is mentioned in the declaration part D. Writing $Z(z)$ for "celestial body z has a magnetic field" (there is no need to invent two names X

and Y for the same predicate!) and $star(t)$ for "the star of planet t", we thus have:

$$\begin{array}{lcl} (\mathcal{P}s : \mathcal{S} \bullet Z(s)) & = & \frac{1}{2} \\ (\mathcal{P}t : \mathcal{T} \mid Z(star(t)) \bullet Z(t)) & = & \frac{1}{2} \\ (\mathcal{P}s : \mathcal{S}; t : \mathcal{T} \mid Z(s) \bullet Z(t)) & = & \frac{1}{3} \\ (\mathcal{P}t : \mathcal{T} \mid \neg\, Z(star(t)) \bullet Z(t)) & = & 0 \\ (\mathcal{P}s : \mathcal{S}; t : \mathcal{T} \mid \neg\, Z(s) \bullet Z(t)) & = & \frac{1}{3} \end{array}$$

In our notation the calculation of "P(Y)" goes without error:

$$\begin{array}{ll} & (\mathcal{P}t : \mathcal{T} \bullet Z(t)) \\ = & (\mathcal{P}t : \mathcal{T} \mid true \bullet Z(t)) \\ = & (\mathcal{P}t : \mathcal{T} \mid (\exists\, s : \mathcal{S} \bullet Z(s) \vee \neg\, Z(s)) \bullet Z(t)) \\ = & \quad \text{(Shunting)} \\ & (\mathcal{P}t : \mathcal{T}; s : \mathcal{S} \mid Z(s) \vee \neg\, Z(s) \bullet Z(t)) \\ = & \quad \text{(Divide\&Conquer)} \\ & (\mathcal{P}t : \mathcal{T}; s : \mathcal{S} \mid Z(s) \bullet Z(t))(\mathcal{P}t : \mathcal{T}; s : \mathcal{S} \bullet Z(s)) + \\ & (\mathcal{P}t : \mathcal{T}; s : \mathcal{S} \mid \neg\, Z(s) \bullet Z(t))(\mathcal{P}t : \mathcal{T}; s : \mathcal{S} \bullet \neg\, Z(s)) \\ = & \quad \text{probabilities given above} \\ = & \frac{1}{3} \times \frac{1}{2} + \frac{1}{3} \times \frac{1}{2} \\ = & \frac{1}{3} \end{array}$$

If we want to discuss several probability distributions on the same space, we need to distinguish them in the notation, say by an index. Thus we may talk about $(\mathcal{P}_1\ D \mid P \bullet Q)$ and $(\mathcal{P}_2\ D \mid P \bullet Q)$ at the same time, with the same D, P and Q, while postulating different probability distributions for $\mathcal{P}_1$ and $\mathcal{P}_2$.

9. CONCLUSION

We have proposed a notation for probabilities that nicely interfaces with set and predicate notation, and other forms. The advantage is ease of understanding (similar aspects are denoted in the same way, in particular the aspect of free and bound variables), and ease of manipulations (transformation to and from set and predicate notation are possible without any change, and laws that already exists in the context of sets and predicates can now be applied as well). An important advantage of the conventional notation is its brevity. However, as we have shown, with suitable abbreviations we achieve the same brevity.

10. REFERENCES

[1] R.S. Bird. An introduction to the theory of lists. In M. Broy, editor, *Logic of Programming and Calculi of Discrete Design*, pages 3–42. Springer Verlag, 1987. Also Technical Monograph PRG–56, Oxford University, Computing Laboratory, Programming Research Group, October 1986.

[2] R.S. Bird. Shall we calculate? *The Squiggolist*, 1(1), 1989.

[3] R.S. Bird. Shall we calculate – II? *The Squiggolist*, 1(3), 1990.

[4] R.S. Bird and O. de Moor. *Algebra of Programming*. Prentice-Hall International, 1996.

[5] Jonathan Bowen. The Z notation web site. `http://www.comlab.ox.ac.uk/archive/z.html`.

[6] E.W. Dijkstra and C.S. Scholten. *Predicate Calculus and Program Semantics*. Springer Verlag, 1990.

[7] Maarten M. Fokkinga. An exercise in transformational programming: Backtracking and Branch-and-Bound. *Science of Computer Programming*, 16(1):19–48, July 1991.

[8] M.M. Fokkinga. Calculate categorically! *Formal Aspects of Computing*, 4(4):673–692, 1992. Obtainable by `ftp://ftp.cs.utwente.nl/pub/doc/Parlevink/fokkinga/mmf91j.ps.Z`.

[9] Djoerd Hiemstra. *Using Language Models for Information Retrieval*. PhD thesis, University of Twente, Netherlands, 2001. CTIT Ph.D. Thesis Series, No. 01-32.

[10] L. Meertens. Algorithmics — towards programming as a mathematical activity. In J.W. de Bakker and J.C. van Vliet, editors, *Proceedings of the CWI Symposium on Mathematics and Computer Science*, pages 289–334. North-Holland, 1986.

[11] Lambert Meertens. Calculate polytypically! In H. Kuchen and S. D. Swiestra, editors, *Proceedings 8th Int. Symp. on Programming Languages: Implementations, Logics, and Programs, PLILP'96, Aachen, Germany, 24–27 Sept 1996*, volume 1140 of *Lecture Notes in Computer Science*, pages 1–16. Springer-Verlag, Berlin, 1996.

[12] E. Meijer, M.M. Fokkinga, and R. Paterson. Functional programming with bananas, lenses, envelopes and barbed wire. In *FPCA91: Functional Programming Languages and Computer Architecture*, volume 523 of *Lect. Notes in Comp. Sc.*, pages 124–144. Springer Verlag, 1991. Obtainable by `http://www.cs.utwente.nl/~fokkinga/mmf91m.ps`.

[13] Jeff Miller. Earliest Uses of Various Mathematical Symbols. `http://members.aol.com/jeff570/mathsym.html`.

[14] Stephen Robertson. On Baysian models and event spaces in information retrieval. Technical note, authors e-maili: `ser@microsoft.com`, May 2002.

[15] J.M. Spivey. *The Z notation: a reference manual (2nd edition)*. Prentice Hall International, UK, 1992.

APPENDIX

A. HISTORY OF THE NOTATION

The following has been taken literally from `http://members.aol.com/jeff570/stat.html` [13], a web-page about the origin of symbols in mathematics.

Apart from the combinatorial symbols very little of the notation of modern probability dates from before the 20th century.

Probability. Symbols for the probability of an event A on the pattern of $P(A)$ or $Pr(A)$ are a relatively recent development given that probability has been studied for centuries.

A.N. Kolmogorov's *Grundbegriffe der Wahrscheinlichkeitsrechnung* (1933) used the symbol $\mathrm{P}(A)$. The use of upper-case letters for events was taken from set theory. H. Cramér's *Random Variables and Probability Distributions* (1937), "the first modern book on probability in English," used $P(A)$. In the same year J.V. Uspensky (*Introduction to Mathematical Probability*) wrote simply (A). W. Feller's influential *An Introduction to Probability Theory and its Applications volume 1* (1950) uses $Pr\{A\}$ and $\mathrm{P}\{A\}$ in later editions. See also the "Earliest Uses of Symbols of Set Theory and Logic" page of this website [13].

Conditional probability. Kolmogorov's (1933) symbol for conditional probability ("die bedingte Wahrscheinlichkeit") was $\mathrm{P}_B(A)$. Cramér (1937) referred to the "relative probability" and wrote $\mathrm{P}_B(A)$. Uspensky (1937) used the term "conditional probability" with the symbol (A, B). The vertical stroke notation $Pr\{A \mid B\}$ was made popular by Feller (1950), though it was used earlier by H. Jeffreys. In his *Scientific Inference* (1931) $P(p \mid q)$ stands for "the probability of the proposition p on the data q." Jeffreys mentions that Keynes and Johnson, earlier Cambridge writers, had used p/q; Jeffreys himself had used $P(p : q)$. The symbols p and q came from Whitehead and Russell's *Principia Mathematica*. See also the "Earliest Uses of Symbols of Set Theory and Logic" page of this website [13].

Random variable. The use of upper and lower case letters to distinguish a random variable from the value it takes, as in $Pr\{X = x_j\}$, became popular around 1950. The convention is used in Feller's *Introduction to Probability Theory*.

Modeling Uncertain Context Information via Event Probabilities

Arthur H. van Bunningen
bunninge@cs.utwente.nl

Ling Feng
ling@cs.utwente.nl

Peter M.G. Apers
apers@cs.utwente.nl

University of Twente
PO Box 217, 7500AE
Enschede, The Netherlands

ABSTRACT

To be able to support context-awareness in an Ambient Intelligent (AmI) environment, one needs a way to model context. Previous research shows that a good way to model context is using Description Logics (DL). Since context data is often coming from sensors and therefore exhibits uncertain character, it is essential to take this uncertainty into account when exploiting and reasoning about context data. In this paper, we provide an event-based probabilistic method to model context uncertainty. Each context can be viewed as a probabilistic event, which can be either a basic or complex event. We show how to deal with correlations of events (i.e., inclusion, identicalness, disjunction, and independence) which are inherent in context data and investigate their influences on the context uncertainty.

1. INTRODUCTION

To support an Ambient Intelligent (AmI) environment where users have ubiquitous access to data anytime and anywhere in such a way that it is a natural part of the environment, data management systems need to recognize changes in its environment and adapt the presented information accordingly to the context of its users. In other words, the data management systems need to be context-aware [18, 16].

To deliver such a context-aware data management system, the problem arises of how to capture low-level information about the environment and interpret it in terms of that accurately reflect human perception of tasks and needs. Moreover, since most context data is acquired from sensor data, it is easily subject to both systematic and random errors [7]. Despite systematic errors could be corrected by calibration techniques, random errors inevitably result in *uncertainty*. According to [7], sources of such random errors include: noise from external sources, random hardware noise, inaccuracies in measurement technique, environmental effects, and imprecision in computing a derived value from the underlying measurements. Furthermore, in some cases uncertainty may even be desirable in order to provide privacy for individuals. This uncertainty links to the quality of available context, and negatively influences context-aware database solutions. To resolve the influence of context uncertainty, as a first step, we need a way to model context and its uncertainty.

In our previous work [8], we proposed to view context from two perspectives: *user-centric* and *environmental*. Examples of user-centric contexts are: user's background (e.g., working area, friends, private doctor, etc.), behavior (activity, intention, etc.), physiological state (temperature, heart rate, etc.), and emotional state (happy, sad, fear, etc.). Environmental contexts can be physical environment (e.g., location, time, humidity, lightness, etc.), social environment (e.g., traffic jam, surrounding people, etc.), and computational environment (e.g., surrounding devices, etc.). A knowledge-based approach for context modeling was further presented in [17], where we explored a variant of Description Logics to model both a world model and users' information preferences, leading to personalized query answering in an AmI world.

The aim of this paper is to investigate the uncertainty inherent in context and develop an event-based probabilistic approach to model context uncertainty to support context-aware data management.

The rest of the paper is organized as follows. Section 2 reviews our previous knowledge-based context model based on Description Logics, and outlines the limitation of probabilistic Description Logics in handling context uncertainty. An event-based probabilistic model is then presented in Section 3 to represent context uncertainty. The influences of event correlations on context uncertainty are examined in Section 4. We conclude the paper in Section 5.

2. CONTEXT MODELING USING DESCRIPTION LOGICS

In the literature, there exist several attempts to model context [13, 15], where [13] concludes that *ontology based languages* are preferable for context modeling. In [17], we explored a variant of Description Logics (DL) to model context for several reasons. First, DL [1] is a (decidable) fragment of first order logic, and is especially suited for knowledge rep-

resentation. This will bene t the development of intelligent data management systems, capable of *adaptiveness, learning, inference,* and *anticipation,* etc., as demanded by the AmI environment. Second, DL forms the basis of ontological languages such as OWL, which has been used to model context in [5]. Furthermore, there exist many tools for dealing with DL knowledge bases such as reasoners and editors. Finally, extensive research has been conducted to investigate the relationship between databases and DL, and map a DL knowledge base into database schemas [3]. In fact, in a number of situations, DL can be used to express the information requests upon large amounts of data stored in existing relational databases [4]. Borgida in [4] presented a system that converts most inferences made by the knowledge base management systems into SQL queries, while keeping an object-centred view and relying on the optimization of the underlying DBMS to gain efficiency.

2.1 A Brief Review of Description Logics

As known, a DL knowledge base consists of a TBox and an ABox. The TBox (i.e., the vocabulary) contains assertions about *concepts* (e.g., Child, Female) and *roles* (e.g., hasRoom, hasActivityType). The ABox contains assertions about *individuals* (e.g., ROOM3061). Concepts and roles can be either *atomic* or *constructed* using concept and role constructors *intersection* ($\sqcap$), *union* ($\sqcup$), and *complement* ($\neg$) (e.g., Child $\sqcap$ Female, hasRoom $\sqcap$ hasActivityType). A concept speci c constructor is the *one-of* construct ($\{a_1, .., a_n\}$) which de nes a concept consisting of a speci c set of individuals $\{a_1, .., a_n\}$. The *top concept* ($\top$) and *bottom concept* ($\bot$) denote respectively all individuals and no individuals. A role speci c operator is the *role-inverse* which de nes the inverse of a certain role (e.g., $roomOf$ is the inverse of $hasRoom$, denoted as $hasRoom \equiv roomOf^{-1}$). Moreover, roles can have full quanti cation, existential quanti cation and number restrictions (e.g., $\forall$ hasChild.Female denotes the individuals of whose children are all female, $\exists$ hasChild.Female denotes the individuals having a female child, and $= 1\ hasChild.Female$ denotes the individuals having exactly one Female child). A *concept expression* contains a set of concepts and/or quanti ed roles which are connected via concept and role constructors. The basic inference on concept expressions in DL is *subsumption* $C \sqsubseteq D$, which is to check whether the concept denoted by D (the *subsumer*) is more general than the one denoted by C (the *subsumee*).

DL can be used to describe a world model (i.e., ontology). A small ontology example is given in Figure 1, where concepts are denoted in CamelCase (e.g. *FreeTimeActivity*) , roles in lowerCamelCase (e.g. *hasRoom*), and instances in all capital letters (e.g. *ROOM3061*).

In our study, we use a DL concept expression to depict a context $\mathcal{C}$. We assume a function *ce* which projects a context $\mathcal{C}$ to a DL concept expression.

As an example of a concept expression, consider a context $\mathcal{C}$ where Eric is drinking co ee either in his own room (room 3061) or in the co eeroom. In a context-aware system/application, this situation could, for example, imply that Eric is available for a meeting. We can represent this context via a DL concept expression $Q = ce(\mathcal{C})$ as follows:

$$Q \equiv \{ERIC\} \sqcap \exists hasActivityType.Relaxing \sqcap \exists hasRoom.\{COFFEEROOM, ROOM3061\} \quad (1)$$

One goal of context-aware data management systems is to determine whether this context is satis ed or not, that is, whether there is at least an instance in the knowledge base which is included in the concept expression Q, $(\exists a(Q(a)))$. When the sensed context data possesses uncertainty, the goal then becomes to calculate the *probability* that there exists at least an instance which is included in the concept expression Q, $(P(\exists a(Q(a))) \in [0, 1])$.

2.2 Limitations of Probabilistic Description Logics in Addressing Uncertainty

Previous work on probabilistic reasoning in DL (e.g. [12, 11, 10]) mainly focused on de ning probabilities on domains. An example of such kind of probability is: "*The probability that a random chosen student is drinking coffee is greater than 0.3*", or as a probabilistic TBox assertion:

$$P(\text{DrinkingCo ee}|\text{Student}) > 0.3$$

These assertions state statistical information and usually result from an experiment or trial. In our research, however, we are interested in probabilities on assertions coming from sensors, i.e., the so-called *degree of belief*. An example of such a probability is: "*The probability that Eric (a particular student) is drinking coffee is greater than* 0.7"; or as a probabilistic ABox assertion:

$$P(\text{DrinkingCo ee}(\text{ERIC})) > 0.7$$

In the latter, we implicitly assume multiple possible worlds with a probability over these possibilities, properties may hold in some worlds and not in others, and the probability of the set of possible worlds in which Eric is drinking co ee is greater than 0.7.

In [11], Jaeger combines both probabilities on domains and degrees of belief by means of cross-entropy minimization, but does not allow expressing probabilistic knowledge on role instances. Furthermore, because he took into account both types of probabilities, his method becomes too complex for our purpose. A good overview of di erent approaches on dealing with uncertainty using DL is given in [2].

