Noname manuscript No.
(will be inserted by the editor)

Xin Dong · Alon Halevy · Cong Yu

Data Integration with Uncertainty

the date of receipt and acceptance should be inserted later

Abstract This paper reports our first set of results on managing uncertainty in data integration. We posit that data-integration systems need to handle uncertainty at three levels and do so in a principled fashion. First, the semantic mappings between the data sources and the mediated schema may be approximate because there may be too many of them to be created and maintained or because in some domains (e.g., bioinformatics) it is not clear what the mappings should be. Second, the data from the sources may be extracted using information extraction techniques and so may yield erroneous data. Third, queries to the system may be posed with keywords rather than in a structured form.

As a first step to building such a system, we introduce the concept of probabilistic schema mappings and analyze their formal foundations. We show that there are two possible semantics for such mappings: *by-table* semantics assumes that there exists a correct mapping but we do not know what it is; *by-tuple* semantics assumes that the correct mapping may depend on the particular tuple in the source data. We present the query complexity and algorithms for answering queries in the presence of probabilistic schema mappings, and we describe an algorithm for efficiently computing the top-k answers to queries in such a setting. Finally, we consider using probabilistic mappings in the scenario of data exchange.

Xin Dong
AT&T Research
E-mail: lunadong@research.att.com
Florham Park, NJ 07932

Alon Halevy
Google Inc.
E-mail: halevy@google.com
Mountain View, CA 94043

Cong Yu
Yahoo! Research
E-mail: congyu@yahoo-inc.com
New York, NY 10018

1 Introduction

Data integration and exchange systems offer a uniform interface to a multitude of data sources and the ability to share data across multiple systems. These systems have recently enjoyed significant research and commercial success [18, 20]. Current data integration systems are essentially a natural extension of traditional database systems in that queries are specified in a structured form and data are modeled in one of the traditional data models (relational, XML). In addition, the data integration system has exact knowledge of how the data in the sources map to the schema used by the data integration system.

We argue that as the scope of data integration applications broadens, such systems need to be able to model uncertainty at their core. Uncertainty can arise for multiple reasons in data integration. First, the semantic mappings between the data sources and the mediated schema may be approximate. For example, in an application like Google Base [16] that enables anyone to upload structured data, or when mapping millions of sources on the deep web [25], we cannot imagine specifying exact mappings. In some domains (e.g., bioinformatics), we do not necessarily know what the exact mapping is. Second, data are often extracted from unstructured sources using information extraction techniques. Since these techniques are approximate, the data obtained from the sources may be uncertain. Finally, if the intended users of the application are not necessarily familiar with schemata, or if the domain of the system is too broad to offer form-based query interfaces (such as web forms), we need to support keyword queries. Hence, another source of uncertainty is the transformation between keyword queries and a set of candidate structured queries.

Dataspace Support Platforms [19] envision data integration systems where sources are added with no effort and the system is constantly evolving in a pay-as-you-go fashion to improve the quality of semantic mappings and query answering. Enabling data integration with uncertainty is a key technology to supporting dataspaces.

This paper takes a first step towards the goal of data integration with uncertainty. We first describe how the architecture of such a system differs from a traditional one (Section 2). At the core, the system models tuples and semantic mappings with probabilities associated with them. Query answering ranks answers and typically tries to obtain the top-k results to a query. These changes lead to a requirement for a new kind of adaptivity in query processing.

We then focus on one core component of data integration with uncertainty, namely probabilistic schema mappings (Section 3). Semantic mappings are the component of a data integration system that specify the relationship between the contents of the different sources. The mappings enable the data integration to reformulate a query posed over the mediated schema into queries over the sources [17, 22]. We introduce *probabilistic schema mappings*, and describe how to answer queries in their presence.

We define probabilistic schema mapping as a set of possible (ordinary) mappings between a source schema and a target schema, where each possible mapping has an associated probability. We begin by considering a simple class of mappings, where each mapping describes a set of correspondences between the attributes of a source table and the attributes of a target table. We argue that there are two possible interpretations of probabilistic schema mappings. In the first, which we formalize as *by-table* semantics, we assume there exists a single correct mapping between the source and the target, but we do not know which one it is. In the second, called *by-tuple* semantics, the correct mapping may depend on the particular tuple in the source to which it is applied. In both cases, the semantics of query answers are a generalization of certain answers [1] for data integration systems.

We describe algorithms for answering queries in the presence of probabilistic schema mappings and then analyze the computational complexity of answering queries (Section 4). We show that the data complexity of answering queries in the presence of probabilistic mappings is PTIME for by-table semantics and #P-complete for by-tuple semantics. We identify a large subclass of real-world queries for which we can still obtain all the by-tuple answers in PTIME. We then describe algorithms for finding the top-k answers to a query (Section 5).

The size of a probabilistic mapping may be quite large, since it essentially enumerates a probability distribution by listing every combination of events in the probability space. In practice, we can often encode the same probability distribution much more concisely. Our next contribution (Section 6) is to identify two concise representations of probabilistic mappings for which query answering can be performed in PTIME in the size of the mapping. We also examine the possibility of representing a probabilistic mapping as a Bayes Net, but show that query answering may still be exponential in the size of a Bayes Net representation of a mapping.

