A PROBABILISTIC RELATIONAL DATA MODEL

Daniel Barbará
Hector Garcia-Molina
Daryl Porter

CS-TR-215-89

January 1989

A Probabilistic Relational Data Model

Daniel Barbará
Hector Garcia-Molina
Daryl Porter

Department of Computer Science
Princeton University
Princeton, New Jersey 08544

ABSTRACT

It is often desirable to represent in a database entities whose properties cannot be deterministically classified. We develop a new data model that includes probabilities associated with the values of the attributes. The notion of missing probabilities is introduced for partially specified probability distributions. This new model offers a richer descriptive language allowing the database to more accurately reflect the uncertain real world. Probabilistic analogs to the basic relational operators are defined and their correctness is studied.

January 11, 1989

A Probabilistic Relational Data Model

Daniel Barbará
Hector Garcia-Molina
Daryl Porter

Department of Computer Science
Princeton University
Princeton, New Jersey 08544

1. Probabilistic Model

Entities with stochastic attributes abound in real database applications. For example, Toyota might have demographic information indicating that customers living in a certain region are likely to purchase a Corolla with probability 0.7 or a Celica with probability 0.3. An oil company might have a database of potential *Oil Sites* with probabilistic projections as to the *Type* and *Quantity* of oil at each site. In a military application, there may be entities such as *Military Bases* where hardware at the camp is uncertain or an entity set like *Spies* whose *Location* attribute is the product of guesswork. In a traditional database application, *Employees*, the qualitative attributes *Work Quality*, or *Enthusiasm* could be introduced. The stock market features *Companies* with many attributes that are notoriously non-deterministic. Finally, *Medical Records* describing, say, a patient's *Susceptibility to Heart Attack* or the *Causes of a Disease* could be stochastic in nature.

In this paper we develop the Probabilistic Data Model (PDM), an extension of the relational model [Dat, Ull] that lets one represent probabilities associated with the values of attributes. In this model, relations have deterministic keys. That is, each tuple represents a known real entity. The non-key attributes describe the properties of the entities and may be deterministic or stochastic in nature.

EXAMPLE 1.1. To illustrate, consider the following probabilistic relation:

Key	Independent Deterministic	Interdependent Stochastic	Independent Stochastic
EMPLOYEE	DEPARTMENT	QUALITY BONUS	SALES
Jon Smith	Toy	0.4 [Great Yes] 0.5 [Good Yes] 0.1 [Fair No]	0.3 [$30,000] 0.7 [$35,000]
Fred Jones	Houseware	1.0 [Good Yes]	0.5 [$20,000] 0.5 [$25,000]

The relation describes two entities, "Jon Smith" and "Fred Jones." Attribute DEPARTMENT is deterministic. e.g., it is certain that Jon Smith works in the Toy department. In this example, QUALITY and BONUS are probabilistic and jointly distributed. The interpretation is that QUALITY and BONUS are random variables whose outcome depends on the EMPLOYEE in consideration. For instance,

Prob[QUALITY = "Great" AND BONUS = "Yes" | EMPLOYEE = "Jon Smith"] = 0.4

It makes sense to consider QUALITY and BONUS jointly distributed if, as the example implies, QUALITY functionally determines BONUS. The last attribute, SALES, describes the expected sales in the coming year by the employee. It is probabilistic but independent of the other non-key attributes. For instance,

Prob[SALES = "$35,000" | EMPLOYEE = "Jon Smith"] = 0.5.

Note that in Example 1.1 the probabilities for each attribute (in a tuple) add up to 1.0, i.e., each probability distribution function is valid. This is a requirement if we are to work with probabilities. However, this requirement could be a problem in practice because it may be difficult to know the exact probabilities for all possible domain values. For example, say $30,000 and $35,000 are the two most likely sales estimates for Jon Smith, with probabilities 0.3 and 0.5 respectively. However, suppose that there are many other values that are possible but unlikely. It would be inconvenient to have to assign probabilities to all the unlikely values.

The most important feature of our model is the inclusion of *missing probabilities* to account for such incompletely specified probability distributions. For example, the Jon Smith tuple of Example 1.1 could instead be:

EXAMPLE 1.2.

EMPLOYEE	DEPARTMENT	QUALITY BONUS	SALES
Jon Smith	Toy	0.3 [Great Yes] 0.4 [Good Yes] 0.2 [Fair *] 0.1 [* *]	0.3 [$30,000] 0.5 [$35,000] 0.2 [*]

Here, 0.2 probability has not been assigned to a particular sales value. It is assumed that this missing probability is distributed over all domain values, but we make *no assumptions* as to how it is distributed. This situation could arise if, for instance, 10 people are polled to estimate Jon Smith's sales for the upcoming year. Three people have estimated sales of 30,000 (e.g., last years sales); five of 35,000 (e.g., last year plus inflation). Two people have not been reached and give rise to the 0.2 missing probability. Since the missing probability could or could not go to the value 30,000, the probability that the sales will be 30,000 next year is actually between 0.3 and 0.3 + 0.2. In this sense, the probability 0.3 associated with the sales value $30,000 is a lower bound.

The missing probability for QUALITY, BONUS is interpreted in a similar fashion. A probability of 0.1 is distributed in an undetermined way over all possible quality, bonus pairs, while 0.2 is distributed only over pairs that have a "Fair" quality component. Thus, the probability that Smith is rated as "Great" and gets a bonus is between 0.3 and 0.3 + 0.1.

We believe that missing probability is a powerful concept. It allows the model to capture uncertainty in data values as well as in the probabilities. It facilitates inserting data into a probabilistic relation, i.e., it is not necessary to have all information before some tuple can be entered. It also makes it possible to eliminate uninteresting information when displaying relations. For example, a user may only be interested in seeing values with probability greater than 0.5; the rest can be ignored. Finally, as we will see later on, missing probability arises naturally during relational operations, even when the base relations have no missing probability.

We have chosen the no assumptions interpretation for missing probabilities for two main reasons. One, we believe it is very natural and powerful. Second, manipulating such missing probabilities as relations are projected, joined, and so on, is surprisingly easy (see Section 2). Other interpretations of missing probabilities are possible, for example, one may wish to consider the missing probabilities to distributed over domain values not explicitly listed in the relation. Another possibility is to consider probabilities not listed in the relation to be *uniformly* distributed. In this case, the probability distribution is known but not enumerated in the relation. We do not want to rule out other interpretations, but due to space limitations, we will only study the no assumptions interpretation. The full system we envision, however, will allow multiple interpretations.