3. AN EVENT-BASED PROBABILISTIC MODEL FOR CONTEXT UNCERTAINTY

Since we are interested in uncertainty of DL concept and role inclusions and only consider DL ABox assertions which may contain uncertainty, we explore the probabilistic database techniques to tackle the probability calculation of concept and role inclusions.

In the database eld, there exist two ways for dealing with uncertainty in the data models, namely, *extensional semantics* and *intentional semantics*. The former approach is to modify the operators of the algebra to compute probabilities. The problem of this approach is that it ignores most correlations between intermediate results and may give di erent results depending on the query plan [6]. These problems

$$
\begin{aligned}
Person &\sqsubseteq Thing \sqcap = 1\ hasRoom.Room \\
&\sqcap \forall hasActivityType.ActivityType \\
&\sqcap \forall hasFriend.Person \\
&\sqcap \forall hasTvInterest.Genre \\
Room &\sqsubseteq Location \\
TVProgram &\sqsubseteq Thing \sqcap \exists hasGenre.Genre \\
hasRoom &\equiv roomOf^{-1}
\end{aligned}
\qquad
\begin{aligned}
ActivityType &\sqsubseteq Thing \\
FreeTimeActivity &\sqsubseteq ActivityType \\
Relaxing &\sqsubseteq FreeTimeActivity \\
Sporting &\sqsubseteq FreeTimeActivity \\
Location &\sqsubseteq Thing \\
Genre &\sqsubseteq Thing \\
hasTvInterest &\equiv tvInterestOf^{-1}
\end{aligned}
$$

Figure 1: A small ontology example in DL.

are circumvented if *intentional semantics* are used, which means keeping track of the events that contribute to a derived fact and determining the probability of the resulting event expression. This is sometimes considered impractical since the event expressions can become very large and calculating the probability from the event expressions is a NP-complete problem.

In the scope of our study, we regard correlations and constraints that exist among concepts and roles highly desirable in an AmI environment (e.g., a person can only be at a single place at one moment). Thus, it is important to capture and model these correlations without approximations. Additionally, some e ective optimization techniques have already been developed for *intentional semantics* [9]. For these considerations, we decide to follow the same line as the *intentional semantics*. In the following, we rst reexamine the notion of probabilistic event developed in [9], and then describe an event-based probabilistic approach for representing context uncertainty.

3.1 Interpreting Context using Event Expressions

The basic idea of our context uncertainty model is to view each context as a probabilistic event, which can be either basic event or complex event. The event expressions of inclusion in an atomic concept $ee(D(a))$ or role $ee(R(a,b))$, as shown in Table 2 correspond a basic event. A combination of basic events forms a *complex event*, corresponding to a DL concept expression where concepts and/or roles are connected via the DL constructors. The DL special top concept $\top$ and bottom concept $\bot$ correspond to the special basic event $\top_e$ (denoting a certain event) and $\bot_e$ (denoting an impossible event), respectively. Each basic event is uniquely identi ed by an event identi er. Let E be the whole set of basic event identi ers, including $\bot_e$ and $\top_e$.

We introduce a recursive function ee which maps a a DL concept expression to an event expression. The de nition of the function is pictorially described in Table 1.

Using this function, we can obtain the event expression for the inclusion of an instance a in the DL concept expression Q from equation 1:

DL Inclusion	event expression
$D(a)$	basic event
$R(a,b)$	basic event
$\top$	$\top_e$
$\bot$	$\bot_e$
$\{b_1,\ldots,b_n\}(a)$	$\top_e$, if $a \in \{b_1,\ldots,b_n\}$
	$\bot_e$, otherwise
$(\neg D)(a)$	$\neg ee(D(a))$
$(D \sqcap E)(a)$	$ee(D(a)) \wedge ee(E(a)))$
$(D \sqcup E)(a)$	$ee(D(a)) \vee ee(E(a)))$
$(\exists R.D)(a)$	$\bigvee_b (ee(R(a,b)) \wedge ee(D(b)))$
$(\forall R.D)(a)$	$\neg(\bigvee_b (ee(R(a,b)) \wedge \neg ee(D(b))))$

Table 1: Recursive de nition of function ee, mapping a DL concept expression to an event expression

$$
\begin{aligned}
ee(Q(a)) = \\
&(a \in \{ERIC\}) \\
&\wedge \bigvee_b (ee(hasActivityType(a,b)) \wedge ee(Relaxing(b))) \\
&\wedge \bigvee_c (ee(hasRoom(a,c)) \\
&\qquad \wedge (c \in \{COFFEEROOM, ROOM3061\}))
\end{aligned}
$$

In this expression we left out the translation of the one-of construct since it will always result in $\top_e$ or $\bot_e$. We can immediately see that there is room for optimization, such as dealing with the boolean result of the one-of-constructs, which leads to the expression:

$$
\begin{aligned}
&\bigvee_b (ee(hasActivityType(ERIC,b)) \wedge ee(Relaxing(b))) \\
&\wedge \bigvee_{c \in \{COFFEEROOM, ROOM3061\}} (ee(hasRoom(ERIC,c)))
\end{aligned}
\qquad (2)
$$

Based on the DL semantics, Table 2 gives a knowledge base example which complies to the ontology in Figure 1, together with all the basic event identi ers and their associated probabilities. Via the equation 2, we can derive the following event expression for the DL concept expression in equation 1.

Concept Inclusion	Basic Event e	$\beta(e)$
$Person(ERIC)$	$\top_e$	1
$Person(PETER)$	$\top_e$	1
$Person(MAARTEN)$	$\top_e$	1
$Relaxing(READING)$	$\top_e$	1
$Relaxing(SLEEPING)$	$\top_e$	1
$Relaxing(DRINKINGCOFFEE)$	$\top_e$	1
$Room(ROOM3061)$	$\top_e$	1
$Room(ROOM4061)$	$\top_e$	1
$Room(COFFEEROOM)$	$\top_e$	1

(a) Concept inclusion

Role Inclusion	Basic Event e	$\beta(e)$
$hasActivityType(ERIC, READING)$	$hAc1$	0.3
$hasActivityType(MAARTEN, SLEEPING)$	$hAc2$	0.7
$hasActivityType(ERIC, DRINKINGCOFFEE)$	$hAc3$	0.8
$hasRoom(ERIC, ROOM3061)$	$hRo1$	0.4
$hasRoom(ERIC, ROOM4061)$	$hRo2$	0.3
$hasRoom(ERIC, COFFEEROOM)$	$hRo3$	0.3

(b) Role inclusion

Table 2: DL Role and concept inclusions with (basic) event identi ers and probabilities.

$$((hAc1 \wedge \top_e) \vee (hAc3 \wedge \top_e)) \wedge (hRo1 \vee hRo3)$$
$$= (hAc1 \vee hAc3) \wedge (hRo1 \vee hRo3) \quad (3)$$

3.2 Using Probabilistic Events to Represent Context Uncertainty

In our uncertainty model, only the probabilities of basic events are given explicitly via a *basic event probability assignment function* β, satisfying the conditions

1. $\beta(\bot_e) = 0$.
2. $\beta(\top_e) = 1$.
3. $(\forall e \in (E \quad \{\bot_e, \top_e\}))\ (0 < \beta(e) < 1)$.

From these, probabilities of complex events can be computed. In general, probabilities for event expressions are given by a general probability assignment function P which maps an event expression to a value in [0, 1].

The inclusion probability of an individual instance a belonging to a concept D is thus given as the P of the event expression for the inclusion, i.e., $P(D(a)) = P(ee(D(a)))$.

According to Table 2, if we assume independence among all the tuples (that is, $P(e_1 \vee e_2) = P(e_1) + P(e_2) \quad P(e_1) * P(e_2)$ and $P(e_1 \wedge e_2) = P(e_1) * P(e_2)$), we can calculate the probability of the event expression 3:

$$P((hAc1 \vee hAc3) \wedge (hRo1 \vee hRo3))$$
$$= (0.3 + 0.8 \quad 0.3 * 0.8) * (0.4 + 0.3 \quad (0.4 * 0.3))$$
$$= 0.4988$$

This is the uncertainty level of the context expression that "*Eric is drinking coffee either in his own room (room 3061) or in the coffeeroom.*" We will discuss the result in case of non-independence in Section 4.

3.3 Properties of Event Probabilities

We describe the uncertainty of a context (i.e., a DL concept expression) in terms of the probability of the corresponding event expression. According to the world model (i.e., ontology) in use, the probabilities of DL instance concept/role inclusions observe the following properties:

$$C \sqsubseteq D \implies \forall a(P(ee(C(a))) \quad P(ee(D(a))))$$
$$C \equiv D \implies \forall a(P(ee(C(a))) = P(ee(D(a))))$$
$$R \equiv S \implies \forall a, b(P(ee(R(a,b))) = P(ee(S(a,b))))$$
$$R \sqsubseteq S \implies \forall a, b(P(ee(R(a,b))) \quad P(ee(S(a,b))))$$

where C and D are DL concepts, R and S are DL roles, and a and b are DL concept instances.

When multiple sensors give conflicting context information, these properties could facilitate the normalization of probabilities. A speci c case is that when the ontology de nes a *number* restriction for a DL role such as:

$$Person \sqsubseteq = 1\ hasRoom.Room$$

and we know that instance a is a person. This means that $P(ee(Person(a))) = 1$ and implies:

$$P(ee((= 1\ hasRoom.Room)(a))) \geq 1$$

We interpret this as that for each possible world, instance a should be in exactly *one* room, and hence, two events indicating two di erent rooms for a will be disjoint. Furthermore, the instance a *should* be in one room in every possible world and thus the probabilities of the events of a being in di erent rooms should add up to 1.

In principle, our *uncertain context model* is supported by the

three pillar functions, that is, function *ce* (which projects a context to a DL concept expression), function *ee* (which maps a DL concept expression to an event expression); and P (which associates a probability with an event expression).

4. INFLUENCES OF EVENT CORRELATIONS ON CONTEXT UNCERTAINTY

Given the fact that many important properties in the domain of context-awareness are usually dependent of each other (e.g. a person can only be at a single location at one time), it is necessary to take into account the correlations among basic events. Often these correlations are determined from statistical analysis which is impractical to do when deploying a context-aware system. However, many useful dependencies for context-awareness can already be acquired from the ontology using inference. In our current study, we consider the following four event correlations:

Disjointness when two events by de nition cannot happen at the same time. E.g., if someone is at a certain room, s/he cannot be at another room at the same time.

Inclusion a special kind of positive correlation where in case one event happens, by de nition, the other happens as well. E.g., if someone is at a room, then s/he is also located in the building which contains this room.

Identicalness when two events by de nition always happen together. E.g., one could de ne that a person is drinking co ee, if and only if s/he is in the co ee room. In this situation, the event "*drinking coffee* and "*being in the coffee room* are identical.

Independence when the occurrence of one event does not say anything about the occurrence of the other. E.g., the TV interests of a person are (probably) not dependent on the room that s/he is in.

We will take these dependencies into account when calculating probabilities of events. The rst three correlations can be inferred according to the DL ontology. When none of these three is speci ed, we assume the independence relationship.

Table 3 illustrates the influence of correlations between two events (e_1 and e_2) on probability calculation. It is adapted from [14].

Correlation	Probability
Disjointness	$P(e_1 \wedge e_2) = 0$
Negatively correlated	$P(e_1 \wedge e_2) < P(e_1) * P(e_2)$
Independent	$P(e_1 \wedge e_2) = P(e_1) * P(e_2)$
Positively correlated	$P(e_1 \wedge e_2) > P(e_1) * P(e_2)$
Identical	$P(e_1 \wedge e_2) = P(e_1) = P(e_2)$

Table 3: Event correlation vs. probability adapted from [14]

For practical purposes, we will only consider dependencies between two events. There could be dependencies among more events de ned by the ontology, however this would mean for each event expression of n events, comparing a maximum of $\frac{1}{2}2^n$ combinations instead of the maximum of $\binom{n}{2}$ when only considering dependencies between two events. Furthermore, dependencies among more than two events are much less likely to have a big influence on the resulting probability of the event expression.

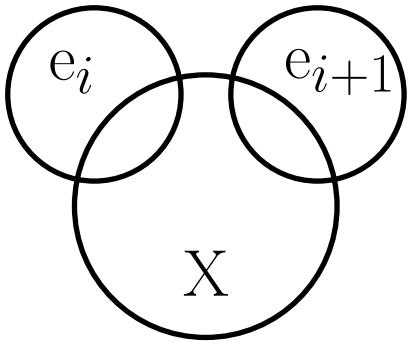


Figure 2: Correlation of two disjoint basic events e_i and e_{i+1} and an event expression X.

To derive these correlations from the ontology, we can for each two events construct two concepts and query the reasoner if they are disjunct, identical, or included in each other. For example, for the events $hRo1$ and $hRo3$ as taken from Table 2b, we could ask the reasoner if the concepts $\{ERIC\} \sqcap \exists hasRoom.\{ROOM3061\}$ and $\{ERIC\} \sqcap \exists hasRoom.\{COFFEEROOM\}$ are disjunct. This is the case since according to the ontology a person can only be at one room ($Person \sqsubseteq = 1\ hasRoom.Room$) and we assume that distinct individual names denote distinct objects (*unique naming assumption*) which means $ROOM3061$ is di erent from $COFFEEROOM$. Although calculating these correlations seems impractical for large numbers of instances and events, we expect optimization strategies such as the pre-calculation of disjunctions or addressing multiple dependent events at the same time to speed up this process to make it worthwhile. In this paper, we only show that proper dependency calculation is possible.

Now that we explained how inclusion, disjunction, and identicalness can be acquired from the ontology, we have to look at the influence of these correlations on the calculation of event probabilities. For this, we rewrite the resulting complex event to its equivalent disjunctive normal form. *Disjunctive normal form* (DNF) is a disjunction consisting of one or more conjunctions of one or more literals where the not operator can only be used as part of a literal (e.g. $(e_1 \wedge e_2 \wedge e_3) \vee (e_1 \wedge \neg e_2 \wedge e_3) \vee (e_1 \wedge e_2 \wedge \neg e_3)$). We require each conjunct to contain each basic event exactly once either in positive or in negative form, which makes them so-called *complete conjuncts* [9]. Since all conjuncts are complete, they will be disjoint and the probability of the complex event is equal to the sum of probabilities of the di erent conjuncts.

Since each literal is equal to a basic event or its negation, we address the dependencies by using rewrite rules on the probability of the complete conjuncts. These rewrite rules

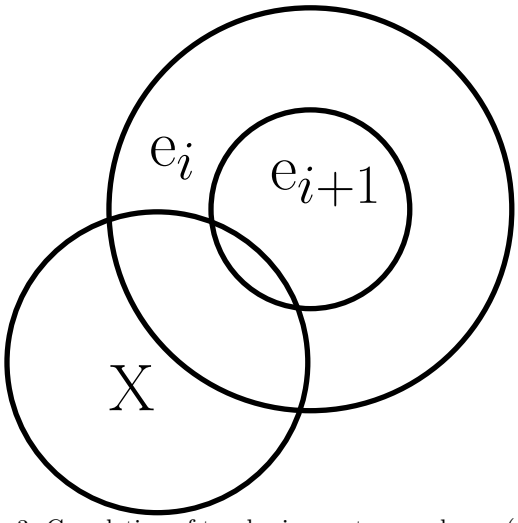


Figure 3: Correlation of two basic events e_i and e_{i+1} (where e_i includes e_{i+1}) and an event expression X.

are straightforward applications of basic probability theory. For this, we compare each event with all other events in its conjunct. Let e_i and e_{i+1} be two basic event expressions, and X can be either a basic or complex event expression. In case an event e_i is disjoint from event e_{i+1} (see Figure 2), we apply the following rewrite rules:

$$P(X \wedge \neg e_i \wedge \neg e_{i+1}) = P(X) - (P(X \wedge e_{i+1}) + P(X \wedge e_i)) \quad (4)$$

$$P(X \wedge \neg e_i \wedge e_{i+1}) = P(X \wedge e_{i+1}) \quad (5)$$

$$P(X \wedge e_i \wedge \neg e_{i+1}) = P(X \wedge e_i) \quad (6)$$

$$P(X \wedge e_i \wedge e_{i+1}) = 0 \quad (7)$$

If event e_i includes event e_{i+1} (see Figure 3) the rewrite rules are as follows:

$$P(X \wedge \neg e_i \wedge \neg e_{i+1}) = P(X \wedge \neg e_i)$$
$$P(X \wedge \neg e_i \wedge e_{i+1}) = 0$$
$$P(X \wedge e_i \wedge \neg e_{i+1}) = P(X) \quad P(X \wedge e_i) + P(X \wedge e_{i+1})$$
$$P(X \wedge e_i \wedge e_{i+1}) = P(X \wedge e_i)$$

Finally, if the events e_i and e_{i+1} are identical, we rewrite the probability of the conjunct as follows:

$$P(X \wedge \neg e_i \wedge \neg e_{i+1}) = P(X \wedge \neg e_i)$$
$$P(X \wedge \neg e_i \wedge e_{i+1}) = 0$$
$$P(X \wedge e_i \wedge \neg e_{i+1}) = 0$$
$$P(X \wedge e_i \wedge e_{i+1}) = P(X \wedge e_i)$$

We apply these rewrite rules as long as there are correlated basic events in one of the conjuncts. Since in each step basic events are removed from the conjuncts, after applying the above rules, the basic events in each of the conjuncts will be independent of each other. In this case we can use techniques for independent events [9] where the probability of each conjunct is calculated by taking the product of the probability of the literals contained in the conjunct. The probability of a literal is equal to the one minus the probability of its basic event (as given by β) if the basic event is negated and equal to the probability of the basic event, otherwise. After obtaining the probabilities of all the conjuncts, we can sum them up to determine the probability of the complex event.

As an example, let's look at the complex event in the previous section $(hAc1 \vee hAc3) \wedge (hRo1 \vee hRo3)$. According to the ontology of Figure 1 in this expression $hRo1$ and $hRo3$ are disjoint and we will consider other events to be independent. If we rewrite this expression in DNF using complete conjuncts, we get:

$$\begin{aligned}
&(hAc1 \wedge hRo3 \wedge hAc3 \wedge hRo1)\\
&\vee (hAc1 \wedge hRo3 \wedge hAc3 \wedge \neg hRo1)\\
&\vee (hAc1 \wedge hRo3 \wedge \neg hAc3 \wedge hRo1)\\
&\vee (hAc1 \wedge hRo3 \wedge \neg hAc3 \wedge \neg hRo1)\\
&\vee (hAc1 \wedge hRo1 \wedge hAc3 \wedge \neg hRo3)\\
&\vee (hAc1 \wedge hRo1 \wedge \neg hAc3 \wedge \neg hRo3)\\
&\vee (hAc3 \wedge hRo1 \wedge \neg hAc1 \wedge hRo3)\\
&\vee (hAc3 \wedge hRo1 \wedge \neg hAc1 \wedge \neg hRo3)\\
&\vee (hAc3 \wedge hRo3 \wedge \neg hAc1 \wedge \neg hRo1)
\end{aligned}$$

The probability of this expression will be equal to:

$$\begin{aligned}
&P(hAc1 \wedge hRo3 \wedge hAc3 \wedge hRo1) && (8)\\
&+ P(hAc1 \wedge hRo3 \wedge hAc3 \wedge \neg hRo1) && (9)\\
&+ P(hAc1 \wedge hRo3 \wedge \neg hAc3 \wedge hRo1) && (10)\\
&+ P(hAc1 \wedge hRo3 \wedge \neg hAc3 \wedge \neg hRo1) && (11)\\
&+ P(hAc1 \wedge hRo1 \wedge hAc3 \wedge \neg hRo3) && (12)\\
&+ P(hAc1 \wedge hRo1 \wedge \neg hAc3 \wedge \neg hRo3) && (13)\\
&+ P(hAc3 \wedge hRo1 \wedge \neg hAc1 \wedge hRo3) && (14)\\
&+ P(hAc3 \wedge hRo1 \wedge \neg hAc1 \wedge \neg hRo3) && (15)\\
&+ P(hAc3 \wedge hRo3 \wedge \neg hAc1 \wedge \neg hRo1) && (16)
\end{aligned}$$

Conjuncts where we have the disjoint terms $hRo1$ and $hRo3$ in positive form evaluate to zero according to equation 7. This means that the probabilities of term 8, 10, and 14 are zero, leading to the following equation:

$$
\begin{aligned}
&0\\
&+ P(hAc1 \wedge hRo3 \wedge hAc3 \wedge \neg hRo1)\\
&+ 0\\
&+ P(hAc1 \wedge hRo3 \wedge \neg hAc3 \wedge \neg hRo1)\\
&+ P(hAc1 \wedge hRo1 \wedge hAc3 \wedge \neg hRo3)\\
&+ P(hAc1 \wedge hRo1 \wedge \neg hAc3 \wedge \neg hRo3)\\
&+ 0\\
&+ P(hAc3 \wedge hRo1 \wedge \neg hAc1 \wedge \neg hRo3)\\
&+ P(hAc3 \wedge hRo3 \wedge \neg hAc1 \wedge \neg hRo1)
\end{aligned}
$$

Furthermore, we can apply equation 5 and 6 to remove from the conjunctions in which one of the disjoint terms ($hRo1$ and $hRo3$) appears in positive form and the other one in negative form, the negative one:

$$
\begin{aligned}
&P(hAc1 \wedge hRo3 \wedge hAc3)\\
&+ P(hAc1 \wedge hRo3 \wedge \neg hAc3)\\
&+ P(hAc3 \wedge hRo3 \wedge \neg hAc1)\\
&+ P(hAc1 \wedge hRo1 \wedge hAc3)\\
&+ P(hAc1 \wedge hRo1 \wedge \neg hAc3)\\
&+ P(hAc3 \wedge hRo1 \wedge \neg hAc1)
\end{aligned}
$$

In the resulting equation, there are no more ontology-related correlations between two events. Because we assume independence among the rest of the events, we can multiply the probability of the basic events contained in the probabilities resulting in the following expression:

$$
\begin{aligned}
&0.3 * 0.3 * 0.8\\
&+ 0.3 * 0.3 * (1 \quad 0.8)\\
&+ 0.8 * 0.3 * (1 \quad 0.3)\\
&+ 0.3 * 0.4 * 0.8\\
&+ 0.3 * 0.4 * (1 \quad 0.8)\\
&+ 0.8 * 0.4 * (1 \quad 0.3)\\
&= 0.602
\end{aligned}
$$

This result is, as expected, more than the result given in the previous section, where independence was assumed among all the basic events.

5. CONCLUDING REMARKS

In this paper we presented an approach for dealing with uncertainty in context information using event probabilities. For this we adapted the method from [9] of using intentional semantics for use with Description Logics. This way of looking at uncertainty gives us the opportunity to correctly model correlations between events, which are considered especially important for context information. Furthermore, we showed how these correlations can be modeled in, and acquired from, a Description Logics ontology. Finally, we demonstrated how these correlations can lead to rewrite rules for event expressions to determine the resulting probability of a complex event.

Our future work will focus on testing the feasibility of the approach by applying it to realistic data sets and context descriptions, which will include looking at optimization strategies. In addition, we will use the results of this study to realize context-aware data management systems under the assumption of uncertain context.

6. REFERENCES

[1] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. Patel-Schneider, editors. *The Description Logic Handbook : theory, implementation, and applications*. Cambridge University Press, 2003.

[2] F. Baader, R. Küsters, and F. Wolter. Extensions to description logics. In F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. Patel-Schneider, editors, *The Description Logic Handbook*, pages 219–261. Cambridge University Press, 2003.

[3] A. Borgida. Description logics in data management. *IEEE Transactions on Knowledge and Data Engineering*, 7(5):671–682, 1995.

[4] A. Borgida and R. Brachman. Loading data into description reasoners. In *SIGMOD '93: Proceedings of the 1993 ACM SIGMOD international conference on Management of data*, pages 217–226, New York, NY, USA, 1993. ACM Press.

[5] H. Chen, T. Finin, and A. Joshi. The soupa ontology for pervasive computing. In *Ontologies for Agents: Theory and Experiences*. Springer, (Whitestein Series in Software Agent Technologies), June 2005.

[6] N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. In M. Nascimento, T. Özsu, D. Kossmann, R. Miller, J. Blakeley, and B. Schiefer, editors, *(e)Proceedings of the Thirtieth International Conference on Very Large Data Bases, Toronto, Canada, August 31 - September 3 2004*, pages 864–875. Morgan Kaufmann, 2004.

[7] E. Elnahrawy and B. Nath. Cleaning and querying noisy sensors. In *WSNA '03: Proceedings of the 2nd ACM international conference on Wireless sensor networks and applications*, pages 78–87, New York, NY, USA, 2003. ACM Press.

[8] L. Feng, P. Apers, and W. Jonker. Towards context-aware data management for ambient intelligence. In *Conference on Database and Expert Systems Applications*, pages 422–431, 2004.

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

[10] R. Giugno and T. Lukasiewicz. P-shoq(d): A probabilistic extension of shoq(d) for probabilistic ontologies in the semantic web. In S. Flesca, S. Greco, N. Leone, and G. Ianni, editors, *Logics in Artificial*

Intelligence, European Conference, JELIA 2002, Cosenza, Italy, September, 23-26, Proceedings, pages 86–97. Springer, 2002.

[11] M. Jaeger. Probabilistic reasoning in terminological logics. In *KR*, pages 305–316, 1994.

[12] D. Koller, A. Levy, and A. Pfe er. P-classic: A tractable probablistic description logic. In *AAAI/IAAI*, pages 390–397, 1997.

[13] T. Strang. A context modeling survey. In *Workshop on Advanced Context Modelling, Reasoning and Management associated with the Sixth International Conference on Ubiquitous Computing (UbiComp 2004)*, September 2004.

[14] D. Suciu and N. Dalvi. Foundations of probabilistic answers to queries. In *SIGMOD '05: Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 963–963, New York, NY, USA, 2005. ACM Press.

[15] A. van Bunningen. Context aware querying - challenges for data management in ambient intelligence. Technical Report TR-CTIT-04-51, University of Twente, P.O. Box 217, December 2004.

[16] A. van Bunningen, L. Feng, and P. Apers. Context for ubiquitous data management. In *International Workshop on Ubiquitous Data Management (UDM2005)*, April 2005.

[17] A. van Bunningen, L. Feng, and P. Apers. A context-aware preference model for database querying in an Ambient Intelligent environment. In *Proceedings of the 17th International Conference on Database and Expert Systems Applications (DEXA)*, to appear.

[18] M. Weiser. The computer for the 21st century. *Scientific American*, 265(3):94–104, 1991.

I have a dream...

Ignorance in the Relational Model

Maarten M. Fokkinga
DB group, dept INF, fac EWI, University of Twente
PO Box 217, NL 7500 AE Enschede, Netherlands
m.m.fokkinga@utwente.nl

Abstract. We hypothesize that an extension-with-conditioning of Dempster-Shafer theory is suitable for encoding uncertainty and ignorance in the Relational Model. We present a formal and well-motivated definition of conditioning, and show the spirit of the required change in the Relational Model and some results that then follow. It remains to be investigated whether these results are satisfactory.

Introduction

1 Ignorance
Ignorance is closely related to uncertainty. Commonly, we say that a property is *uncertain* if it is not considered true or false but, instead, it is assigned a probability of being true. Now consider a set of exhaustive and mutually disjoint properties. Probability theory requires that the probabilities assigned to these properties add up to 1. *Ignorance* is the phenomenom that the "probabilities" do not add up to 1. Formally, one axiom of probability theory is not fulfilled, and hence we speak of *belief* instead of probability. Dempster-Shafer theory gives a proper formalization (summarized in paragraph 8–10), and we shall build upon that theory (paragraph 11–15).

2 Setting
Our work is an attempt to improve upon Choenni *et al.* [1, 2] in the following aspects: a more fundamental approach and better motivated definition of *conditioning*, and a better formalization of an *extension* of the Relational Model in order to deal with ignorance. We borrow the following example from Choenni [1], and take it to be the *leading* example.

3 Example: CIA
The ship type department of the CIA has 0.6 evidence that the type of ship *Maria* is *Frigate*, and 0.3 evidence that it is *Tugboat*; for the remaining 0.1 there is ignorance. This is encoded in the one-row table *SHIP* below at the left. The type speed department of the CIA has evidence that 30% of the frigates has a max speed of 20 knots, and 70% has 30 knots, whereas all tugboats have a 15 knots max speed. This is encoded in the two-row table *DESC*ription at the right:

SHIP

Name	*Type*	
Maria	*Frigate*	$\mapsto 0.6$
	Tugboat	$\mapsto 0.3$
	$*$	$\mapsto 0.1$

DESC

Type	*Speed*	
Frigate	$20K$	$\mapsto 0.3$
	$30K$	$\mapsto 0.7$
Tugboat	$15K$	$\mapsto 1.0$

For the purpose of decision making, the US government requests to join the information. Here are two candidate results that they might get offered:

join candidate 1

Name	*Type*	*Speed*	
Maria	*Frigate*	$20K$	$\mapsto 0.18$
	Frigate	$30K$	$\mapsto 0.42$
	Tugboat	$15K$	$\mapsto 0.3$
	$*$	$*$	$\mapsto 0.1$

join candidate 2

Name	*Type*	*Speed*	
Maria	*Frigate*	$20K$	$\mapsto 0.18$
	Frigate	$30K$	$\mapsto 0.42$
	Tugboat	$*$	$\mapsto 0.3$
	$*$	$20K$	$\mapsto 0.03$
	$*$	$30K$	$\mapsto 0.07$
Maria	*Frigate*	$*$	$\mapsto 0.6$
	Tugboat	$15K$	$\mapsto 0.3$
	$*$	$15K$	$\mapsto 0.1$

The one-row table *join candidate* 1 is obtained by "intuitive combination". However, the information in this table is *too weak* in the sense that the probability of "the max speed of *Maria* is $20K$" has an upperbound (when all ignorance goes to this case) of $0.18 + 0.1 = 0.28$, whereas that upperbound is $0.6{\times}0.3 + 0.1{\times}0.3 = 0.21$ according the original *SHIP* and *DESC*.

The two-row table *join candidate* 2 is proposed by Choenni *et al.* [1]. This information is *too strong*: The first row of the table expresses that the probability of "the max speed of *Maria* is $20K$" has a lowerbound (when all ignorance about the speed is not in favor of this case) of $0.18 + 0.03 = 0.21$, whereas that lowerbound is only $0.6{\times}0.3 = 0.18$ according to *SHIP* informally joined with *DESC*.

4 Goal, plan

Our goal is to extend the Relational Model and relational operators (like projection, selection, and in particular the join) in such a way that we can offer the US government the right information. Moreover, we should also be able to relate in a formal way the above candidate joins to "the correct join" of *SHIP* and *DESC*. The next paragraph gives the outline of the theory that we want to develop, and paragraph 6 discusses the previous example in the theory that we envisage.

5 Hypothesis, focus

In order to deal with uncertainty and ignorance, Dempster has weakened probability theory to what currently is known as *Dempster-Shafer theory*. The primary notion is *bpa* (basic probability assignment), from which the notions of *belief*, *plausibility*, and *ignorance* can be defined; and conversely. Our *hypothesis* is that an extension of Dempster-Shafer theory provides a solution for the problem how to deal with uncertainty and ignorance in the Relational Model, and we want to investigate this hypothesis. The main line, then is as follows.

In the leading example of paragraph 3, we start out with bpa's as *attribute values* in the table, and observe that all our attempts for a join lead to a table with a "bpa covering several attributes", which we call *tupled-bpa*, or just *t-bpa* for short. It can be shown that this generalization (*our* generalization!) of bpa to t-bpa is not essential: t-bpa's can be expressed as bpa's (though at considerable loss of readability), and vice versa. Continuing with taking joins of the resulting table with other tables will lead to tables in which t-bpa's cover more and more attributes. Therefore we generalize the notion of relation right away to one where *each row is a t-bpa*.

Moreover, we see that in Choenni's attempt the failure is due to the omission of a *condition* in the bpa: the 0.03 evidence for max speed $20K$ cannot be given unconditionally, but is only valid if it is known that the type is *Frigate*. Therefore we generalize right away to one where each row is a "conditioned t-bpa", or *ct-bpa* for short; the definition of "conditioned bpa" (or conditioned t-bpa) is new and is the focus is this paper. This notion of conditioning differs entirely from the notion of conditioning brie y discussed by Shafer [3], from the notion defined by Choenni [1, 2], and from the notion of conditioning as known in probability theory.

6 Envisaged solution

Once the *conditioning* (and *tupling*) extension to Dempster-Shafer theory has been developed and, based on this, also a new Relational Model, we expect to be able to deal formally with the example in the following way.

To deal with uncertainty and ignorance is quite straightforward: let each row in each table be a ct-bpa. We call such relations: *ui-relations*, where the letters 'ui' derive from 'uncertainty and ignorance'. For example, we encode the one-row table *SHIP* of paragraph 3 as an ui-relation with one row (that is, one ct-bpa) with the following pretty-print:

$SHIP'$

Name	*Type*	*Name*	*Type*	
*	*Frigate*	*Maria*	*	$\mapsto 0.6$
*	*Tugboat*	*Maria*	*	$\mapsto 0.3$
*	*	*Maria*	*	$\mapsto 0.1$

Fully written out the relation reads as in Figure 1.