A central premise of our model is that keys are deterministic. This is not the only choice, but we feel that deterministic keys are very natural and lead to simple relational operators. Furthermore, it is still possible to represent stochastic entities with our model. For instance, suppose that we are not certain whether Jon Smith and Fred Jones are employees in company Acme of Example 1.1. Say the probability that Jon Smith "exists," i.e., works for Acme, is at least 0.7. To represent this, we can include an attribute COMPANY in our relation, making its distribution for Jon Smith assign 0.7 probability to "Acme," the rest missing.

Before continuing it will be useful to briefly discuss where the "probabilistic data" for our relations come from. There are actually many possibilities. One is to have users assign the probabilities according to their "confidence" or "belief" in the values [Pea]. Thus a user, after reading recommendation letters for a job candidate, may say that he is 80% confident that this is a good candidate.

A second option is to compute the probabilities from an underlying sample. For example, a conventional deterministic relation may contain the evaluations of Fred Jones made by 100 people. Say 15 people find Jones "Very Good", 85 find him "Good". The relative frequencies can lead to probabilities of 15/100 for "Very Good," 85/100 for "Good". Our extended model includes a new operator (Stochastic) that will automatically generate a probabilistic relation out of an underlying deterministic relation containing samples (see Section 6).

Probabilities can also be assigned to data according to their timeliness. For instance, suppose that at 9:00am the position of a ship is reported as X. As time progresses, the likelihood that the ship is still at X decreases, so the user may provide a time decreasing function to generate the probabilities attached to positions. (Note that missing probability is useful once again to cover the unknown locations of the ship.)

Incidentally, in this paper we will focus exclusively on discrete probability distribution functions. Although we do not discuss it here, it may be possible to extend our model to continuous probabilities, and this opens the door to other sources of probabilistic data. For example, we may know that a given radar has a normally distributed error with standard deviation of one mile. Thus, the position of an airplane will be given by a normal distribution.

The basic relational operators for probabilistic data are introduced via examples in Section 2. Section 3 contains a formalization of our model, plus a precise definition of the operators. Section 4 looks at the "correctness" of the operators on missing probability. In Section 5 we explain the differences between the PDM and previous ideas. Finally, Section 6 briefly summarizes PDM components that are not covered in detail in this paper.

2. Examples

In this section we present illustrative examples for the three basic operators in the PDM. The formal definitions are covered in the next section. Let us begin with the project operator. For the project examples we use the following relation.

PROJECT EXAMPLE Relation r_1

KEY	A B	C
k1	0.3 [a1 b1] 0.4 [a2 b1] 0.2 [a2 b2] 0.1 [* b2]	0.4 [c1] 0.6 [c2]
k2	1.0 [a2 b2]	0.5 [c2] 0.5 [c3]

EXAMPLE 2.1 PROJECT r_1 ONTO KEY, B.

This amounts to computing the marginal probability distribution for B. The result is:

KEY	B
k1	0.7 [b1] 0.3 [b2]
k2	1.0 [b2]

EXAMPLE 2.2 PROJECT r_1 ONTO KEY, A, C.

The result is:

KEY	A	C
k1	0.3 [a1] 0.6 [a2] 0.1 [*]	0.4 [c1] 0.6 [c2] 0.6 [c2]
k2	1.0 [b2]	0.5 [c2] 0.5 [c3]

Note that operationally the wildcard value * is treated as if it were simply another A attribute value. Intuitively, this appears to be correct: In the resulting tuple for k1, 0.1 probability is distributed over all possible A values because that was the amount spread over [*, b2] values in r_1. In Section 4 we will prove that treating * as another domain value is the correct way to perform projects.

The next operator is select. The select condition can refer to the attribute values, their probabilities, or both. A condition refers to a single attribute group, but several conditions can be joined together. Each individual condition is of the form

group_name: value condition, probability condition

A condition can be of two types. In a certainty condition we require that the condition hold with absolute certainty in order for a tuple to appear in the result relation. In a possibility condition we request that the condition hold at least under one interpretation of the missing probabilities. We use **V**, **P** to refer to the value and probability components of a certainty condition, and **v**, **p** to the components of a possibility condition.

For our examples we use the following relation:

SELECT EXAMPLE Relation r_2

KEY	A B C	D
k1	0.3 [a1 b1 c1] 0.2 [a2 b1 c1] 0.4 [a2 b2 c1]	0.4 [d1] 0.6 [d2]
k2	0.5 [a2 b2 c1] 0.3 [a3 b2 c2] 0.2 [* b2 *]	0.5 [d2] 0.5 [d3]

EXAMPLE 2.3 SELECT r_2 WHERE A_B_C: **V** = [a2, *, *], **P** > 0.5.

This query selects all tuples that certainly have a value of a2 in their A component with probability greater than 0.5. Tuple k1 satisfies this condition. That is, if we project out the "do not care" attributes in the group (B, C), we see that in k1, A = a2 occurs with probability 0.2 + 0.4, which is greater than 0.5. For tuple k2, we *cannot* be certain that a2 occurs with probability larger than 0.5; this may or may not be the case, depending on how the missing probability is distributed. Thus, the result is:

KEY	A B C	D
k1	0.3 [a1 b1 c1] 0.5 [a2 b1 c1] 0.2 [a2 b2 c1]	0.4 [d1] 0.6 [d2]

EXAMPLE 2.4. SELECT r_2 WHERE A_B_C: **v** = [a2, *, *], **p** > 0.5.
This is similar to example 2.3, except that we now select tuples where it is possible that A takes on a2 with probability greater than 0.5. Here, both tuples satisfy the condition and the result is the entire relation. (In tuple k2 we *cannot* rule out the possibility that *a* 2 occurs with probability greater than 0.5.)

EXAMPLE 2.5. SELECT r_2 WHERE A_B_C: **v** = [a3, *, *], **p** > 0.
This selects tuples where we cannot rule out that A might take on the value a3. The results is the k2 tuple of r_2. This example illustrates Lipski's upper bound select (see [Lip],[Lip2]).

EXAMPLE 2.6. SELECT r_2 WHERE A_B_C: **V** = [*, *, c1], **P** = 1.0.
This finds all tuples where it is absolutely certain that C = c1. The result contains only tuple k1. This query implements Lipski's lower bound select ([Lip], [Lip2]). Note that we do not require any additional operators for Lipski's upper and lower bounds. Our probabilistic model is general enough so that these particular conditions can be expressed very naturally.

Let us continue with the join operator. Consider the following two relations:

SHIPS

NAME	TYPE
Maria	0.6 [Frigate] 0.4 [Tugboat]

DESCR

TYPE	MAX-SPEED
Frigate	0.7 [20-knots] 0.3 [30-knots]
Tugboat	1.0 [15-knots]

We are 40% sure the Maria is a Tugboat. We know that Tugboats have a maximum speed of 15 knots, so it makes sense to say that we are 40% sure the Maria's maximum speed is 15 knots. Similarly, we are 60% sure she is a Frigate, and Frigates have a maximum speed of 20 knots with probability 0.7, so it is reasonable to expect the Maria to have a maximum speed of 20 knots with probability 0.6 times 0.7. This motivates the natural join operator.