Each star, , is pronounced "*unknown*" and stands for the entire domain of the corresponding attribute: the *Name*-star stands for $\{Maria, \ldots\}$, the *Type*-star stands for $\{Frigate, Tugboat, \ldots\}$, and so on. The meaning of the first line of the above one-row relation is, roughly: "there is evidence 0.6 for that the type is *Frigate* on the condition that the ship is *Maria*. More precisely, the line means:

> There is evidence 0.6 for the property
> $(Name, Type) \subseteq (unknown, \{Frigate\})$
> on the condition that
> $(Name, Type) \subseteq (\{Maria\}, unknown)$
> is true.

Here it is understood that $(U, V) \subseteq (X, Y)$ means: $U \subseteq X \wedge V \subseteq Y$.

So, the one-row *ui-relation* $SHIP'$ given above encodes that the ship type department of the CIA has 0.6 evidence that the type of a ship is *Frigate if its name is Maria*, and 0.3 evidence that it is *Tugboat*; for the remainder there is ignorance.

Further, the type speed department of the CIA has evidence that 30% of the frigates has a max speed of 20 knots, and 70% has 30 knots, whereas all tugboats have a 15 knots max speed. This is encoded in the two-row ui-relation with the following pretty-print:

$DESC'$

Type	*Speed*	*Type*	*Speed*	
*	$20K$	*Frigate*	*	$\mapsto 0.3$
*	$30K$	*Frigate*	*	$\mapsto 0.7$
*	$15K$	*Tugboat*	*	$\mapsto 1.0$

For the purpose of decision making, the US government requests to join the information. Here is what they get:

$SHIP' \bowtie DESC'$

Name	*Type*	*Speed*	*Name*	*Type*	*Speed*	
*	*Frigate*	$20K$	*Maria*	*	*	$\mapsto 0.18$
*	*Frigate*	$30K$	*Maria*	*	*	$\mapsto 0.42$
*	*Tugboat*	*	*Maria*	*	*	$\mapsto 0.3$
*	*	$20K$	*Maria*	*Frigate*	*	$\mapsto 0.03$
*	*	$30K$	*Maria*	*Frigate*	*	$\mapsto 0.07$
*	*Frigate*	*	*Maria*	*	*	$\mapsto 0.6$
*	*Tugboat*	$15K$	*Maria*	*	*	$\mapsto 0.3$
*	*	$15K$	*Maria*	*Tugboat*	*	$\mapsto 0.1$

Note the last two lines of the first row, and the last line of the second row: these say that the ship type is unknown but yet the speed is certain to some degree *if* the type happens to be frigate or tugboat, respectively. (And if

$$
\begin{array}{l}
\{ \\
(\{Name \mapsto *, \quad Type \mapsto \{Frigate\}\} \quad | \quad \{Name \mapsto \{Maria\}, \quad Type \mapsto *\}) \quad \mapsto 0.6, \\
(\{Name \mapsto *, \quad Type \mapsto \{Tugboat\}\} \quad | \quad \{Name \mapsto \{Maria\}, \quad Type \mapsto *\}) \quad \mapsto 0.3, \\
(\{Name \mapsto *, \quad Type \mapsto \quad * \quad\} \quad | \quad \{Name \mapsto \{Maria\}, \quad Type \mapsto *\}) \quad \mapsto 0.1 \\
\}
\end{array}
$$

Figure 1: *Fully written-out one-row ui-relation SHIP′ (see paragraph 6).*

the condition is not fulfilled, the evidence supports just *unknown* — see paragraph 12.) In order to eliminate this fine-grained conditioned information and obtain information that is somewhat easier to understand for the US government, we can weaken each row by "condition-restricting it to $\{Name\}$", that is, replacing all conditions except *Name* by *unknown*, which we will be able to denote formally by $(\{Name\}\blacktriangleleft_c) * (SHIP' \bowtie DESC')$:

Name	*Type*	*Speed*	*Name*	*Type*	*Speed*	
*	*Frigate*	20*K*	*Maria*	*	*	$\mapsto 0.18$
*	*Frigate*	30*K*	*Maria*	*	*	$\mapsto 0.42$
*	*Tugboat*	*	*Maria*	*	*	$\mapsto 0.3$
*	*	*	*Maria*	*	*	$\mapsto 0.03{+}0.07$
*	*Frigate*	*	*Maria*	*	*	$\mapsto 0.6$
*	*Tugboat*	15*K*	*Maria*	*	*	$\mapsto 0.3$
*	*	*	*Maria*	*	*	$\mapsto 0.1$

Taking the "least upper bound" of the two rows gives the following one-row ui-relation:

Name	*Type*	*Speed*	*Name*	*Type*	*Speed*	
*	*Frigate*	20*K*	*Maria*	*	*	$\mapsto 0.18$
*	*Frigate*	30*K*	*Maria*	*	*	$\mapsto 0.42$
*	*Tugboat*	15*K*	*Maria*	*	*	$\mapsto 0.3$
*	*	*	*Maria*	*	*	$\mapsto 0.1$

As observed in paragraph 3 this information is too weak.

Phrased in our concepts, the kind of join that Choenni [1] (and [2]?) proposes, yields for $SHIP'$ and $DESC'$ the following ui-relation (again, compare with paragraph 3):

Name	*Type*	*Speed*	*Name*	*Type*	*Speed*	
*	*Frigate*	20*K*	*Maria*	*	*	$\mapsto 0.18$
*	*Frigate*	30*K*	*Maria*	*	*	$\mapsto 0.42$
*	*Tugboat*	*	*Maria*	*	*	$\mapsto 0.3$
*	*	20*K*	*Maria*	*	*	$\mapsto 0.03$
*	*	30*K*	*Maria*	*	*	$\mapsto 0.07$
*	*Frigate*	*	*Maria*	*	*	$\mapsto 0.6$
*	*Tugboat*	15*K*	*Maria*	*	*	$\mapsto 0.3$
*	*	15*K*	*Maria*	*	*	$\mapsto 0.1$

As observed in paragraph 3 this information is too strong.

The well-known Dempster-Shafer theory

Although in the running example there are three domains (*Name*, *Type*, and *Speed*), Dempster-Shafer theory deals only with one domain. This restriction is without loss of generality, as discussed in paragraph 15.

7 Notational conventions

We consider functions to be *sets of argument-result pairs*, and use the notation $x \mapsto y$ as a suggestive synonym for the pair (x, y). So, the set $\{a \mapsto 3, b \mapsto 2, c \mapsto 3\}$ is a function that maps a to 3, maps b to 2, and c to 3. For summation we use a notation without subscripting:

$$
\begin{array}{ll}
\Sigma\, x, y \bullet expr(x, y) & \Sigma\, x, y \mid cond(x, y) \bullet expr(x, y) \\
= \Sigma_{x,y}\, expr(x, y) & = \Sigma_{x,y \ s.\,th.\ cond(x,y)}\, expr(x, y)
\end{array}
$$

We do so because in some cases the '$x, y \mid cond(x, y)$' part is just too large to be written as a subscript (see for example paragraph 13).

In the context of a set D we let P, Q vary over subsets of D, that is, $P, Q : \mathbb{P}\, D$, and we sometimes write $*$ for D.

8 Basic probability assignment

For the reader not familiar with Dempster-Shafer theory, we provide an intuition in the appendix paragraph 18. We build on this intuition later when we generalize the theory with conditioning.

Let D be a finite set. A *basic probability assignment over* D, abbreviated *bpa*, is a total function $m : \mathbb{P}\, D \longrightarrow [0, 1]$ satisfying:

$$
\begin{array}{lcll}
m\, P & = & 0 & \text{whenever } P = \varnothing \\
\Sigma\, P \bullet m\, P & = & 1 &
\end{array}
$$

The latter equation means that the sum of values $m\, P$, for all subsets P of D, equals 1.

Sometimes, a bpa is called a mass function, hence letter m for bpa's. We omit the entries $\{\ldots\} \mapsto 0$ in the presentation of a bpa. A bpa m induces a *belief* Bel, a *plausibility* Pl, and an *ignorance* Ig as total functions of type $\mathbb{P}\, D \longrightarrow [0, 1]$ as follows:

$$
\begin{array}{lcl}
Bel\, P & = & \Sigma\, P' \mid P' \subseteq P \bullet m\, P' \\
Pl\, P & = & 1 - Bel(D \setminus P) \\
Ig\, P & = & Pl\, P - Bel\, P
\end{array}
$$

A single value $d \in D$ may be considered as a bpa, namely the bpa m_d that maps every $P \subseteq D$ to zero except $P = \{d\}$, that is, $m_d = \{\{d\} \mapsto 1\}$.

9 Example

Department m of the CIA has 0.6 evidence that the type of a certain ship is *Frigate*, and 0.3 evidence that it is *Tugboat*; for the remainder there is ignorance. The department is characterized as follows, as a bpa over $\{Frigate, Tugboat, \ldots\}$:

$$
m \quad = \quad \{\{Frigate\} \mapsto 0.6,\ \{Tugboat\} \mapsto 0.3,\ * \mapsto 0.1\}
$$

Recall that $*$ stands for the full set $\{Frigate, Tugboat, \ldots\}$.

10 Combination of bpa's
Dempster defines a combination $\oplus$ of bpa's, now commonly known as *Dempster's combination rule*, or *orthogonal sum*. We give the formal definition here, and our intuition in appendix paragraph 19. We build on this intuition later when we generalize the theory with conditioning.

Let m_1 and m_2 be bpa's over D. If constant κ, defined below, equals 0, then the combination of m_1 and m_2 is said not to exist; if κ differs from 0, then the combination $m_1 \oplus m_2$ is a bpa over D defined as follows:

$$
\begin{array}{l}
(m_1 \oplus m_2)\, P = 0 \qquad \text{if } P = \varnothing, \quad \text{else:} \\
(m_1 \oplus m_2)\, P = (\Sigma\, P_1, P_2 \mid P_1 \cap P_2 = P \bullet m_1\, P_1 \times m_2\, P_2)/\kappa \\
\text{where} \\
\kappa \qquad\qquad = (\Sigma\, P_1, P_2 \mid P_1 \cap P_2 \neq \varnothing \bullet m_1\, P_1 \times m_2\, P_2)
\end{array}
$$

It is easily checked that whenever $m_1 \oplus m_2$ is defined, it is a bpa; notice that κ equals "the sum of all $m_1 \oplus m_2$-results if normalization '$/\kappa$' were left out of $\oplus$'s definition". More precisely, κ = "the above $(\Sigma \ \ldots)$ except that part '$P_1 \cap P_2 = P$' is extended with 'for some $P \neq \varnothing$'. "

Generalization: Conditioning (and Tupling)

11 Conditioned bpa
Let D be a finite set. A *conditioned bpa over D*, *c-bpa* for short, is a total function $m : D \times D \longrightarrow [0, 1]$ such that:

$$
\begin{array}{lcll}
m\,(P \mid Q) & = & 0 & \text{whenever} \quad P \cap Q = \varnothing \\
\Sigma\, P, Q \bullet m\,(P \mid Q) & = & 1 &
\end{array}
$$

Consistent with common practice in probability theory, we separate the two arguments of a c-bpa with a '|' rather than a comma, and interpret the second one as the condition and the first one as the conclusion.

The belief, plausibility, and ignorance induced by m are defined as follows:

$$
\begin{array}{lcl}
\mathit{Bel}\,(P \mid Q) & = & \Sigma\, P', Q' \mid P' \subseteq P \wedge Q' \supseteq Q \bullet m\,(P' \mid Q') \\
\mathit{Pl}\,(P \mid Q) & = & 1 - \mathit{Bel}\,(D \setminus P \mid Q) \\
\mathit{Ig}\,(P \mid Q) & = & \mathit{Pl}\,(P \mid Q) - \mathit{Bel}\,(P \mid Q)
\end{array}
$$

12 Interpretation
A c-bpa m is interpreted as an agent, having *conditioned* evidences. Specifically, the statement $m\,(P \mid Q) = x$ is interpreted as follows:

> Agent m has evidence x supporting *just* P provided that a proposition from Q is true; if the condition is not fulfilled, the evidence supports just D unconditionally.

Due to the exhaustiveness of the set D of propositions, there cannot be any evidence for $P \mid Q$ in case $P \cap Q$ is empty. The interpretation of the condition plays also an important role in the combination of two c-bpa's.

13 Combination of c-bpa's
Let m_1 and m_2 be c-bpa's over D. If constant κ defined below equals 0, then the combination of m_1 and m_2 is said not to exist; if κ differs from 0, then the combination $m_1 \oplus m_2$ is a c-bpa over D defined as follows — *with an indispensable(!) explanation following the de nition*:

$$
\begin{array}{l}
(m_1 \oplus m_2)(P \mid Q) \ = \ 0 \qquad \text{if } P \cap Q = \varnothing \quad \text{else:} \\
(m_1 \oplus m_2)(P \mid Q) \ = \\
\quad (\Sigma\ P_1, Q_1, P_2, Q_2 \\
\qquad \mid \qquad P_1' \cap P_2' = P \quad \wedge \quad Q_1' \cap Q_2' = Q \\
\qquad\quad \text{where} \\
\qquad\quad P_1', Q_1' = (\textbf{if} \qquad Q_1 \subseteq D \setminus P_2 \ \textbf{then} \ *\ , \ * \\
\qquad\qquad\qquad\quad \textbf{if } P_2 \subseteq Q_1 \not\subseteq D \setminus P_2 \ \textbf{then} \ P_1,\ * \\
\qquad\qquad\qquad\quad \textbf{if } P_2 \not\subseteq Q_1 \not\subseteq D \setminus P_2 \ \textbf{then} \ P_1, Q_1\ ), \\
\qquad\quad P_2', Q_2' = (\textbf{if} \qquad Q_2 \subseteq D \setminus P_1 \ \textbf{then} \ *\ , \ * \\
\qquad\qquad\qquad\quad \textbf{if } P_1 \subseteq Q_2 \not\subseteq D \setminus P_1 \ \textbf{then} \ P_2,\ * \\
\qquad\qquad\qquad\quad \textbf{if } P_1 \not\subseteq Q_2 \not\subseteq D \setminus P_1 \ \textbf{then} \ P_2, Q_2\ ) \\
\qquad \bullet\ m_1\,(P_1 \mid Q_1) \times m_2\,(P_2 \mid Q_2) \\
\quad)\ /\ \kappa
\end{array}
$$

The first clause is an immediate consequence of the observation in paragraph 12. We shall now explain the second clause. Let P, Q be arbitrary with non-empty intersection. The summation constraint in between '|' and '•' characterizes the possible $P_1 \mid Q_1$ and $P_2 \mid Q_2$ that *in combination* support just $P \mid Q$; precisely for these P_1, Q_1, P_2, Q_2 the product $m_1(P_1 \mid Q_1) \times m_2(P_2 \mid Q_2)$ is taken into the summation for $P \mid Q$. We explain the characterization of P_1, Q_1 only; the characterization of P_2, Q_2 is similar.

> In the general case, the evidence that agents 1 holds in support for $P_1 \mid Q_1$ will, *in combination with the other agent*, support just $P_1 \cap ... \mid Q_1 \cap ...$ (where the dots ... stand for the contribution of the other agent): the intersection in the conclusion is the same one as for normal bpa's, and the intersection in the condition expresses a conjunction of the conditions. However, for agent 1 there are two circumstances that lead to a change of its contribution in the combination.

- First, agent 1's condition Q_1 might be *in*consistent with the other agents conclusion P_2 ($Q_1 \cap P_2 = \varnothing$ or, equivalently, $Q_1 \subseteq D \setminus P_2$). The interpretation of paragraph 12 says: "if the condition is not fulfilled, the evidence supports just D unconditionally." So, agent 1's evidence supports, *in the combination*, just $P_1' \cap ... \mid Q_1' \cap ...$ where $P_1', Q_1' = *, *$. This is covered by the first branch for P_1', Q_1'.
- Second, agent 1's condition Q_1 might be implied by the other agents conclusion P_2 ($P_2 \subseteq Q_1$ or, equivalently, $P_2 \subseteq Q_1 \not\subseteq D \setminus P_2$). In the combination, then, condition Q_1 is fulfilled and may be weakened to $*$. So, agent 1's evidence supports, *in the combination*, just $P_1' \cap ... \mid Q_1' \cap ...$ where $P_1', Q_1' = P_1, *$. This is covered by the second branch for P_1', Q_1'.

  > *Note.* Recall that the condition of m_1 requests the *truth* of a member in Q_1 whereas $P_2 \subseteq Q_1$ only asserts some *evidence* supporting Q_1. Yet, the condition is discarded, by putting $Q_1' = *$. This is justified since the combination $m_1 \oplus m_2$ doesn't assert any truth but only some evidence.