EXAMPLE 2.7. JOIN SHIPS, DESCR.
There must be a common attribute, in this case TYPE. That attribute must be the key of one of the relations. The resulting relation is:

NAME	TYPE MAX-SPEED
Maria	0.6×0.7 [Frigate 20-knots]
	0.6×0.3 [Frigate 30-knots]
	0.4×1.0 [Tugboat 15-knots]

EXAMPLE 2.8.
Natural join can generate missing probabilities, even where the initial relations have none. Suppose that the DESCR relation given earlier does not contain a tuple for Tugboat. The result of the join in this case would be:

NAME	TYPE MAX-SPEED
Maria	0.6×0.7 [Frigate 20-knots]
	0.6×0.3 [Frigate 30-knots]
	0.4 [Tugboat *]

The resulting relation has 0.4 missing probability. The interpretation is quite natural. With probability 0.4 the TYPE_MAX-SPEED attribute takes on a value [Tugboat *], where "*" is some speed value. The distribution of this probability over all possible [Tugboat *] values is unknown.

3. Probabilistic Relational Algebra

In this section we formalize our probabilistic model and relational operators. Our goal is to specify what constitutes a probabilistic relation and exactly how the operators function.

We start by defining relations with a *single group* of jointly distributed attributes, and their operators. At the end of this section we then show how multi-group relations (what we will call multi-relations) can be viewed simply as a collection of single-group relations.

Definition 3.1. *Probabilistic Relation.* Let K be a set of attributes $K_1, K_2, \ldots, K_n$ and A be a set of attributes $A_1, A_2, \ldots, A_m$. The domains are given by $dom(K_i)$ and $dom(A_i)$. A probabilistic relation r (on K, A) is a function (many to one) from $dom(K_1) \times \cdots \times dom(K_n)$ into **PF**. The set **PF** is a set of probability functions for A. Each $\beta \in$ **PF** is a function (many to one) from $dom(A_1) \cup \{*\} \times \cdots \times dom(A_m) \cup \{*\}$ into the real number interval [0,1]. The symbol $* \notin dom(A_i)$ for $1 \le i \le m$, represents instances in which the value of the attribute A_i is unknown, i.e., acts as a wildcard. Every β must satisfy the following property:

$$\sum_a \beta(a) = 1 \quad \bigcirc$$

Probabilistic relations are displayed in table form in the obvious way. Note that the mappings of probability functions into 0 are not displayed in the table. For example, the table

KEY	A B
k1	0.6 [a1 b1] 0.2 [a1 b2] 0.1 [a1 *] 0.1 [* *]
k2	1.0 [a1 b2]

is equivalent to the relation $r(k1) = \beta_1$, $r(k2) = \beta_2$, where β_1 and β_2 are defined as follows:

$$\beta_1([a1 \ \ b1])=0.6$$
$$\beta_1([a1 \ \ b2])=0.2$$
$$\beta_1([a1 \ \ *])=0.1$$
$$\beta_1([* \ \ *])=0.1$$
$$\beta_2([a1 \ \ b2])=1.0$$

and all other values are mapped by β_1 and β_2 to 0. We refer to the probabilistic functions of a relation as *rows* since they correspond to tuples in the table representation. For instance, we call $r(k1)$ $(= \beta_1)$ and $r(k2)$ $(= \beta_2)$ rows.

Definition 3.2 *Cover* We say that $a' = (a_1', a_2', \ldots, a_m')$ covers $a = (a_1, a_2, \ldots, a_m)$ if for all i, $1 \le i \le m$, either $a_i' = a_i$ or $a_i' = *$. ○

3.1 Semantics of Probabilistic Relations

A probabilistic relation r on K, A defines the probabilities of certain events. One can think of the K and A attributes as random variables. If there are no missing probabilities

$$r(k)(a) = Prob[A = a \mid K = k]$$

where $k = <k_1, \ldots, k_n>$, $a = <a_1, \ldots, a_m>$, $k_i \in dom(K_i)$ and $a_i \in dom(A_i)$ Note that the expression $A = a$ above is shorthand for the event "$A_1 = a_1$ and $A_2 = a_2$ and ... and $A_m = a_m$." The event $K = k$ has similar meaning.

With missing probabilities, and a containing no wildcards,

$$r(k)(a) \le Prob[A = a \mid K = k] \le \sum_{a' \text{ covers } a} r(k)(a')$$

That is, the function $r(k)(a)$ gives a lower bound of the probability assigned to a. The upper bound is found by adding probabilities assigned to all a's that cover a.

In the rest of this section, we will continue to refer to tuples k and a as we have just defined. We will also use the shorthand $A = a$ for events as described above.

Whenever a probabilistic relation is defined, there is an *implict independence assumption.* In other words, we assume that the probabilities given in r are *independent* of any other possible database attributes. So, the existence of r in the database indirectly implies that

$$Prob[A = a \mid K = k \text{ and } B = b] = Prob[A = a \mid K = k]$$

for any set of attributes B appearing in other relations such that $B \cap K = B \cap A = \emptyset$.

3.2 Project

Before discussing the project operator, we need to make the following definition:

Definition 3.3. *Restriction of a tuple.* Let a be a tuple over a set of attributes A, and let B be a subset of these attributes. Then $a[B]$ is the sub-tuple that contains the corresponding B elements. For example, if $A = A_1, A_2, A_3$ and $B = A_2, A_3$, then $<a_1, a_2, a_3>[B] = <a_2, a_3>$. $\bigcirc$

Consider $r' = $ PROJECT r ONTO α, where r is a probabilistic relation over K, A, and α is a set of attributes $\alpha_1, \alpha_2, \ldots$. Let $A' = A \cap \alpha$, the non-key attributes that remain in r'.

With no missing probabilities, the interpretation of r' is that $r'(k)(a') = Prob[A' = a' \mid K = k]$. From basic probability theory, we know that this probability can be computed as follows:

$$Prob[A' = a' \mid K = k] = \sum_{\text{all } a \text{ s.t. } a[A'] = a'} Prob[A = a \mid K = k]$$

This equation tells us immediately how to define r', at least for the case with no missing probabilities. In Section 2 we argued that with missing probabilities, * can be treated simply as another attribute value. This is what the next definition does, but we leave the formal proof that this is correct for Section 4.