- In the remaining ("general") case, as we have said above, agent 1's evidence supports, *in the combination*, just $P_1' \cap ... \mid Q_1' \cap ...$ where $P_1', Q_1' = P_1, Q_1$. This is covered by the third branch for P_1', Q_1'.

It remains to define κ; it must make the total sum over $m_1 \oplus m_2$ equal to one. Hence, κ is defined to be "the sum of all $m_1 \oplus m_2$-results if normalization '$/\kappa$' were left out of $\oplus$'s definition". Equivalently, take $\kappa =$ "the above $(\Sigma \ldots)$ except that part '$P_1' \cap P_2' = P \wedge Q_1' \cap Q_2' = Q$' is extended with 'for some P, Q with $P \cap Q \neq \varnothing$'. "

14 Example
Take D to be a set of numbers, partitioned into *small* and *large*, with *tiny* $\subset$ *small* and *huge* $\subset$ *large*, and let *even*, *odd*, *prime* have their conventional meaning. Consider the following c-bpa's:

$$\begin{array}{lcl} m_1 & = & \{(\mathit{small} \mid *) \mapsto 0.4, \quad (\mathit{huge} \mid \mathit{prime}) \mapsto 0.6\} \\ m_2 & = & \{(\mathit{even} \mid \mathit{large}) \mapsto 0.3, \quad (\mathit{odd} \mid \mathit{tiny}) \mapsto 0.7\} \end{array}$$

Then the combination $m_1 \oplus m_2$ is computed as follows:

$\mathit{small} \mid *$	0.4	$\mathit{small} \cap * \mid * \cap *$	$\mathit{small} \cap \mathit{odd} \mid * \cap \mathit{tiny}$
$\mathit{huge} \mid \mathit{prime}$	0.6	— (see note †)	$\mathit{huge} \cap * \mid \mathit{prime} \cap *$
		0.3	0.7
$m_1 \Uparrow \quad m_2 \Longrightarrow$		$\mathit{even} \mid \mathit{large}$	$\mathit{odd} \mid \mathit{tiny}$

Note †: $\mathit{huge} \cap \mathit{even} \mid \mathit{prime} \cap *$ is discarded since $\mathit{huge} \cap \mathit{even} \cap \mathit{prime} \cap * = \varnothing$

For the upper-left rectangle of the "square", note that m_1's condition $*$ is implied by m_2's conclusion *even* (that is, $* \supseteq \mathit{even}$), so m_1's evidence for $\mathit{small} \mid *$ is dealt with as $\mathit{small} \mid *$ in the combination (the $*$ is discarded and replaced by $*$); further, m_2's condition *large* is inconsistent with m_1's conclusion *small* (they have an empty intersection), hence m_2's evidence for $\mathit{even} \mid \mathit{large}$ is dealt with in the combination as $* \mid *$. Similarly for the lower-right rectangle. For the upper-right rectangle, note that m_1's condition $*$ is implied by m_2's conclusion *odd* (that is, $* \supseteq \mathit{odd}$), so that m_1's evidence for $\mathit{small} \mid *$ is dealt with as $\mathit{small} \mid *$ in the combination (the $*$ is discarded and replaced by $*$); and further, m_2's condition *tiny* is not implied by m_1's conclusion *small* and these two are not inconsistent ($\mathit{tiny} \cap \mathit{small} \neq \varnothing$), so m_2's evidence for $\mathit{odd} \mid \mathit{tiny}$ is dealt with unchanged in the combination. For the lower-left rectangle, we have the same situation as for the upper-right rectangle. However, since $\mathit{huge} \cap \mathit{even} \cap \mathit{prime} \cap * = \varnothing$ (that is, "$P \cap Q = \varnothing$" — there are no huge even primes), the evidence for the combined case $\mathit{huge} \cap \mathit{even} \mid \mathit{prime} \cap *$ must be zero by definition of the notion of c-bpa. All together:

$$\begin{array}{lcl} m_1 \oplus m_2 & = & \{\, (\mathit{small} \mid *) \mapsto 0.12/\kappa, \\ & & \quad (\mathit{small} \cap \mathit{odd} \mid \mathit{tiny}) \mapsto 0.28/\kappa, \\ & & \quad (\mathit{huge} \mid \mathit{prime}) \mapsto 0.42/\kappa \ \} \\ & & \text{where} \\ & & \kappa = 0.12 + 0.28 + 0.42 \end{array}$$

15 Tupling
So far, the formal definitions assume that there is a single domain of discourse, D. However, in the CIA-example there are clearly several distinct domains: *Name*, *Type*, and *Speed*. In order to deal with such a situation, we need to extend Dempster-Shafer theory, and our generalization with conditioning, in such a way that several domains can be dealt with simultaneously. This is achieved by the notion of "tupled-bpa" (*t-bpa*, for short). It is not hard to do so, but space limitations do not permit us to give the details.

In fact, the notion of t-bpa is super uous in the sense that normal bpa's can already express (although in a rather complicated and unpractical way) what we wish to express with t-bpa's. This came for us as a little surprise, because in general $\mathbb{P}\, D_1 \times \cdots \times \mathbb{P}\, D_n$ and $\mathbb{P}(D_1 \times \cdots \times D_n)$ are quite different; the explanation, however, is that the former can be *embedded* in the latter. For example, take t-bpa m over $D = (D_1, D_2)$ as follows:

$$m \quad = \quad \{\ldots, \quad (\{a, b, c\}, \{p, q\}) \mapsto x, \quad \ldots\}$$

This m can be viewed as denoting the following normal bpa m' over $D' = D_1 \times D_2$:

$$\begin{array}{l} m' \\ = \{..., \ \{a, b, c\} \times \{p, q\} \mapsto x, \ ...\} \\ = \{..., \ \{(a, p), (a, q), (b, p), (b, q), (c, p), (c, q)\} \mapsto x, \ ...\} \end{array}$$

So, m' maps a subset P of $D_1 \times D_2$ to 0 except when P happens to be equal to $\pi_1 P \times \pi_2 P$, in which case $m'\, P = x$ iff $m(\pi_1 P, \pi_2 P) = x$ — where π_i is the projection of a set of tuples to coordinate i, that is, $\pi_i\, P = \{(x_1, x_2) : P \bullet x_i\}$. Formally, the normal bpa m' that represents the t-bpa m, is defined as follows:

$$m'\, P \quad = \quad \textbf{if } P = \pi_1\, P \times \pi_2\, P \textbf{ then } m\,(\pi_1\, P, \pi_2\, P) \textbf{ else } 0$$

The construction is fully general, as can be proved formally.

Similarly for the combination of conditioning and tupling: ct-bpa. Again, space restrictions do not permit us to give the details.

Extending the Relational Model

Having generalized Dempster-Shafer theory with conditioning and extended it with tupling as well, the formal definitions of the well-known classical Relational Model and our new one look very similar — even for the join operation. This is one half of our goal (the other half being the condition that conditioning expresses indeed what we intuitively wish to express). We want to show the *similarity* here, in particular for the join operation, without intending or attempting to explain the formulas. For the die-hards that nevertheless *do* want to understand every detail (which is not necessary to observe the similarity!), we provide some missing definitions in the appendix: paragraph 21–23.

16 The classical Relational Model
Let (A, D) be a *schema* (definition omitted). A *relation over* (A, D) is a subset of $\Pi_A D$ (definition omitted). We let R vary over relations, and r over members of R. Then we define:

$$\begin{array}{lllcl} \textbf{Projection} & \pi_B R & = & \{r : R \bullet B \lhd r\} & \\ \textbf{Transformation} & f * R & = & \{r : R \bullet f\, r\} & \\ \textbf{Selection} & \sigma_P R & = & \{r : R \mid r \in P\} & = R \cap P \end{array}$$

Join. For $i = 1, 2$, let (A_i, D_i) be a schema, and R_i be a relation over (A_i, D_i) such that the domain assignments agree on the common attributes: $D_1\, a = D_2\, a$ for all a in $A_1 \cap A_2$. Then:

$$R_1 \bowtie R_2 \quad =$$

$$\{r_1 : R_1;\ r_2 : R_2 \mid \text{“function } r_1 \cup r_2 \text{ exists”} \bullet r_1 \cup r_2\}$$

Recall that functions are sets of argument-result pairs, so that for functions f and g the union $f \cup g$ is a well-defined set; it denotes a *function* again if f and g agree on their common arguments. So, since D_1 and D_2 are assumed to yield the same results on $A_1 \cap A_2$, the expression $D_1 \cup D_2$ denotes a proper function. If some D_1 and D_2 have a different domain for some common attribute a, then $R_1 \bowtie R_2$ is not defined. Note that, here, the condition “function $r_1 \cup r_2$ exists” formally means: $(\forall\, a : A_1 \cap A_2 \bullet r_1\, a = r_2\, a)$. The join is a relation over $(A_1 \cup A_2, D_1 \cup D_2)$.

17 Relations with uncertainty and ignorance

Let (A, D) be a finite schema. A *relation-with-uncertainty-and-ignorance over* (A, D), *ui-relation* for short, is a set of ct-bpa's over (A, D). We let letter R range over ui-relations, and letter m range over members (being ct-bpa's) of R. We define:

Projection	$\pi_B R$	$=$	$\{m : R \bullet B \blacktriangleleft m\}$
Transformation	$f * R$	$=$	$\{m : R \bullet f\, m\}$
Selection	$\sigma_P R$	$=$	$\{m : R \mid m{\in}P\} = R \cap P$

Join. For $i = 1, 2$, let (A_i, D_i) be a finite schema, and R_i be an ui-relation over (A_i, D_i) such that the domain assignments agree on the common attributes: $D_1\, a = D_2\, a$ for all a in $A_1 \cap A_2$. Then the *join of* R_1 *and* R_2, denoted $R_1 \bowtie R_2$, is the ui-relation over $(A_1 \cup A_2,\ D_1 \cup D_2)$ that contains for each pair (m_1, m_2) in $R_1 \times R_2$ the combination $m_1' \oplus m_2'$ provided it exists (where m_1' is the “obvious extension” of m_1 to $A_1 \cup A_2$, and similarly for m_2'):

$$R_1 \bowtie R_2 \quad = \quad \{m_1 : R_1;\ m_2 : R_2 \mid \text{“}m_1' \oplus m_2' \text{ exists”} \bullet m_1' \oplus m_2'\}$$

Within the above right-hand side, ct-bpa m_1' over schema $(A_1 \cup A_2, D_1 \cup D_2)$ is constructed out of ct-bpa m_1 over (A_1, D_1) by “extending m_1 with $*$ in the entries for all $a \notin A_1$”. Formally, m_1' maps $(P \mid Q)$ to positive x if and only if m_1 maps $(A_1 \triangleleft P \mid A_1 \triangleleft Q)$ to positive x and for all $a \in A \backslash A_1$ we have $P\, a = Q\, a = * = D_2\, a$:

$$\begin{array}{lcll} m_1'\,(P \mid Q) & = & \textbf{if} & (\forall\, a : A \backslash A_1 \bullet P\, a = D_2\, a = Q\, a) \\ & & \textbf{then} & m_1\,(A_1 \triangleleft P \mid A_1 \triangleleft Q) \\ & & \textbf{else} & 0 \end{array}$$

Similarly for the construction of m_2' out of m_2.

Note that the above definition of $R_1 \bowtie R_2$ has the same structure as the definition of the normal join, which we consider as a necessary condition in order to call our attempt successful (and the attempt by Choenni *et al.* fails in this respect). For both joins we find that only some pairs of the Cartesian product of $R_1 \times R_2$ will give rise to a row in the result, or more specifically, when “the combination” of a pair of rows exists, the pair contributes a row to the result, but when “the combination” does not exist, the rows are considered contradictory and the pair does not contribute to the result. Hence, for both joins, the size of the join $R_1 \bowtie R_2$ may be smaller than the size of the Cartesian product $R_1 \times R_2$.

Discussion

Just presenting a set of formal well-formed definitions is in itself no guarantee that some practical problem has been solved. We do have the formal definitions, but we are not yet sure that they give the outcome that one wants to have in practice. For instance, does the initial relation *SHIP* (borrowed from Choenni [1]) make sense (*two* rows about *Maria*), and does *our* relation *SHIP′* makes sense? Is there something special about keys, like *Maria*? We do not know the final answers yet. We consider our work as an attempt improve the sketch by Choenni for encoding ignorance in the relational model.

Acknowledgment. I've had many useful conversations with Henk Blanken.

. REFERENCES

[1] Sunil Choenni, Henk Ernst Blok, and Maarten Fokkinga. Extending the Relational Model with Uncertainty and Ignorance. Technical Report TR-CTIT-04-29, Centre for Telematics and Information Technology, University of Twente, The Netherlands, July 2004.

[2] Sunil Choenni, Henk Ernst Blok, and Erik Leertouwer. Handling Uncertainty and Ignorance in Databases: A Rule to Combine Dependent Data. In *Proceedings of the 11th Intl. Conf. on Database Systems for Advanced Applications (DASFAA)*, April 2006.

[3] G. Shafer. *A Mathematical Theory of Evidence*. Princeton University Press, Princeton University, 1976.

APPENDIX

18 Appendix: Interpretation of bpa

Referring to paragraph 8, we interpret the finite D as an *exhaustive* set of *distinct* propositions about some topic: D is the frame of discernment. We imagine that there are *infallible* agents that in some way or another may have obtained evidence for sets of propositions; more precisely, an agent m may have for subsets P of D “evidence of amount x supporting *just* P”, meaning that agent m is certain to degree x that a proposition in P is true. We denote this fact by:

$$m\, P \quad = \quad x$$

We assume that only the ratio between the various evidences matter, so that an amount of evidence is expressed as a number in $[0, 1]$. Moreover, in order that the agents can compare and combine their findings, we assume that each agent normalizes his evidences; thanks to the exhaustiveness of D we can do it in such a way that the sum of all evidences is one: $\Sigma\, P \bullet m\, P = 1$. The sum is well-defined thanks to finiteness of D. Since set D is exhaustive, it is impossible that there is evidence for the empty set of propositions: $m\, \varnothing = 0$. Since the propositions are *distinct*, different $p, p' \in D$ denote different propositions, so that there is no need for constraints that relate $m\,\{p\}$ to $m\,\{p'\}$. Such an agent m, then, is formally characterized by a bpa.

In practice, an agent will have evidences only for some subsets of D. In that case, we stipulate that the agent has evidence 0 for each missing $P \neq D$.

Note that an agent may have evidence x supporting just P and also another evidence supporting just a subset P' of P;

in this way the agent may differentiate between the individual propositions of D. The following *indi erence* principle is crucial (for the definition of combination in paragraph 10):

> "Evidence x supporting *just* P" does not distinguish between the individual propositions in P: it allows for arbitrarily differentiated evidences for the individual propositions and subsets of P by some other means.

In particular, $m(\{p_1, \ldots, p_n\}) = x$ does *not* imply or follow from $m(\{p_i\}) = \frac{x}{n}$ for $i = 1 \ldots n$; these two assertions will lead to distinct beliefs (defined below).

Further following the above interpretation, it is natural to say that for an agent m, the *belief* in P is the sum of all evidences supporting parts of P. In addition, the plausibility in P is the "un-belief" in the complement of P, and ignorance in P is the difference between the plausibility and belief in P.

Note that the "weakest" set of propositions is D itself, since evidence for the set D gives no information at all; in particular, evidence 1 for D (and consequently evidence 0 for every proper subset of D) signals complete ignorance.

19 Appendix: Intuition of Dempster's $\oplus$

Shafer "provides no conclusive *a priori* argument for Dempster's rule" but sees that "the rule does seem to re ect the pooling of evidence, provided..." [3, page 57]. He explains the rule by interpreting the definition for $m_1 \oplus m_2$ in a geometrical way as in Figure 2. Here we try to give "a motivated intuition" for Dempster's combination rule presented in paragraph 10.

Let D be a frame of discernment. Consider two *independent* agents characterized by m_1 and m_2, respectively. In what way can the two agents further act as one, combining their evidences? For this, we make the following observations:

(i) The infallibility and independence of the agents implies that each agent wants to combine *each* "piece of evidence" held by the other with *all* his own evidences, in such a way that it is done *proportionally to his own evidences*.

(ii) The indifference principle implies that for both agents together the evidences supporting P_1 according to agent 1 and supporting P_2 according to agent 2 together support $P_1 \cap P_2$, provided this intersection is nonempty.

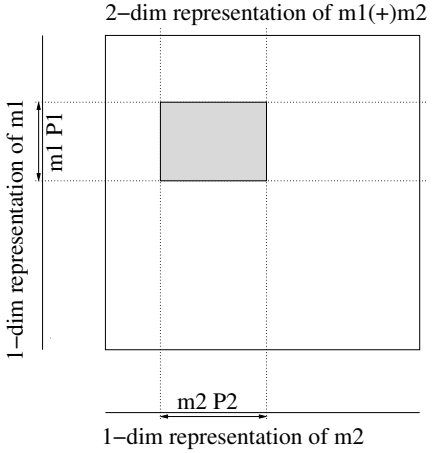
Elaborating this, we find the following:

- According to the notion of bpa, the empty set of propositions is not supported at all.
- According to (i) there exists a constant c_1 for agent 1 such that each "piece of evidence" $m_2\,P_2$ is 'combined' with each P into evidence $c_1 \times m_1\,P \times m_2\,P_2$ that, according to (ii), supports $P \cap P_2$, provided this intersection is nonempty.
- Symmetrically, according to (i) there exists a constant c_2 for agent 2 such that each "piece of evidence" $m_1\,P_1$ is 'combined' with each P into evidence $c_2 \times m_1\,P_1 \times m_2\,P$ that, according to (ii), supports $P_1 \cap P$, provided this intersection is nonempty.

For the sake of symmetry we take $c_1 = c_2$, and define $\kappa = 1/c_1 = 1/c_2$, and we have established the defining equations for $m_1 \oplus m_2$.

Regarding the value of κ, we notice that in order that the combination is a bpa, all evidences must sum up to 1, and therefore κ must be equal to the sum of all summands in the defining equations for $m_1 \oplus m_2$. It follows that when $\kappa = 0$, no subset P of D is supported by evidence of both m_1 and m_2. In that case the two agents cannot agree in accordance with the principles; the evidences that they hold are contradictory, and the combination of m_1 and m_2 simply does not exist. This concludes my intuition for the definition of $m_1 \oplus m_2$.

Figure 2 geometrically relates various items mentioned above to each other, and can be used to organize an actual computation of a bpa combination.



In the 1-dimensional representation of a bpa m, a line of length 1 is divided into segments; there is a bijection between sets P and segments, and the length of the segment for P equals the evidence $m\,P$ supporting P.

In the 2-dimensional representation of $m_1 \oplus m_2$, a square with area 1 is divided into rectangles; a rectangle represents some evidence (proportional to the size of its area) supporting some P. There is however no bijection between the subsets P and the rectangles: *di erent* rectangles may represent evidences supporting the *same* P, and *some* rectangles may not represent evidence for an P at all. In particular, the shaded rectangle has area $m_1\,P_1 \times m_2\,P_2$, and $(m_1\,P_1 \times m_2\,P_2)/\kappa$ is considered combined evidence supporting $P_1 \cap P_2$, provided this intersection is nonempty. In case $P_1 \cap P_2 = \varnothing$, then the shaded area is not used as combined evidence, and therefore the total amount of evidences in the square may sum up to less than 1.

Figure 2: Geometrical representation of the items in the definition of $m_1 \oplus m_2$

20 Appendix: Attack and defense
The following criticism on Dempster's combination $\oplus$ is well-known. Consider the following two agents in the context of $D = \{p_1, p_2, p_3\}$:

$$\begin{array}{lcl} m_1 & = & \{\{p_1\} \mapsto 0.9999,\ \{p_3\} \mapsto 0.0001\} \\ m_2 & = & \{\{p_2\} \mapsto 0.9999,\ \{p_3\} \mapsto 0.0001\} \end{array}$$

So, each agent i believes almost certainly in proposition p_i, and considers p_3 very unplausible. Yet, their combination gives full certainty to p_3, which might be considered counter-intuitive:

$$m_1 \oplus m_2 \quad = \quad \{\{p_3\} \mapsto 1.0\}$$

The defense is clear: in view of the infallibility of agents, proposition p_3 is the only one that can be true according to both agents together. Truth of an proposition with very low but positive plausibility is not inconsistent with an agents view of the world.

Some formal definitions

We give some of the missing definitions; space limitations do not permit to give all.

21 Appendix: Schema
A pair (A, D) of a set A and a function D that maps each $a \in A$ to a [finite] set, is called a [*nite*] *schema*.
Given a schema (A, D), the notion of labeled products $\Pi_A D$ and $\Pi^{\mathbb{P}}_A D$ make sense, as explained in the next paragraph.

22 Appendix: Labeled products
Members of a product $D_1 \times \cdots \times D_n$ are called *tuples* and denoted $(x_1, \ldots, x_n)$. Unfortunately, for some manipulations the concepts of product and tuple, with the ellipses "..."-notation, do not work well (for example, the "union" and "join" are not easy to express). The formulas work out far more beautiful and manipulatable when we view a tuple $(x_1, \ldots, x_n)$ as a function $x = \{1 \mapsto x_1, \ldots, n \mapsto x_n\}$, so that $x\, i = x_i$. Correspondingly, D is viewed a function $D = \{1 \mapsto D_1, \ldots, n \mapsto D_n\}$ with $D\, i = D_i$, and the role of $D_1 \times \ldots \times D_n$ is now taken over by 'the set of functions x with $x\, i \in D\, i$', denoted by $\Pi_{1..n} D$. In short, we exploit the following isomorphism ($\approx$):

$$\begin{array}{lcll} (x_1, \ldots, x_n) & \approx & \{1 \mapsto x_1,\ \ldots,\ n \mapsto x_n\} & = \ x \\ D_1 \times \cdots \times D_n & \approx & \left(\begin{array}{l}\text{the set of functions } x \\ \text{with } \forall\, i : 1..n \bullet x\, i \in D_i\end{array}\right) & = \ \Pi_{1..n} D \end{array}$$

Actually, we can now generalize a bit, and use arbitrary set A instead of $1..n$ to label the components: the set $\Pi_A D$ is a *labeled product*, and a member of $\Pi_A D$ is an *A-labeled tuple over D*:

$$\Pi_A D \quad = \quad \begin{array}{l}\text{the set of all functions } x \text{ with domain } A \\ \text{satisfying } \ \forall\, a : A \bullet x\, a \in D\, a\end{array}$$

Example. Take:

$$\begin{array}{lll} A = \{\mathit{Name}, & \mathit{Age}, & \mathit{Sex} \ \} \\ D = \{\mathit{Name} \mapsto \mathit{Text}, & \mathit{Age} \mapsto \mathit{Number}, & \mathit{Sex} \mapsto \{\text{'F'}, \text{'M'}\}\} \\ r = \{\mathit{Name} \mapsto \text{'Alice'}, & \mathit{Age} \mapsto 13, & \mathit{Sex} \mapsto \text{'F'} \ \} \\ r' = \{\mathit{Name} \mapsto \text{'Bill'}, & \mathit{Age} \mapsto 50, & \mathit{Sex} \mapsto \text{'M'} \ \} \end{array}$$

Then, r and r' are A-labeled tuples over D, that is, $r, r' \in \Pi_A D$. Imposing an order on A, say $A = (\mathit{Name}, \mathit{Age}, \mathit{Sex})$, and using the conventional tuple notation, these equations read:

$$\begin{array}{lcl} A & = & (\mathit{Name}, \quad \mathit{Age}, \quad\ \mathit{Sex}\ \ ) \\ D & = & (\mathit{Text}, \quad \mathit{Number}, \quad \{\text{'F'}, \text{'M'}\}) \\ r & = & (\text{'Alice'}, \quad 13, \quad\ \text{'F'}\ \ ) \\ r' & = & (\text{'Bill'}, \quad 50, \quad\ \text{'M'}\ \ ) \end{array}$$

Now $r, r' \in D_1 \times D_2 \times D_3$. The order on A, namely 'first *Name* then *Age* then *Sex*', is absent in the A-labeled tuples but essential in the conventional tuple notation. The conventional tuple notation *necessitates* an order on A (thus forcing some overspecification), whereas the labeled products and tuples don't do so.
(*End of example.*)

Generalizing slightly, we also define labeled products and tuples that are *set-valued* (with $P_i \subseteq D_i$):

$$\begin{array}{lcll} (P_1, \ldots, P_n) & \approx & \{1 \mapsto P_1, \ldots, n \mapsto P_n\} & = \ P \\ \mathbb{P} D_1 \times \cdots \times \mathbb{P} D_n & \approx & \left(\begin{array}{l}\text{the set of fcts } P \text{ with} \\ \forall\, i : 1 \,..\, n \bullet P\, i \subseteq D_i\end{array}\right) & = \ \Pi^{\mathbb{P}}_{1..n} D \end{array}$$

So,

$$\Pi^{\mathbb{P}}_A D \quad = \quad \begin{array}{l}\text{the set of all functions } P \text{ with domain } A \\ \text{satisfying } \ \forall\, a : A \bullet P\, a \subseteq D\, a\end{array}$$

23 Appendix: Domain restriction
Let f be a function with domain A; then the *domain restriction* of f to set B is the function $\lambda\, a : A \cap B \bullet f\, a$, for which we introduce the abbreviation: $B \lhd f$.

Let (A, D) be a finite schema, m be a ct-bpa over (A, D), and $B \subseteq A$. The *restriction of m to B*, denoted $B \blacktriangleleft m$, is the ct-bpa over (A, D) obtained from m by changing in each entry the conditions and conclusions for $A \setminus B$ into $*$, and simultaneously also, if the change has effect on the conditions, the other conclusions of the entry:

$$\begin{array}{lll} (B \blacktriangleleft m)\,(P \mid Q) = 0 & \text{if } \neg\, (\forall\, a : A \setminus B \bullet P\, a = * = Q\, a), & \\ & \text{else:} & \\ (B \blacktriangleleft m)\,(P \mid Q) = \Sigma\, P', Q' & & \\ \qquad\qquad \mid & (\forall\, a : B \bullet Q'\, a = Q\, a) \wedge & \\ & \textbf{if } Q' = Q & \\ & \quad \textbf{then } (\forall\, a : B \bullet P'\, a = P\, a) & \\ & \quad \textbf{else } P = \bar{*} & \\ \qquad\qquad \bullet & m\,(P' \mid Q') & \end{array}$$

For example, take $A = 1 \,..\, 3$ and $B = 1 \,..\, 2$ and consider, using the conventional tuple notation:

$$\begin{array}{l} \{(a_0, b_0, * \mid a, b, *) \mapsto v, \\ \ (a_0, b_0, c_0 \mid a, b, *) \mapsto w, \\ \ (a_1, b_1, c_1 \mid a, b, c\ ) \mapsto x, \\ \ (a_2, b_2, c_2 \mid a, b, c') \mapsto y\ \} \end{array}$$

This ct-bpa is mapped by $B\blacktriangleleft$ to the following ct-bpa:

$$\begin{array}{l} \{(a_0, b_0, * \mid a, b, *) \mapsto v + w, \\ \ (*, *, * \mid a, b, *) \mapsto x + y\ \} \end{array}$$

Integrity Checking for Uncertain Data

Hendrik Decker*
Instituto Tecnológico de Informática
E-46071 Valencia, Spain
hendrik@iti.upv.es

Davide Martinenghi†
Free University of Bozen/Bolzano
I-39100 Bolzano, Italy
martinenghi@inf.unibz.it

ABSTRACT

The uncertainty associated to stored information can be put in direct correspondence to the extent to which these data violate conditions expressed as semantic integrity constraints. Thus, imposing and checking such constraints provides a better control over uncertain data. We present and discuss a condition which ensures the violation tolerance of methods for integrity checking. Usually, such methods are supposed to work correctly only if all constraints are satisfied before each update. Applied to express and check conditions about uncertain data, violation tolerance means that stored data the uncertainty of which violates integrity can be tolerated while updates can be safely checked for introducing violations of constraints about uncertainty. We also discuss the soundness and completeness of violation-tolerant integrity checking and assert it for several methods.

1. INTRODUCTION

In general, the expressive power of arbitrary first-order predicate logic sentences, called "integrity constraints", is needed in databases to express "semantic" information that goes beyond their otherwise comparatively simplistic positive factual content. In particular, conditions for characterising semantically imperfect data can be expressed by such constraints. The idea is that the database is considered to be free of semantically imperfect data if and only if its given state satisfies the imposed constraints. Otherwise, if any of the constraints is violated, i.e., not satisfied, the database is taken to contain imperfect data.

In this paper, we propose to gain a better control over uncertain data by expressing semantic properties about them (e.g., conditions that qualify stored data as uncertain) by integrity constraints. If each such property is satisfied, there are no uncertain data that would violate the constraints. Conversely, violation means that the data that are responsible for violation may qualify as uncertain. By capturing uncertainty properties of data by integrity constraints, it can be hoped to achieve a double benefit: firstly, to use the expressive power of the syntax of semantic integrity constraints also for describing arbitrarily general uncertainty properties of data; secondly, to make use of established integrity checking methods in order to efficiently check data also for uncertainty. Thus, monitoring and controlling stored and incoming new data and updates with regard to their potential uncertainty can be enhanced.

In section 2, we first try to gain a better understanding of the similarities and differences, respectively, between integrity and the lack of uncertainty, on one hand, and between violated integrity and uncertainty, on the other. Then, we claim that, in spite of seemingly severe differences, it is indeed possible to represent conditions for capturing uncertainty in the form of integrity constraints, and to check them by making use of well-known methods for integrity checking. In the remainder of the paper, we substantiate this claim. In section 3, we propose a general definition of the violation tolerance of any given integrity checking method. In section 4, we present and discuss a condition which ensures violation tolerance. Also, the soundness of violation-tolerant integrity checking is asserted for several methods. In section 5, we define and assert properties of completeness of violation-tolerant integrity checking. In section 6, we observe a preponderance of attention paid to satisfaction, and counter-balance it by highlighting violation of integrity. In section 7, we address related work and conclude.

*supported by the Spanish grant TIC2003-09420-C02i

†supported by the TONES IST project financed by the European Union 6th Framework Programme under contract number FP6-7603

2. INTEGRITY AND UNCERTAINTY

In subsection 2.1, we look at similarities, in 2.2 at differences between integrity and uncertainty. It will turn out that, regardless of any differences, the representation of conditions about uncertainty by integrity constraint syntax is possible, and also their efficient evaluation by well-known integrity checking methods.

2.1 Integrity and Uncertainty: Similarities

Traditionally, integrity constraints are a means to express correctness conditions with which all stored data must com-

ply. Upon each attempt to update data that may cause a violation of integrity, the constraints imposed on the database are checked, and the update is committed only if it does not cause integrity violation. For example, in a civil registry database containing information about citizens including their marital status, entering *married(john, mary)* will violate $\forall x \forall y \forall z (married(x, y) \wedge married(x, z) \rightarrow y \neq z)$, i.e., a constraint forbidding bigamy, if *married(john, susan)* is already stored. Also, in the presence of this tuple, the constraint $\forall x \forall y\ married(x, y) \rightarrow person(x, m) \wedge person(y, m)$ for requiring an entry for each spouse of each married couple, with marital status attribute set to *m(arried)* in the *person* table of the database will signal violation upon an attempt to delete the tuple *person(susan, m)*.

Also conditions for characterising database entries as uncertain can be expressed in the syntax of integrity constraints. Uncertain data can be understood to be either imprecise or incomplete in some sense, or indefinite, or precarious (i.e., unsafe, e.g., a potential security risk), or dubious (suspicious), or potentially malign, or have less than 100% (though not 0%) probability of truth or confidence. For example, $uncertain \leftarrow person(x, null)$ qualifies each entry in the *person* table as uncertain by a namesake 0-ary predicate if the marital status of that entry is unknown, as represented by a *null* value. Or, $\forall x\ person(x, null) \rightarrow person(x, s) \vee person(x, m) \vee person(x, d) \vee person(x, w)$ says that each person with unknown marital status is either single or married or divorced or widowed. Similarly, entries of persons with birth date before the 20th century can be characterized as dubious. By amalgamating higher-order predicates into first-order terms [14], also sentences such as $uncertain \leftarrow confidence(table\text{-}entry(x, y), z) \wedge z < threshold$ may serve as constraints for capturing uncertainty about entries x in database tables y such that the confidence factor z of x is below a certain threshold (where *threshold* is some possibly parameterisable constant).

Now, it may perhaps be perceived as a bit of a stretch to also subsume data which violate conventional integrity constraints as uncertain, since they are usually considered to be certainly bad, while uncertain data are not certainly bad. But at least it seems fair to consider data which violate integrity as border cases of uncertain data. Be that as it may, we have seen that it is possible to qualify uncertain data in the syntax of integrity constraints. Hence, it should also be possible to check incoming data for violations of constraints about uncertainty (and, symmetrically, check deletions for having the effect of making existing data uncertain, such as deletions of person records which may render the marital status information of their spouse inconsistent or unknown).