Definition 3.4. *Project.* The project of r over α is the following probabilistic relation r' (over K, A'). If $r(k)$ is not defined, then $r'(k)$ is not defined either. If $r(k)$ is defined, then $r'(k)$ is the function β defined by

$$\beta(a') = \sum_{\text{all } a \text{ s.t. } a[A'] = a'} r(k)(a) \qquad \bigcirc$$

To check that the function β given in Definition 3.4 is valid, we compute

$$S = \sum_{a'} \beta(a') = \sum_{a'} \sum_{\text{all } a \text{ s.t. } a[A'] = a'} r(k)(a)$$

Consider an arbitrary tuple $z = <z_1, \ldots, z_m>$ (where $z_i \in dom(A_i) \cup \{*\}$). It will be included in the S sum when $a' = z[A']$ and $a = z$ (and in no other case). So, since all z tuples are being considered once in the sum,

$$S = \sum_{a} r(k)(a) = 1$$

Hence, the β functions of Definition 3.4 are valid.

3.3 Select

We now turn our attention to selects. A select condition $C(k,r)$ evaluates to true if row k of r satisfies the selection. In this paper we consider a simplified syntax for these conditions. (It could be extended in several useful ways. The extensions are straightforward and are not discussed here.) We assume that conditions are of the form $\mathbf{V} = a$, $\mathbf{P}\ op\ p$ (or $\mathbf{v} = a$, $\mathbf{p}\ op\ p$), where a is a tuple with possible *'s, op is an operator like $=$, $<$, etc., and p is an real number in [0, 1]. Recall that **V**, **P** are used for certainty conditions, **v**, **p** for possibility conditions. As we saw in Example 2.3, the "do not care" fields in a have to be projected out in order to select the appropriate rows. This implicit project is included in the following definitions.

Definition 3.5. The conditions $C(k,r)$: $\mathbf{V} = a$, $\mathbf{P}\ op\ p$ when op is $>, \geq, =$; and $C(k,r)$: $\mathbf{v} = a$, $\mathbf{p}\ op\ p$ when op is $<, \leq$ are evaluated as follows. Let A' be the attributes of r where a has no *'s, and let $a' = a[A']$. Let r_j = PROJECT r OVER A'. Then $C(k,r)$ is true if $r_j(k)(a')\ op\ p$ is true. ○

Definition 3.6. The conditions $C(k,r)$: $\mathbf{v} = a$, $\mathbf{p}\ op\ p$ when op is $>, \geq, =$; and $C(k,r)$: $\mathbf{V} = a$, $\mathbf{P}\ op\ p$ when op is $<, \leq$ are evaluated as follows. Again, let A' be the attributes of r where a has no *'s, and let $a' = a[A']$. Let r_j = PROJECT r OVER A'. Then $C(k,r)$ is true if $\sum_{\text{all } x \text{ s.t. } x \text{ covers } a'} r_j(k)(x)\ \ op\ p$ is true. ○

Definition 3.7 *Select.* Let r' = **SELECT** r **WHERE** C be the target relation obtained by the operation select over the probabilistic relation r defined on K, A. Probabilistic relation r' is also defined on K, A. If $r(k)$ is undefined or if $C(k,r)$ is false, then $r'(k)$ is undefined. Otherwise, $r'(k) = r(k)$. ○

3.4 Join

Definition 3.8. *Natural Join.* Consider two relations, r on K, A, and s on A, B. Let r' be the natural join of r and s. Relation r' is over K, AB. If $r(k)$ is not defined, then $r'(k)$ is not defined. Otherwise the function is defined as follows:

```
if $s(a)$ is defined then
        $r'(k)(ab) = r(k)(a) \times s(a)(b)$
else if $s(a)$ is not defined then
        $r'(k)(a^*) = r(k)(a)$
        $r'(k)(ab) = 0$ for all $b$ other than *.
```

Note that if $a = <a_1,...,a_m>$ and $b = <b_1,...,b_o>$, then ab is the tuple $<a_1,...,a_m,b_1, \dots ,b_o>$. The tuple a^* represents $<a_1,...,a_m,*,...,*>$. ○

If no missing probabilities are present, this definition is justified by basic probability theory. The value $r'(k)(ab)$ should be equal to $Prob[AB = ab \mid K = k]$. Using Bayes' theorem, we get:

$$Prob[AB = ab \mid K = k] = Prob[B = b \mid K = k \text{ and } A = a]\, Prob[A = a \mid K = k]$$

Because of our independence assumption, the probability of $B = b$ only depends on A, not on K. Thus we can simplify:

$$Prob[AB = ab \mid K = k] = Prob[B = b \mid A = a]\, Prob[A = a \mid K = k]$$

This is the formula that is used in Definition 3.8. We discuss the correctness of Definition 3.8 under missing probabilities in Section 4.

It is straightforward to check that the probability functions given in Definition 3.8 are valid. We omit the proof here.

3.5 Multi-Relations

Definition 3.9. *Multi-Relation.* A table with multiple groups is represented by a multi-relation. A multi-relation R on attributes $K, A_1, ..., A_q$ is a tuple $<r_1,...,r_q>$ where each r_i is a probabilistic relation on K, A_i. There is one condition (we call it the "same keys" constraint) that must be satisfied by the relations: if $r_i(k)$ is defined, then for all j, $1 \le j \le q$, $r_j(k)$ must also be defined. ○

A multi-relation is displayed in table form by concatenating the individual relations. However, the key attribute column is only displayed once, on the left. The condition in Definition 3.9 ensures that in every row of the table, the probability function will be defined.

As an example, consider mutli-relation r_1 used for the project examples of Section 2. (It was called a relation then since we had not introduced multi-relations.) It is composed of a relation on K, AB and another on K, C. Attributes A and B are jointly distributed, but because of our independence assumption (Section 3.1), C and A, as well as C and B, are independent.

Relational operators on such a multi-relation can now be defined in terms of the operators on the component relations. Let $R = <r_1, \dots ,r_q>$ be a multi-relation on $K, A_1, ..., A_q$.

• *Project.*

PROJECT R ONTO $\alpha = \langle r_1', ..., r_q' \rangle$ where $r_i' =$ PROJECT r_i ONTO α.

• *Select.*

The select condition c for R has $q+1$ components, $c_0, c_1, ..., c_q$. The condition c_i is intended to select within relation r_i. Condition c_0 selects keys. If no selection is desired for r_i, then c_i is set to "true."

SELECT R WHERE $c_0, c_1, ..., c_q = \langle r_1', ..., r_q' \rangle$ where $r_i' =$ SELECT r_i WHERE d. Condition $d(k, r_i)$ is true if and only if $c_0(k)$ and $c_1(k, r_1)$ and ... and $c_q(k, r_q)$. (Note that implements a logical AND of the conditions. More general multi-relation selects could be defined in a similar fashion.)