Thus, the satisfaction of each constraint can be seen as a necessary condition for avoiding uncertainty about the stored information. Conversely, data that violate integrity undoubtedly convey some degree of uncertainty. Upon each attempt to update data that are potentially uncertain since they may violate integrity, the integrity constraints of the database should be checked. The update can certainly be committed if it does not cause integrity violation. Otherwise, a warning of uncertainty can be issued, and further action may be taken before or after rejecting or committing (a possibly modified version of) the update.

The plain evaluation of constraints upon each update may be unfeasibly expensive, due to their arbitrarily high query complexity. However, satisfaction or violation of constraints can be checked by well-known methods (e.g., [21, 5, 17, 22, 10, 4]) that are supposed to work in a significantly more efficient manner than plain evaluation. Yet, the price to be paid for this gain of efficiency is possibly as high as to make such methods useless in practice: in order to ensure the correctness of their output, each of them assumes that integrity be satisfied before the update, i.e., that there is no uncertainty whatsoever about the semantic correctness of stored data. In practice, this is clearly asking too much, for two reasons.

Firstly, it is unlikely in general that all semantic properties of interest are expressed as integrity constraints, so that integrity satisfaction would be equivalent with an absolute certainty about the data's correctness. This "semantic completeness" of the integrity constraints, however, is not the topic of this paper, because it is application-dependent, hard to quantify and qualify precisely, and thus cannot be expected in general. As we have already noted, satisfaction of constraints is a necessary condition for avoiding uncertainty (or whatever else is captured by the integrity conditions), but, as we have just seen, it is not a sufficient one.

The second reason why it is unlikely in practice that all integrity constraints are satisfied in each state is known to everybody with practical experience in working with large, real-life databases. For example, integrity checking is usually turned off for uploading bulk data, and is not always checked completely afterwards. This likely causes inconsistencies, constraint violations and uncertainty. Another example is the cleansing process of data warehousing, which rarely is in the position to avoid *all* integrity violations of data extracted from often inhomogeneous sources. More examples can be found in update-intensive databases. For instance, when triggers are used, their firing caused by some update may in turn cause the firing of badly controlled sequences of follow-up triggers that might not care or wait for successful integrity checks before taking action. Also, replication mechanisms in distributed databases that enter streaming data into an information store, or trade off consistency against availability, cannot be expected to always have sufficient time for exhaustive integrity checks (cf. [1]).

Now, the good news is that, as opposed to to common belief, many well-known approaches to integrity checking can indeed abandon the assumption that integrity be satisfied before each update, without sacrificing the certainty of their results and any of their efficiency. For uncertain data that violate integrity constraint, this means that integrity checking can tolerate them and leave the task of improving their integrity status to separate, possibly off-line processes. Such processes need not run, let alone be completed, before updates checked for integrity satisfaction are committed, as traditional approaches to integrity checking would require.

Given the categorical stance with which the assumption of constraint satisfaction before updates is postulated in virtually all approaches to database integrity, the preceding paragraph may read like a paradox. More such oddities are addressed in the following subsection, where we are going

to take a closer look at some differences between violated integrity and uncertainty.

2.2 Integrity and Uncertainty: Differences

In the previous section, we have proposed to see data that lack integrity as a border case of uncertain data. However, there is a significant difference. Data that lack integrity are not just uncertain, but definitely unwanted, while uncertain data may or may not have integrity. For example, the integrity constraint

$$\mathit{violated} \leftarrow \mathit{emp}(x),\ \mathit{age}(x,y),\ y < 14$$

expresses that integrity is violated by underage employment (because there is a law by which this constraint is enforced), while the formula

$$\mathit{uncertain} \leftarrow \mathit{emp}(x),\ \mathit{age}(x,y),\ y > \mathit{retirement\_age}$$

expresses that overage employment data qualify as uncertain, in the sense of dubious (since, though not forbidden, it may contradict an employer's general policy that a person beyond retirement age would remain employed). Another example is the integrity constraint

$$\mathit{violated} \leftarrow \mathit{email}(x),\ \mathit{sent}(x,y),\ \mathit{received}(x,z),\ y > z$$

which declares that integrity is violated if the sent-date of an email item is after its received-date (assuming that both x and y are normalised wrt the same time zone), and the formula

$$\mathit{uncertain} \leftarrow \mathit{email}(x, \mathit{from}(y)),\ \mathit{suspect}(y)$$

which classifies an email item x received from y as uncertain if the latter qualifies as suspect, although the message content may unexpectedly be valid and unproblematic. (It does not matter here how the *suspect* predicate is defined.)

Due to this semantic difference between data that violate integrity and data that qualify as uncertain, which has also been observed in [20], and in spite of the fact that the same syntax can be used to represent conditions for integrity and uncertainty, the use of known integrity checking methods for checking the uncertainty of data may be deemed problematic, if not unfeasible, for the following reason.

Virtually all methods for improving the efficiency of integrity checking assume that integrity be satisfied before a given update is checked for integrity preservation or violation. That way, the evaluation of the integrity status can focus on the relevant part of the data that are actually involved in, or affected by the update, while ignoring the rest, since it is known to satisfy integrity. This assumption, however, cannot be expected to hold for uncertain data, since they may very well be part of the stored data, i.e., not each stored data item can be assumed to lack uncertainty whenever some update needs to be checked for uncertainty. Hence, we are facing again essentially the same problem as addressed already in the previous section. In the following section, we are going to see that, surprisingly, the assumption can be abandoned without further ado.

3. VIOLATION TOLERANCE OF INTEGRITY CHECKING

Throughout we assume the usual terminological and notational conventions for relational and deductive databases, as known from the standard literature. With regard to approaches to database integrity, we refer to [21, 5, 17, 22, 4] and others as surveyed in [19]. However, the following definitions are independent of any concrete approach.

Different methods employ different notions of integrity satisfaction and violation, and use different criteria to determine these properties. In fact, each method, say, $\mathcal{M}$ can be identified with its criteria, which in turn can be formalised as a function that takes as input a database (i.e., a set of database facts and rules), an integrity theory (i.e., a finite set of integrity constraints), and an update (i.e., a bipartite finite set of database clauses to be inserted and deleted, resp.), and outputs upon termination one of the values in $\{\mathit{satisfied}, \mathit{violated}\}$. For a database D and an update U, let D^U denote the updated database. Thus, the soundness of a method $\mathcal{M}$ can be stated as follows.

Definition 1. **(Soundness of integrity checking)** An integrity checking method $\mathcal{M}$ is *sound* if the following holds, for each database D, each integrity theory IC that is satisfied in D, and each update U.

($\bullet$) IC is satisfied in D^U if $\mathcal{M}(D, IC, U) = \mathit{satisfied}$.

For example, the approach in [21] generates a conjunction $\Gamma(U, IC)$ of simplifications of certain instances of those constraints in IC that are possibly affected by an update U, and asserts that, under the assumption that integrity is satisfied in the old state, integrity remains satisfied in the new state (i.e., $\mathcal{M}(D, IC, U) = \mathit{satisfied}$, in terms of definition 1) if and only if $\Gamma(U, IC)$ evaluates to *true* in the updated state D^U; otherwise, integrity is violated. Under the same assumption, the approach in [22] runs an SLDNF-based resolution proof procedure, extended by some forward reasoning steps by which it is possible to delimit the search space to be traversed to those parts of the union of database and integrity constraints that are actually affected by a given update. The procedure asserts that integrity remains satisfied in D^U if the resulting search space is finitely failed; integrity is violated if the search space contains a refutation indicating inconsistency. In terms of the definition above, the $\mathcal{M}(D, IC, U)$ of [22] is the result of the traversed search space with given input from D, IC, U. In a similar manner, definition 1 can be instantiated for virtually any method of integrity checking in the literature.

Next, we formally define the notion of violation tolerance. As indicated above, the intuition of violation tolerance of an approach $\mathcal{M}$ to integrity checking is that the existence of cases of violated constraints (which may indicate the violation of a condition on the uncertainty of related data, or, more generally, the violation of any semantic property as expressed by some constraint) needs to be tolerated while $\mathcal{M}$ monitors and controls new cases of integrity violation as introduced by updates. Moreover, the cases of integrity that had been satisfied before the update are supposed to

remain satisfied afterwards. So, if the constraints express properties for preventing uncertainty, then those data that have not been uncertain should not be turned into uncertain ones by updates of other data, nor should uncertain data be introduced by any update. In order to capture this idea more formally, we first need to make precise what we mean by "cases".

Definition 2. **(Global variable, Case)**

Let W be an integrity constraint.

a) Each variable x in W that is $\forall$-quantified but not dominated by any $\exists$ quantifier (i.e., $\exists$ does not occur left of the quantifier of x in W) in the prenex normal form of W is called a *global variable of* W. Let *global(W)* denote the set of global variables in W.

b) $W\sigma$ is called a *case of* W if σ is a substitution such that $Range(\sigma) \subseteq global(W)$ and $Image(\sigma) \cap global(W) = \emptyset$.

Clearly, each variable in a constraint W represented in the standardised datalog denial form [22] is a global variable of W. Note that not any substitution whatsoever, but only substitutions of global variables of a constraint yield valid cases. Most integrity checking methods focus on certain cases of integrity constraints. These are simplified forms of original constraints that are sufficient for determining their satisfaction or violation upon given updates. In general, substitutions of non-global variables do not lead to valid simplifications.

For example, consider the integrity constraint $\exists x \forall y$ *emp(y)* $\rightarrow$ *sup(x, y)* requiring the existence of an individual x who is superior of all employees y, in the database of some enterprise. Suppose that this constraint is satisfied when the tuple $emp(e)$ is inserted to the database, recording the hiring of a new employee e. Then, checking the instance $\exists x$ *emp(e)* $\rightarrow$ *sup(x, e)* may very well evaluate to *true* while the original constraint may have become *false*, even if some new *sup* tuples are inserted along with *emp(e)*.

Further note that cases of an integrity constraint need not be ground, and that also each constraint W itself as well as each variant of W is a case of W.

With this, soundness of violation tolerance of an approach $\mathcal{M}$ to integrity checking can be defined as follows.

Definition 3. **(Violation tolerance)**
An integrity checking method $\mathcal{M}$ is *violation-tolerant* if the following holds, for each database D, each integrity theory IC, each finite set IC' of cases of constraints in IC that is satisfied in D, and each update U.

($\bullet\bullet$) IC' is satisfied in D^U if $\mathcal{M}(D, IC, U) = satisfied$.

Note that, even though there may well be an infinity of cases of constraints in IC, the finiteness requirement for IC' entails no loss of generality, as long as satisfaction of a set of constraints is supposed to be defined by requiring that each constraint be satisfied, as usual. Moreover, ($\bullet\bullet$) guarantees satisfaction of any number of cases if $\mathcal{M}$ returns *satisfied* for IC as its second argument, which is always finite.

Clearly, for violation-tolerant integrity checking, ($\bullet\bullet$) suggests to use the very same method $\mathcal{M}$ as in the traditional case, where satisfaction of all of IC in D is required. Hence, with the relaxation of definition 3 that only some cases IC' but not necessarily all constraints are satisfied before the update, no loss at all of efficiency is associated, whereas the gains are immense: with a violation-tolerant method, it will be possible to continue database operations even in the presence of (obvious or hidden, known or unknown) cases of integrity violation (which for better or worse is rather the rule than the exception in practice), while maintaining the integrity of all cases which have complied with the constraints. Whenever $\mathcal{M}$ is employed, no new cases of integrity violation will be introduced, while existing "bad" cases may disappear (by intention or even accidentally) by executing updates which have passed the integrity test of $\mathcal{M}$. So far, with the strict requirement of integrity satisfaction in the old state, not the least bit of integrity violation was tolerable. Hence, the known correctness results of virtually all approaches to database integrity would remain useless for the majority of all practical cases, and in particular for constraints about uncertainty, unless they can be shown to be violation-tolerant.

Of course, the preceding observations, as nice as they may be, would be void if no violation-tolerant method existed. Fortunately, however, most known approaches to database integrity that we have looked at so far are indeed violation-tolerant. A notable exception is the method in [10]. The following section introduces a sufficient condition by which it is fairly easy to check and assert violation tolerance.

4. A SUFFICIENT CONDITION FOR VIOLATION TOLERANCE

For a database D, an integrity theory IC, an update U and a finite set IC' of cases of constraints in IC that is satisfied in D, a straightforward special case of ($\bullet$) obviously is

($\bullet\bullet\bullet$) IC' is satisfied in D^U if $\mathcal{M}(D, IC', U) = satisfied$

For a given method $\mathcal{M}$, it is easy to see that its violation tolerance as expressed by ($\bullet\bullet$) directly follows from ($\bullet\bullet\bullet$) if the following condition is satisfied for each database D, each integrity theory IC, each finite set IC' of cases of constraints in IC that is satisfied in D, and each update U.

(#) If $\mathcal{M}(D, IC, U)$=*satisfied* then $\mathcal{M}(D, IC', U)$=*satisfied*

Hence, with regard to definition 3, we immediately have the following result, for each sound approach $\mathcal{M}$ to integrity checking.

THEOREM 1. *$\mathcal{M}$ is violation-tolerant if (#) holds.*

Proofs to verify (#) for the approaches in [21, 5, 17, 22] are fairly easy because each of them generates simplified forms of constraints, such that, roughly speaking, the truth value

of the simplified form of any case of a constraint W in IC is implied by the truth value of the simplified form of W itself, from which (#) follows. The subsequent counter-example of a method $\mathcal{M}$ which is sound according to definition 3 but not violation-tolerant shows that violation tolerance is not a matter of course. Methods such as [10, 4] also are not violation-tolerant, essentially because their optimisations of simplified constraints involves the elimination of redundancies that are independent of the database. However, when constraints are given in denial form (which is one of two common standard forms for representing integrity constraints), and when the optimisation phase of the method in [4] is applied separately to each individual constraint in IC instead of to IC as a whole, that method also is violation-tolerant.

Now, we construct a method $\mathcal{M}$ which is not violation-tolerant. Let $\mathcal{M}(D, IC, U)$ be

1. *satisfied* (resp., *violated*) if $\exists x(p(x) \wedge x \neq a)$ is satisfied (resp., violated) in D^U, whenever IC contains $\leftarrow p(x)$ and U precisely consists of inserting $p(a)$.

2. *satisfied* (resp., *violated*) if IC is satisfied (resp., violated) in D^U otherwise.

The implementation of $\mathcal{M}$ can recur on any standard method except for a special treatment of the constraint $\leftarrow p(x)$ and attempted insertions of $p(a)$. Clearly, $\mathcal{M}$ is sound in the sense of definition 1, that is, if IC is satisfied in D then $\mathcal{M}(D, IC, U) = \textit{satisfied}$ entails the satisfaction of IC in D^U. (We remark that, here, also the converse holds, i.e., that satisfaction of IC entails $\mathcal{M}(D, IC, U) = \textit{satisfied}$.) Indeed, whenever IC holds in D and point 1 applies, $\mathcal{M}(D, IC, U) = \textit{violated}$, which correctly indicates that the update violates integrity; when point 2 applies, the evaluations of IC in D^U and $\mathcal{M}(D, IC, U)$ coincide by definition, so soundness is granted by definition.

Now, let us consider a database D which contains the sole fact $p(b)$, and $IC = \{\leftarrow p(x)\}$, i.e., integrity is not satisfied in D. Further, let U be as in point 1 above, and let $W' = \leftarrow p(a)$ be a case of the constraint in IC which obviously is satisfied in D. Although $\mathcal{M}(D, IC, U) = \textit{satisfied}$, W' is satisfied in D but not in D^U, i.e., the satisfied case W' is not preserved after the update, even though the corresponding checking condition given by $\mathcal{M}$ is satisfied for IC. Thus, $\mathcal{M}$ is sound according to def. 3 but not violation-tolerant.

To conclude this example, we remark that it can also be understood as an indication that approaches to integrity that implement a special case treatment for certain cases (e.g., cases with lower or higher uncertainty) tend to be less violation-tolerant (and, in general, more error-prone) than standard methods.

5. COMPLETENESS OF VIOLATION TOLERANCE

We note that def. 1 is only a statement about the soundness of $\mathcal{M}$. Completeness can be defined by the following version of the *only-if* half of (•).

Definition 4. **(Completeness of integrity checking)** An integrity checking method $\mathcal{M}$ is *complete* if the following holds, for each database D, each integrity theory IC that is satisfied in D, and each update U.

(∘) If IC is satisfied in D^U then $\mathcal{M}(D, IC, U) = \textit{satisfied}$.

For range-restricted integrity constraints and several significant classes of logic databases including relational ones, completeness has been shown to hold for the methods in [21, 17, 22, 4] and others in the respective original papers.