• *Natural Join.*

Let $S = \langle s_1, \ldots, s_t \rangle$ be a second multi-relation on A_i, B_1, ..., B_t.

JOIN R, $S = \langle r_1', \ldots, r_{i-1}', r_{i+1}', \ldots, r_q', s_1', \ldots, s_t' \rangle$, where $r_j' = r_j$ $(j \neq i)$, and $s_j' =$ JOIN r_i, s_j.

For each of the above definitions, it is relatively simple to show that result is a valid multi-relation, i.e., that the same-keys constraint holds.

Finally, in Section 1 we introduced deterministic attributes. Deterministic attributes are just like probabilistic ones except that the probability functions always assign probability 1.0 to a single value. When such attributes are displayed in table form, the probabilities are stripped away.

4. Lossy Operations

The project and join operators generate new probability distribution functions in the result. In Section 3 we showed that these new functions were correct when no missing probabilities were involved. In this section we address the issue of whether the new functions are still meaningful when missing probabilities are involved.

We start by showing that in certain cases join yields functions that are "unexpected." Consider the following join of relations r and s:

EXAMPLE 4.1.

r

K	A
k	0.4 [a1] 0.6 [*]

s

A	B
a1	0.7 [b1] 0.3 [b2]

Join *r*, *s*

K	A B
k	0.28 [a1 b1] 0.12 [a1 b2] 0.6 [* *]

In the resulting join, the probability of tuple [a1 b1] ranges from 0.28 to 0.88. (This last value is obtained when all the missing probability of 0.6 is assumed to fall on [a1 b1].) However, if we look at the original relations, there is no way we can obtain a probability of 0.88 for [a1 b1]. Even if in *r* we assume all of the 0.6 missing probability is given to [a1], when we join we obtain at most 1.0×0.7 for [a1 b1]. Thus, the result we have shown does not accurately represent the join.

Another way of looking at this problem is that the missing probability of 0.6 [* *] in the result is not correct. It is not the case that we know nothing about the distribution of this 0.6 probability over all possible A, B tuples. In particular, if the first component happens to be a1, then the missing probability should be assigned according to the a1 distribution shown in *s*. Hence, the most of the 0.6 probability that could be assigned to [a1 b1] is 0.6×0.7. However, all of the 0.6 can be assigned to say [a2 b1]. Our PDM is not rich enough to capture this subtlety in the missing probabilities.

It would be possible to extend the model for missing probabilities to represent "partially unknown" distributions, but this would lead to a cumbersome model and language. Instead we have stayed with the simple interpretation of missing probabilities. There are several good reasons for this. First, the problem illustrated in this example only occurs with joins when the first relation has missing probabilities. It never occurs with projects or selects. (We will formally prove this in this section.) So a system implementing our model could simply refuse to perform joins that have the problem.

Second, the "problem" is not necessarily bad. The data in the resulting relation can still be useful. It provides a simplified but "diffused" view of the join. The probability ranges implied by the result are larger than should be, but the minimum value is still correct. (We will prove this later.) For example, as we indicated above, the range for [a1 b1] is given as 0.28 to 0.88, but it should be 0.28 to 0.7. The range for [a1 b2] is shown as 0.12 to 0.72, but should be 0.12 to 0.3.

In a sense, the problem illustrated here causes "information loss." The resulting relation has less information about the probabilities involved that what could be inferred form the original relations. If a user does not want to loose information, then he or she should not perform joins when the first relation has missing probabilities. An analogy can be drawn to lossy joins in conventional relational algebra [Ull]. There too information is lost in certain joins. The loss is different from what we have illustrated, but the point is the same: If one does not wish to have less information in the resulting relation, then the operation in question should not be performed. But it is up to one to decide in each particular case.

In the rest of this section we formalize the notion of probabilistic information loss. We only consider single attribute relations; the generalization to multi-relations is straightforward.

Definition 4.1 *Potential Set.* Given a tuple with key k in a probabilistic relation r, the potential set of $r(k)$, or simply $PSET(r(k))$, is defined as the set of all functions (without missing probabilities) that can be obtained from $r(k)$, each one by assuming a particular distribution of the missing probabilities. More formally, $PSET(r(k))$ is the set of all valid probabilistic functions β such that

(i) for all tuples a with *'s, $\beta(a) = 0$;

(ii) for all tuples a with no *'s, $r(k)(a) \leq \beta(a) \leq \sum_{a' \text{ covers } a} r(k)(a')$ ○

To illustrate, consider relation r of Example 4.1. The function that maps [a1] to a probability of 0.5 and [a2] to 0.5 is in $PSET(r(k))$. No function that maps [a1] to 0.2 could be in $PSET(r(k))$.

It is clear from Definition 4.1 that for any row with key k in which there are no missing probabilities $PSET(r(k)) = \{r(k)\}$. By extension, the PSET of a relation r is simply

$$PSET(r) = \{r' \mid r'(k) \in PSET(r(k)) \text{ for all } k\}$$

Definition 4.2 *Probabilistically Lossless Operations.* Let $\oplus$ be an operation over relations $r_1, r_2, \ldots, r_n$ such that $r = \oplus(r_1, r_2, \ldots, r_n)$. The operation is probabilistically lossless if for all k such that $r(k)$ is defined either

(i) $r(k) = r_i(k)$ for some i, $1 \leq i \leq n$; or

(ii) $PSET(\, r(k)\,) = \bigcup_{\text{all } r_i' \in PSET(r_i)} PSET[\oplus(r_1', \ldots, r_n')(k)]$ ○

An operation that is not lossless is called lossy. Definition 4.2 states that for an operation to be lossless the probability function assigned to every tuple in the result must be meaningful. This occurs if the function was in the original set of functions, or if it has a potential set that can be reproduced by the potential sets of tuples in the operand relations. In part (ii) of the definition, notice that even though r_1', ..., r_n' have no missing probabilities, $\oplus(r_1', \ldots, r_n')(k)$ may have missing probabilities (in particular if $\oplus$ is a join, see Definition 3.8). This makes necessary the application of the $PSET$ function to $\oplus(r_1', \ldots, r_n')(k)$.

We now show that project and select always compute correct probability functions.

Lemma 4.1 Project is a probabilistically lossless operation.

Proof: See Appendix.

Lemma 4.2 Select is a probabilistically lossless operation.

Proof: See Appendix.

As was shown in Example 4.1, join can be lossy in some cases. To determine in what cases it is not, we introduce the following definition:

Definition 4.1. A relation r has property *NMP* (no missing probability) if no row in r is a function with missing probabilities.

Lemma 4.3 If a relation r has property *NMP*, then JOIN r, s is probabilistically lossless, for any relation s.

Proof: See Appendix.

In the cases where $r' =$ JOIN r, s is lossy, the result is still of partial use because the lower bounds for probabilities are correct.

Lemma 4.4. Let $r' =$ JOIN r, s, where r is over K, A, and s is over A, B. Say $r'(k)(ab) = p$, where ab has no *'s. Then $Prob[AB = ab \mid K = k]$ is indeed greater than or equal to p.

Proof: See Appendix.

In closing this section, notice that we have focused on possible information loss at the level of probability distribution functions. It is also possible to study information loss at the relation level. For instance, suppose we compute $t =$ JOIN r, s. Is it possible to recover the original relations r and s from t? If it were, then the join would be lossless in the conventional sense. Clearly, if the join is probabilistically lossy, then it is impossible to recover the original relations. However, when the join is probabilistically lossless, then it is possible to recover r and s, i.e., the join is also lossless in the conventional sense. For instance, r can be recovered by projecting t over the attributes of r. To recover s we need a new operator that extracts conditional probabilities. Due to space limitations we cannot cover this new operator nor the proof that a probabilistically lossless join is also conventionally lossless.

5. Related Work

The PDM we are proposing here was inspired by many of the prior ideas relating to uncertain information [Cav, Gel, Imi, Imi2, Imi3, Lip, Lip2, Liu, Men, Mor, Raj, Rei, Vas]. However, our model is distinguished by two critical aspects: (a) the use of probabilities to describe uncertainty, as opposed to other mechanisms such as fuzzy sets, possibilities, sets of alternatives, null values, and so on; and (b) the introduction of missing probabilities.

To understand aspect (a), let us briefly compare the PDM to Fuzzy Relations [Raj]. Both models may look similar on the surface, but there are important differences. A fuzzy relation is viewed as a fuzzy set, with tuples that may or may not be in a relation r. A *Possibility Distribution Function*, μ_r is defined to map each tuple t to a value over $[0,1]$. Additionally, the domain of an attribute can also be expressed as a fuzzy set, defining a possibility distribution function for it.

EXAMPLE 5.1 Consider the following fuzzy relation.

EMPLOYEE	SALES	μ
Jon Smith	\$30,000 / 0.3 \$40,000 / 0.6 \$50,000 / 0.4	0.70
Jim Jones	\$10,000 / 0.8	0.8

The numbers next to the sales figures indicate the values of the possibility function for the sales attribute. For instance, Possibility(Sales of Jon Smith = \$30,000) = 0.3. The value of $\mu = 0.7$, associated with the tuple of Jon Smith, indicates that the possibility that this tuple belongs to the relation is 0.7. Note that the values of a possibility function do not have to sum to 1.0.

Possibilities and probabilities are obviously different concepts. One major difference is that it is hard to represent with possibilities attributes like SALES that have a single outcome. In other words, the example above implies that SALES for Jon Smith can take on several values at once (i.e., it is a fuzzy set). This is not the case for SALES (and for a large number of common database attributes, we think). A probabilistic function, on the other hand, captures this exactly: SALES can take on a single value out of the listed ones, and each probability gives the likelihood that it happens. In addition to being able to represent single outcome attributes, the PDM can still model fuzzy sets if necessary. (See the example of the employee working at Acme in Section 1.) Although possibilities and fuzzy sets have been carefully studied, probabilities are much more widely known (e.g., by every engineer and scientist) and have strong intuitive appeal. Thus, they seem like a good basis for building a *general purpose* incomplete information model.

Probabilities have been used in one database model we know of [Cav]. In it, each tuple is assigned a probability, representing the likelihood that the tuple appears in the relation.
EXAMPLE 5.2. The following is a relation in the [Cav] model. $A,B \in \{0, 1\}$

A	B	p(Tuple = .)
0	0	0.35
0	1	0.25
1	0	0.01
1	1	0.39

For instance, the probability that tuple $<0,0>$ exists is 0.35. The PDM we propose provides a richer structure than this, allowing probabilities to be assigned to individual attributes. As described in the introduction, we think deterministic keys (which do not exist in [Cav]) are very useful, especially in applications where the entities in question clearly exist (e.g., a patient in a hospital, a military base). Finally, and most importantly, the PDM provides for missing probabilities. As we have argued, this provides for substantial flexibility in modeling real applications.

In closing this section, we briefly discuss two research areas that are also related to our work. One is statistical databases. For example, in [Won], Wong views queries as statistical experiments in which information was incomplete because, for example, it was old. His method centered around computing the query response as those tuples which minimized the two types of statistical errors. In another paper [Gho], Ghosh discusses statistical relational tables. Although there are similarities (see for instance the operator Stochastic in Section 6), our main interest in this paper is on uncertain, not statistical information.

Our work is also related to the current efforts with non-first normal form relations [Dad, Rot, Rot2]. As a matter of fact, our probabilistic relations are simply nested relations with depth two. Each attribute in the top level relation can be a relation that gives the probability distribution function for the attribute. Our focus in this paper will be on the probabilistic operators and not on the non-first normal form issues.

6. Conclusions

We have presented a probabilistic relational data model where tuples have deterministic keys and both probabilistic and deterministic attribute groups. We believe that the major strengths of our model are its generality and naturalness. Most existing incomplete information models can be encompassed by ours. For example, a null value can be represented by a 1.0 [*] distribution. A list of alternate values [Lip] can be represented by a distribution that assigns each value an equal probability. As discussed in Section 5, fuzzy sets can also be represented in our model. At the same time, these other models cannot represent the full range of probabilities we can.

Another important strength, in our opinion, is the concept of missing probability. Not only does this concept make it easier to input probabilistic data, but it also makes joins feasible when join attribute values are missing (Example 2.8).

In this paper we have focused on the three basic operators project, select, and join. However, the model can naturally support many other useful operators. Here we provide a brief summary. (Some of these are described in more detail in [Gar].)

Conventional Operators Applied to Probabilistic Relations. The basic set operators INTERSECT, UNION, MINUS work as in a conventional system. The only complication is that UNION may produce a relation containing two rows with the same key but different functions. The COMBINE operator, discussed below, can be used to produce a single probability function. In addition, INSERT, DELETE, and UPDATE operators can be introduced to modify existing relations.

Operators for Multi-Relations. Our system provides operators for manipulating multi-relations. For example, GLUE can produce a multi-relation out of a collection of simple relations. CUT is used to break a multi-relation into its components. The GROUP operator combines two groups into a single one, computing the joint probability distribution.

New Operators. The following operators do not have a counterpart in relational databases:

(a) *STOCHASTIC* : This operator takes as input a deterministic relation containing samples and a probabilistic schema, returning a probabilistic relation based on that schema. (The probabilistic schema describes what attributes comprise the KEY, and which of the dependent attributes are jointly distributed or independent.) The probabilities are computed as relative frequencies of the samples.