In analogy to the complementarity of (•) and (∘), the question about the validity of the *only-if* half of (••) in def. 3 arises. Not surprisingly, the *only-if* half of (#) is a sufficient condition for the *only-if* half of (••), in full analogy to theorem 1. More precisely, it is easy to see that, for each each database D, each integrity theory IC, each finite set IC' of cases of constraints in IC that is satisfied in D, each update U and each integrity checking method $\mathcal{M}$ that is complete for integrity checking (i.e., that satisfies (∘)), the condition

(##) If $\mathcal{M}(D, IC', U) = \textit{satisfied}$ then $\mathcal{M}(D, IC, U) = \textit{satisfied}$

is sufficient for proving the completeness of violation tolerance of $\mathcal{M}$, i.e., to prove

(∘∘) If IC' is satisfied in D^U, then $\mathcal{M}(D, IC, U) = \textit{satisfied}$.

However, these formal results, though correct, are of limited value, since neither (##) nor (∘∘) hold for several of the most well-known methods, as the following simple example shows.

Let D consist of the fact $q(a)$ and the rule $p(x) \leftarrow q(x)$, $IC = \{\leftarrow p(x)\}$ and U be the insertion of $p(a)$. Then, the set of correct answers derivable from D would not change after committing the update, and hence, also all cases of IC that are satisfied in D remain satisfied in D^U. And indeed, approaches that notice the redundancy of the update, such as the method in [5], will simply stop right away, signalling integrity preservation for this example. However, the methods in [22] and [17], as well as several other ones will signal that U violates integrity (just the same as under the here not valid assumption that integrity was satisfied in D). Hence, neither (##) nor (∘∘) holds in general for several methods. We remark that the cited methods do not check the redundancy of the update since such additional checks require additional fact base accesses and may incur a considerable overhead in general.

Yet, we have the following completeness result.

THEOREM 2. ***(Completeness of violation tolerance)*** *Let $\mathcal{M}$ be a sound and complete method for integrity checking, D a database, IC an integrity theory, U an update and W^* a case of a constraint W in IC such that W^* is satisfied in D. If W^* is not satisfied in D^U, i.e., if W^* is violated by U, then the following holds.*
(a) $\mathcal{M}(D, \{W\}, U) = \textit{violated}$,
(b) $\mathcal{M}(D, IC, U) = \textit{violated}$.

Part (a) of theorem 2 holds because the violation of W^* in D^U entails that, *a fortiori*, W (and also IC) is violated in D^U, and hence completeness of $\mathcal{M}$ entails that $\mathcal{M}(D, W, U)$ outputs *violated*. Part (b) is a direct consequence of part (a) and the soundness of $\mathcal{M}$.

To conclude this section, we conjecture that ($\circ\circ$) holds for complete methods $\mathcal{M}$ in databases where integrity violation cannot be corroborated by any update, i.e., where no case W^* of any constraint that is already violated in D can be violated anew by any update. As an example, consider relational databases that do not allow duplicate entries of tuples in any table. Further, only conjunctive queries (definite datalog denials) be allowed as integrity constraints. For such databases and integrity theories, the constraints are all of form $\leftarrow A_1, \ldots, A_n$ $(n > 0)$, where each A_i is an atom. Such denials may only be violated by the insertion of some ground fact A such that A unifies with one of the A_i. Conversely, if any update U of ground facts leaves all cases of constraints that are satisfied in D unviolated in D^U, then no denial can be violated at all (since each violation caused by U would introduce a violated case that has been satisfied in D). This intuitive argument remains to be verified by a more formal proof. In case it turns out to be correct, a generalization of the conjecture for denials with negation lends itself for verification, but we leave that to future research.

6. CHECKING VIOLATION

Soundness and completeness of integrity checking (definitions 1 and 4, respectively), as well as the definition of violation tolerance (definition 3) and the result about soundness of violation-tolerant integrity checking (theorem 1) are conceived with regard to checking the satisfaction of constraints, but nothing is explicitly stated wrt violation. This is due to the prevalent interest of integrity checking in the preservation of integrity satisfaction, i.e., in the "good" cases, while the "bad" cases of integrity violation tend to receive less attention. However, as unwanted as they may be, they deserve a comparable amount of attention. To reflect this is the objective of this section.

For instance, the soundness and completeness of integrity checking methods $\mathcal{M}$ could in principle be defined in terms of violation in analogy to ($\bullet$) and ($\circ$) by the *if-* (soundness) and the *only-if* half (completeness) of condition ($\diamond$) below, for databases D, integrity theories IC that are satisfied in D and updates U, as follows.

($\diamond$) IC is violated in D^U iff $\mathcal{M}(D, IC, U) = \textit{violated}$.

For two-valued definitions of integrity which do not admit values other than *satisfied* and *violated* (e.g., 'unknown', 'undefined', 'overdefined' or values for graded integrity), *not satisfied* is equivalent to *violated* and vice-versa. Hence, it follows by the definitions in sections $3-5$ that conditions ($\bullet$) and the *if* half of ($\diamond$), as well as ($\circ$) and the *only-if* half of ($\diamond$), are equivalent.

Similarly, for each set IC' of cases of constraints in IC that is satisfied in D, the violation tolerance of an integrity checking method $\mathcal{M}$ could also be defined via the following condition, in analogy to $\bullet\bullet$, as follows.

($\diamond\diamond$) If IC' is violated in D^U then $\mathcal{M}(D, IC, U) = \textit{violated}$.

For two-valued definitions of integrity and sound and complete methods for integrity checking that tolerate violation, it can be easily shown that ($\bullet\bullet$) and ($\diamond\diamond$) are equivalent, since the latter is simply the contraposition of the former.

So far, the results in this section (symmetries and relationships between satisfaction and violation on one hand, and between soundness and completeness, on the other) may appear as mere formal exercises without groundbreaking insights. However, for several methods, satisfaction and violation are not symmetrical issues. For instance, for many approaches, the termination of checking either the satisfaction or the violation of integrity cannot be guaranteed, while checking violation or, respectively, satisfaction will always terminate for large, significant classes of databases, constraints and updates. This is due to the semi-decidability of satisfiability in general. But there is more to the lack of symmetry between satisfaction and violation, to be looked at in the remainder of this section, by which the preceding formal investigations can be justified.

As a side remark, we firstly point out that soundness and completeness for checking violation of integrity is interesting because most of the approaches cited in this paper, and many others, do not only signal violation in case any constraint is no longer satisfied. Rather, they precisely identify the cases of constraints that are violated by the update. Such detail of information about uncertain data is very useful for repairing violations introduced by updates, e.g., in abductive procedures such as [6].

In general, completeness of integrity checking does not come for free. For example, several approaches to database integrity generate simplified conditions that are only sufficient but not necessary. That is, the satisfaction of such conditions ensures the preservation of integrity by given updates, but if they are not satisfied, then integrity is not necessarily violated. Thus, whenever these conditions are not satisfied, such methods need to perform further checks in order to determine the integrity status of the updated database. Examples of such approaches that lack completeness of checking integrity satisfaction (which, due to the analogies between ($\bullet$), ($\circ$) and ($\diamond$), can also be understood as a lack of soundness of checking integrity violation) are [10, 16, 11]. Despite their deficiencies, such methods may provide an advantage. In many applications, updates that violate integrity are rare. Therefore, a performance advantage for checking satisfaction which is obtained by a trade-off to the disadvantage of completeness (i.e., checking violation) may be convenient for such applications. (By the way, the lack of violation tolerance of the method in [10] is independent of its lack of completeness.)

Our last argument for defending the attention paid to checking constraints for violation is directly related to uncertainty. In databases with a propensity of containing uncertain information, the usual prevalence of satisfaction over violation may easily invert to the necessity of being more attentive to updates that possibly violate integrity rather than preserve it. Then, methods such as the aforementioned ones which only provide sufficient conditions for satisfaction but not

for violation have a definite disadvantage. As an example, imagine a database in which email coming in via a central email server node is stored. Further, imagine that the integrity constraints are such that they block spam. Because of high volumes and boost frequencies of incoming mail, not each spam message can be expected to be trapped since exhaustive checks might be unaffordable. Hence, suspicious email (e.g., with harmful attachments) may be stored and integrity cannot be expected to be 100% satisfied, since there may be more spam than correct email which does not violate integrity. In such settings, a violation-tolerant approach to integrity is in demand, and particularly one that is sound for violation. Whenever such a method indicates violation, the incoming message surely qualifies as spam and thus can safely be dumped. This behaviour is preferable to that of a method which would make every effort trying to guarantee satisfaction, i.e., making sure that a message which has passed the test really is not spam.

7. CONCLUSION

We have shown that it is possible to use integrity constraints and simplification methods for their evaluation as a means of expressing and checking conditions that qualify stored data as uncertain. This is due to two reasons. Firstly, any kind of semantic information can be expressed by arbitrary first-order syntax, be it integrity, uncertainty, or whatever properties that are wanted or unwanted. In general, any property of interest that needs to be checked and monitored can be represented in that form. This advantage of the expressive power of the syntax of integrity constraints is well-known and only needed to be recalled.

The disadvantage of such arbitrary syntax is that the evaluation of formulas expressed in it may be unfeasibly expensive. Since there are known methods with which the evaluation of constraints can be considerably simplified, integrity constraints in databases (i.e., properties required to hold in each state) are usually expressed in that syntax, in spite of its complexity. In the literature, these simplifications have been believed to be sound only in case the unsimplified forms are all satisfied before an updated state is checked for preserving satisfaction. Thus, the idea to use the simplification technology for something like checking uncertainty could not arise, because it would be unreasonable to assume a complete absence of uncertainty before an update is checked for satisfying or violating constraints about uncertainty.

However, as conveyed in this paper, many well-established methods for integrity checking can be used straightforwardly for also checking other semantically complex properties, such as constraints about uncertainty, even if the database contains uncertain information that violates these constraints. So, our findings about violation tolerance embody the second reason mentioned above, why it is possibly to use known technology for integrity checking also for checking conditions about uncertainty. The result is significant, because, hitherto, all approaches to database integrity have required that no constraint be violated before any update could be checked efficiently for preserving or violating integrity.

In detail, we have qualified as violation-tolerant those approaches to database integrity that have the ability to relax the assumption of full satisfaction of integrity before updates. With such approaches, the presence of uncertain information is admissible while the invariance of satisfied cases of imposed constraints about uncertainty can be checked efficiently. We also have defined and discussed sufficient conditions for the soundness and completeness of violation-tolerant methods. Most known methods (with some noteworthy exceptions) have turned out to be violation-tolerant, while completeness seems to be achievable only under particular circumstances.

In general, we believe that the possibility of abandoning the assumption of complete satisfaction of all constraints increases the applicability of integrity checking technology significantly.

To the best of our knowledge, the allegedly fundamental assumption that integrity needs to be satisfied before an update in order to perform correct and efficient integrity checking has never been challenged by other authors. Motro's work on "integrity and uncertainty" [20] uses a similar terminology and the related terms "validity" (soundness) and "completeness". However, his work is distinctly different from ours because Motro does not deal with methods for improving the efficiency of integrity checking, which, for the work presented in this paper, is key.

Nevertheless, it cannot be overlooked that "inconsistency tolerance" has recently become an important issue. Still, most work in this area is concerned with query answering in inconsistent databases; in this sense, the notions of repair (cf., e.g., [2]) and consistent query answering are crucial and yet very different from what this paper suggests.

Several paraconsistent logic approaches have received some attention in the recent past (cf., e.g., [9]), most of which, however, abandon classical first-order logic, possibly by resorting to multivalued logic, which we have not considered in this paper. Last, we note that standard resolution-based query answering (as in [22] and [15]) is in some sense also paraconsistent, in that it does not consider irrelevant database clauses for evaluating integrity constraints (considered as queries) upon an update.

In future work, besides checking whether other, more recent methods also tolerate integrity violation, we intend to broaden the notion of tolerance such that not only approaches for integrity checking, but also methods for repairing violations and for view updating can be applied in the presence of inconsistency and/or uncertainty.

8. REFERENCES

[1] J.E. Armendáriz, H.Decker, F.D. Muñoz: Trying to Cater for Replication Consistency and Integrity of Highly Available Data. To appear in *Proc. DEXA Workshop LAAIC'06*.

[2] L. Bertossi, J. Chomicki: Query Answering in Inconsistent Databases. In J. Chomicki, R. van der Meyden, G. Saake (eds), *Logics for Emerging Applications of Databases*, 43–83. Springer, 2004.

[3] C.-L. Chang and R. Lee: Symbolic Logic and Mechanical Theorem Proving. *Computer Science Classics*. Academic Press, 1973.

[4] H. Christiansen, D. Martinenghi: On Simplification of Database Integrity Constraints. *Fundamenta Informaticae* 71(4):371–417, A. Pettorossi, M. Proietti (eds.), IOS Press, 2006. See also [18].

[5] H. Decker: Integrity Enforcement on Deductive Databases. In L. Kerschberg (ed), *Expert Database Systems*, EDS'86, 381–395. Benjamin/Cummings, 1987.

[6] H. Decker: An Extension of SLD by Abduction and Integrity Maintenance for View Updating in Deductive Databases. *Proc. JICSLP'96*, 157–169, MIT Press, 1996.

[7] H. Decker: Historical and Computational Aspects of Paraconsistency in View of the Logic Foundation of Databases. In L. Bertossi, G. Katona, K.-D. Schewe, B. Thalheim (eds), *Semantics in Databases*, 63–81. LNCS 2582, Springer, 2003.

[8] H. Decker: Total Unbiased Multivalued Paraconsistent Semantics of Database Integrity. *DEXA Workshop LAAIC'05*, 813–817. IEEE Computer Society, 2005.

[9] H. Decker, J. Villadsen, T. Waragai (eds): *Paraconsistent Computational Logic*. Proc. ICLP Workshop at FLoC'02. Dat. Skrifter vol. 95, Roskilde Univ., 2002.

[10] A. Gupta, Y. Sagiv, J. D. Ullman, and J. Widom. Constraint checking with partial information. In *Proceedings of the Thirteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, May 24-26, 1994, Minneapolis, Minnesota*, pages 45–55. ACM Press, 1994.

[11] L. Henschen, W. McCune, and S. Naqvi. Compiling constraint-checking programs from first-order formulas. In H. Gallaire, J. Minker, and J.-M. Nicolas, editors, *Advances In Database Theory, February 1-5, 1988, Los Angeles, California, USA*, volume 2, pages 145–169. Plenum Press, New York, 1984.

[12] A. Kakas, R. A. Kowalski, F. Toni: Abductive Logic Programming. *J. Logic and Computation* 2(6):719-770, 1992.

[13] R. A. Kowalski: *Logic for Problem Solving*. Elsevier, 1979.

[14] K. Bowen, R. A. Kowalski: Amalgamating Language and Metalanguage. In K. Clark, S.-A. Tärnlund (eds), *Logic Programming*, 153-172. Academic Press, 1982.

[15] R. A. Kowalski, F. Sadri, P. Soper: Integrity Checking in Deductive Databases. In *Proc. 13th VLDB*, 61-69. Morgan Kaufmann, 1987.

[16] S. Y. Lee and T. W. Ling. Further improvements on integrity constraint checking for stratifiable deductive databases. In T. M. Vijayaraman, A. P. Buchmann, C. Mohan, and N. L. Sarda, editors, *VLDB'96, Proceedings of 22th International Conference on Very Large Data Bases, September 3-6, 1996, Mumbai (Bombay), India*, pages 495–505. Morgan Kaufmann, 1996.

[17] J. W. Lloyd, L. Sonenberg, R. W. Topor: Integrity constraint checking in stratified databases. *Journal of Logic Programming* 4(4):331–343, 1987.

[18] D. Martinenghi: *Advanced Techniques for Efficient Data Integrity Checking*. PhD thesis, Roskilde University, Denmark, in *Datalogiske Skrifter* vol. 105, `http://www.ruc.dk/dat/forskning/skrifter/DS105.pdf`, 2005.

[19] D. Martinenghi, H. Christiansen, H. Decker: Integrity Checking and Maintenance in Relational and Deductive Databases, and beyond. In Z. Ma (ed), *Intelligent Databases: Technologies and Applications*, to appear. Idea Group Publishing, 2006.

[20] A. Motro. Integrity = validity + completeness. *ACM Transactions on Database Systems (TODS)* 14(4):480–502, 1989.

[21] J.-M. Nicolas: Logic for Improving Integrity Checking in Relational Data Bases. *Acta Informatica*, 18:227–253, 1982.

[22] F. Sadri, R. A. Kowalski: A Theorem-Proving Approach to Database Integrity. In J. Minker (ed), *Foundations of Deductive Databases and Logic Programming*, 313–362. Morgan Kaufmann, 1988.

[23] J. Widom, S. Ceri: *Active Database Systems*. Morgan Kaufmann, 1996.