(b) *DISCRETE*. The DISCRETE operator transforms a probabilistic relation into a deterministic one. Each probability distribution is replaced by the expected value of the attributes. If the domains are not numeric, a conversion function must be provided. For example, if student "Fred" is expected to get "A" with probability 0.7 and "B" with probability 0.3, then the DISCRETE relation would show "Fred" with a grade of $0.7 \times 4 + 0.3 \times 3 = 3.7$, assuming "A" is mapped to a numeric grade of 4 and "B" to 3.

(c) *COMBINE* : This operator can be used to combine two probability distributions. For example, suppose that two independent observers keep track of the distance of objects to a given site. Each observer generates a probabilistic relation for his data. If two such relations are available, it may be desirable to perform a union to combine the information. In the union there may be two distributions for the same object. The two distributions can be combined into a single one, possibly giving a different weight to each observer.

(d) *CONDITIONAL*. Say we have a relation r with key K and attributes AB. CONDITIONAL can convert r into r' with key KA and attribute B. The value $r'(ka)(b)$ will then equal $Prob\,[B = b \mid K = k \text{ and } A = a]$.

(e) *ε-JOIN* : This operator acts like the natural join, except the attributes for which the tables are joined are both probabilistic. The corresponding tuples are joined if the two distributions are less than ε apart according to a predetermined distance function.

(f) *ε-SELECT* : This operator selects tuples for which the distribution function is less than epsilon apart from a given distribution, again according to a pre-defined distance function.

Finally, it is possible to allow other interpretations for missing probabilities, or to predefine common probability distribution functions. For example, the notation 0.4 U[a2 ... a5] can be used in a relation instead of assigning 0.1 probability to each of a2, a3, a4, a5. Wildcards can also be used. For instance, 0.3 U[*] distributes 0.3 probability among all possible domain values. This is simply shorthand notation: the relational operators should manipulate a relation with uniform distributions simply as if they had been expanded.

7. References

[Cav] Cavallo, R. & Pittarelli, M., "The Theory of Probabilistic Databases," *Proceedings of the 13th Conference on Very Large Databases*, 1987.

[Dad] Dadam, P., Kuespert, K. et al, "A DBMS Prototype to Support Non-First Normal Form Relations: An Integrated View on Flat Tables & Hierarchies", *SIGMOD Proceedings*, 1986.

[Dat] Date, C., *An Introduction to Database Systems Vol. 1,* Addison-Wesley, 4th Edition, 1986.

[Gar] Garcia-Molina, H & Porter, D., "Supporting Probabilistic Data in a Relational System," Technical Report TR-147, Department of Computer Science, Princeton University, February 1988.

[Gel] Gelenbe, E. & Hebrail, G., "A Probability Model of Uncertainty in Databases," *Proceedings of the International Conference on Data Engineering*, Feb. 1986.

[Gho] Ghosh, S., "Statistical Relational Tables for Statistical Database Management," *IEEE Transactions on Software Engineering*, Dec. 1986.

[Imi] Imielinski, T., "Query Processing in Deductive Databases with Incomplete Information," Rutgers Technical Report DCS-TR-177, Mar. 1986.

[Imi2] Imielinski, T., "Automated Deduction in Databases with Incomplete Information," Rutgers Technical Report DCS-TR-181, Mar. 1986.

[Imi3] Imielinski, T. & Lipski, W., "Incomplete Information in Relational Databases," *Journal of the ACM*, Vol. 31, No. 4, Oct. 1984.

[Lip] Lipski, W., 'On Semantic Issues Connected with Incomplete Information Databases," *ACM Transactions on Database Systems*, Vol. 4, No. 3, Sep. 1979.

[Lip2] Lipski, W., "On the Logic of Incomplete Information," *Proceedings of the 6th International Symposium of Mathematical Foundations of Computer Science*, September 1977.

[Liu] Liu, Ken-Chih & Sunderraman, R., "On Representing Indefinite & Maybe Information in Relational Databases," *Proceedings of the 4th International Conference on Data Engineering*, February 1988.

[Men] Mendelson, H. & Saharia, A., "Incomplete Information Costs & Database Design," *ACM Transactions on Database Systems*, Vol. 11, June 1986.

[Mey] Meyer, P. *Introductory Probability & Statistical Applications,* Addison-Wesley, 2nd Edition, 1970.

[Mor] Morrissey, J. & van Rijsbergen, C., "A Formal Treatment of Missing & Imprecise Information," *Proceedings of SIGIR*, 1987.

[Pea] Pearl, Judea, "Fusion, Propagation, and Structuring in Belief Networks," *Artificial Intelligence* 29, 1986, pp 241-288.

[Raj] Raju, K.V.S.V.N., & Majumdar A., "Fuzzy Functional Dependencies and Lossless Join Decomposition of Fuzzy Relational Database Systems," *ACM Transactions on Database Systems*, Vol. 13, No. 2, June 1988.

[Rei] Reiter, R., "A Sound & Sometimes Complete Query Evaluation Algorithm for Relational Databases with Null Values," *Journal of the ACM*, Vol. 33, No. 2, Apr. 1986.

[Rot] Roth, M., Korth, H., & Silberschatz, A., "Extended Algebra & Calculus for Non-1NF Relational Databases," University of Texas at Austin Technical Report TR-84-36, Jan. 1985.

[Rot2] Roth, M., Korth, H., & Silberschatz, A., "Null Values in Non-1NF Relational Databases," University of Texas at Austin Technical Report TR-85-32, Dec. 1985.

[Ull] Ullman, J. D., *Principles of Database and Knowledge-Base Systems*, Computer Science Press, 1988.

[Vas] Vassilou, Y., "Functional Dependencies & Incomplete Information," *Proceedings of the 6th International Conference on Very Large Databases*, Montreal, Oct. 1980.

[Won] Wong, E., "A Statistical Approach to Incomplete Information in Database Systems," *ACM Transactions on Database Systems*, Vol. 7, No. 3, Sep. 1982.

Appendix

This appendix contains the proofs for the Lemmas of Section 4.

Lemma 4.1 Project is a probabilistically lossless operation.

Proof: Let r_1 be a relation over K, A_1. Let r = PROJECT r_1 OVER α and say $A = A_1 \cap \alpha$, the non-key attributes that remain in r. Let k be any key such that $r(k)$ is defined. To show that the set equality of Definition 4.2 (part (ii)) holds, we proceed in two parts:

(1) Take any $\beta \in PSET(r(k))$. We want to show that this β is also in some $PSET$[(PROJECT r_1' OVER α)(k)] where r_1' is in $PSET(r_1)$. Since r_1' has no missing probabilities, it is enough to find β in a project of some r_1'. Say $\beta(a) = p_1 + p_2$ for some tuple a over A. This can only happen if $r(k)(a) = p_1$ and $\sum_{x \in X} r(k)(x) \geq p_1 + p_2$, where X is the set of all tuples (over A) that cover a. Since r was computed through a project of r_1, it must be the case that $\sum_{y \in Y_1} r_1(k)(y) = p_1$ where $Y_1 = \{y \text{ over } A_1 \mid y[A] = a\}$. Similarly, it must be the case that $\sum_{y \in Y_2} r_1(k)(y) \geq p_1 + p_2$, where $Y_2 = \{y \text{ over } A_1 \mid y[A] = x \text{ for some } x \in X\}$. We can now create a particular r_1' in $PSET(r_1)$ that assigns a combined probability $p_1 + p_2$ to the tuples that are covered by tuples in Y_1 and Y_2. That is, we chose r_1' so that $\sum_{z \in Z} r_1'(k)(z) = p_1 + p_2$, where $Z = \{z \text{ over } A_1 \mid y \text{ covers } z \text{ for some } y \in Y_1 \text{ or } y \in Y_2\}$. Note that for each tuple $z \in Z$, $z[A] = a$. Hence, when r_1' is projected, all the tuples in Z collapse into tuple a and this tuple receives exactly $p_1 + p_2$ probability. Thus we have shown that the probability function β in $PSET(r(k))$ can be found by projecting a relation r_1' in $PSET(r_1)$.

(2) Take a probability function β obtained with some key k from the project of r_1', a relation in $PSET(r_1)$. We now want to show that this same function exists in $PSET(r(k))$. To do this, we simply reverse the steps of part (1). If $\beta(a) = p_1 + p_2$, then $\sum_{z \in Z} r_1'(k)(z) = p_1 + p_2$, where Z contains tuples z such that $z[A] = a$. Continuing, the sum of $r_1(k)$ over tuples in $Y_1 = Z$ must equal p_1. The sum of $r_1(k)$ over Y_2 tuples must be greater than $p_1 + p_2$, where Y_2 contains all tuples covered by one in Z. This implies that $r(k)(a) = p_1$ and the sum of $r(k)$ over X tuples is greater than $p_1 + p_2$, where X is the set of tuples Y_2 projects to. Since all the tuples in X cover a, we can find a function in $PSET(r)$ that maps a to probability $p_1 + p_2$. This is the function we were looking for. ○

Lemma 4.2 Select is a probabilistically lossless operation.

Proof: A select does not modify the probability functions. Any $r(k)$ that exists in SELECT r_1 WHERE C must appear in r_1. Thus, select is lossless by case (i) of Definition 4.2. ○

Lemma 4.3 If a relation r has property *NMP*, then JOIN r, s is probabilistically lossless, for any relation s.

Proof: Let r_1 be a relation over K, A, and r_2 a relation over A, B. Let r = JOIN r_1, r_2. Let k be a key such that $r(k)$ is defined.

(1) Take any $\beta \in PSET(r(k))$, and consider a tuple ab over AB with no wildcards. As a first sub-case, assume that $r_2(a)$ is defined. Then $\beta(ab)$ must equal $p_1(p_2 + p_3)$, where $p_1 = r_1(k)(a)$, $p_2 = r_2(a)(b)$, and $\sum_{y \in Y} r_1(k)(a) \times r_2(a)(y) \geq p_1(p_2 + p_3)$, where Y is the set of all B tuples that cover b. Now, in $PSET(r_1)$ there is only one relation, $r_1' = r_1$ (r_1 has NMP property). From $PSET(r_2)$ we take the relation r_2' that assigns to row a, value b probability $p_2 + p_3$. Let t = JOIN r_1', r_2'. Clearly, $t(k)(ab) = p_1(p_2+p_3)$. This is the same function β that we started with.

For the second sub-case, assume that $r_2(a)$ is undefined. Then $\beta(ab) = p \leq r(k)(a^*) = r_1(k)(a)$. We again take $r_1' = r_1$ and let r_2' be any member of $PSET(r_2)$. Let t = JOIN r_1', r_2'. Since $r_2'(a)$ is undefined, $t(k)(a^*) \geq p$. When we take the $PSET$ of $t(k)$ we assign probability p to tuple ab, obtaining the same function as β.

(2) Take any relation $r_2' \in PSET(r_2)$. Again, $PSET(r_1) = r_1 = r_1'$. Let t = JOIN r_1', r_2'. For the first sub-case, say $t(k)$ has missing probabilities. Then there is some a such that $r_2'(a)$ is undefined and $r_1(k)(a) \neq 0$. By definition, $t(k)(a^*) = r_1(k)(a)$. When we perform $PSET(t(k))$ we can assign to a particular ab tuple some value between 0 and $r_1(k)(a)$, say p. Now we show that this same assignment can be obtained through $PSET(r(k))$. If $r_2'(a)$ is not defined, then $r_2(a)$ is also undefined. Hence, $r(k)(a^*) = r_1(k)(a) \geq p$. When we do a $PSET$ of $r(k)$, we can assign probability p to tuple ab, arriving at the same function we had.

For the second sub-case, say $t(k)$ has NMP or $r_2'(a)$ is defined. Say $t(k)(ab) = p_1(p_2+p_3)$. Thus, $r_1(k)(a) = p_1$, $r_2(a)(b) = p_2$, and $\sum_{y \in Y} r_1(k)(a) \times r_2(a)(y) \geq p_1(p_2 + p_3)$, where Y is the set of all B tuples that cover b. When we join r_1 and r_2 we get $r(k)(ab) = p_1 p_2$ and the sum of $r(k)(ax)$ over tuples x that cover b is at most $p_1(p_2+p_3)$. When we perform $PSET(r(k))$, we assign probability $p_1(p_2+p_3)$ to $r'(k)(ab)$, obtaining the same function as $t(k)$. ○

Lemma 4.4. Let r' = JOIN r, s, where r is over K, A, and s is over A, B. Say $r'(k)(ab) = p$, where ab has no *'s. Then $Prob[AB = ab \mid K = k]$ is indeed greater than or equal to p.

Proof: If $p = 0$, the lemma is trivially true. If $p \neq 0$, it must have been computed as $p = r(k)(a) \times s(a)(b)$ (Definition 3.8). From Section 3.4, we know that $Prob[AB = ab \mid K = k] = Prob[A = a \mid K = k] \times Prob[B = b \mid A = a]$. We know that $Prob[A = a \mid K = k] \geq r(k)(a)$ and $Prob[B = b \mid A = a] \geq s(a)(b)$ (Section 3.1). Thus

$$Prob[AB = ab \mid K = k] \geq r(k)(a) \times s(a)(b) = p \quad ○$$