We then consider using probabilistic mappings in the scenario of data exchange (Section 7), where the goal is to create an instance of the target schema that is consistent with the data in the sources. We show that we can create a probabilistic database representing a *core universal solution* in polynomial time. As in the case of non-probabilistic mappings, the core universal solution can be used to find all the answers to a given query. This section also shows the close relationship between probabilistic databases and probabilistic schema mappings. In addition, we study some of the basic properties of probabilistic schema mappings: mapping composition and inversion (Section 8).

Finally, we consider several more powerful mapping languages, such as complex mappings, where the correspondences are between sets of attributes, and conditional mappings, where the mapping is conditioned on a property of the tuple to which it is applied (Section 9).

This article is an extended version of a previous conference paper [9]. The material in Sections 7 and 8 is new, as are the proofs of all the formal results. As follow-up work, [32] describes how to create probabilistic mappings and build a self-configuring data integration system. [32] has also reported experimental results on real-world data sets collected from the Web, showing that applying a probabilistic model in data integration enables producing high-quality query answers with no human intervention.

2 Overview of the System

This section describes the requirements from a data integration system that supports uncertainty and the overall architecture of the system. We frame our specific contributions in the context of this architecture.

2.1 Uncertainty in data integration

A data integration system needs to handle uncertainty at three levels.

Uncertain schema mappings: Data integration systems rely on schema mappings for specifying the semantic relationships between the data in the sources and the terms used in the mediated schema. However, schema mappings can be inaccurate. In many applications it is impossible to create and maintain precise mappings between data sources. This can be because the users are not skilled enough to provide precise mappings, such as in personal information management [8], because people do not understand the domain well and thus do not even know what correct mappings are, such as in bioinformatics, or because the scale of the data prevents generating

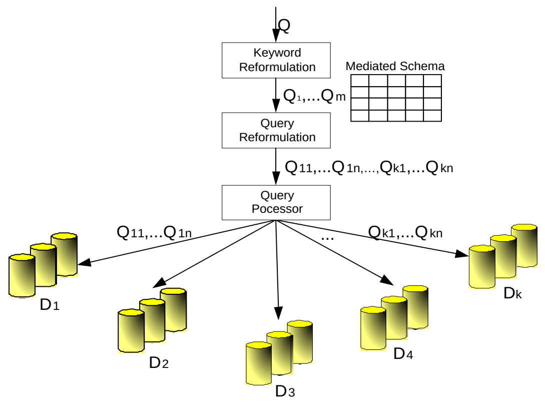

Fig. 1 Architecture of a data-integration system that handles uncertainty.

and maintaining precise mappings, such as in integrating data of the web scale [25]. Hence, in practice, schema mappings are often generated by semi-automatic tools and not necessarily verified by domain experts.

Uncertain data: By nature, data integration systems need to handle uncertain data. One reason for uncertainty is that data are often extracted from unstructured or semi-structured sources by automatic methods (e.g., HTML pages, emails, blogs). A second reason is that data may come from sources that are unreliable or not up to date. For example, in enterprise settings, it is common for informational data such as gender, racial, and income level to be dirty or missing, even when the transactional data is precise.

Uncertain queries: In some data integration applications, especially on the web, queries will be posed as keywords rather than as structured queries against a well defined schema. The system needs to translate these queries into some structured form so they can be reformulated with respect to the data sources. At this step, the system may generate multiple candidate structured queries and have some uncertainty about which is the real intent of the user.

2.2 System architecture

Given the previously discussed requirements, we describe the architecture of a data integration system that manages uncertainty at its core. We describe the system by contrasting it to a traditional data integration system.

The first and most fundamental characteristic of this system is that it is based on a probabilistic data model. This characteristic means two things. First, as we process data in the system we attach probabilities to each tuple. Second, and the focus of this paper, we associate schema mappings with probabilities, modeling the uncertainty about the correctness of the mappings. We use these probabilities to rank answers.

Second, whereas traditional data integration systems begin by reformulating a query onto the schemas of the data sources, a data integration system with uncertainty needs to first reformulate a keyword query into a set of candidate structured queries. We refer to this step as *keyword reformulation*. Note that keyword reformulation is different from techniques for keyword search on structured data (e.g., [21,2]) in that (a) it does not assume access to all the data in the sources or that the sources support keyword search, and (b) it tries to distinguish different structural elements in the query in order to pose more precise queries to the sources (e.g., realizing that in the keyword query "Chicago weather", "weather" is an attribute label and "Chicago" is an instance name). That being said, keyword reformulation should benefit from techniques that support answering keyword search on structured data.

Third, the query answering model is different. Instead of necessarily finding *all* answers to a given query, our goal is typically to find the top-k answers, and rank these answers most effectively.

The final difference from traditional data integration systems is that our query processing will need to be more adaptive than usual. Instead of generating a query answering plan and executing it, the steps we take in query processing will depend on results of previous steps. We note that adaptive query processing has been discussed quite a bit in data integration [23], where the need for adaptivity arises from the fact that data sources did not answer as quickly as expected or that we did not have accurate statistics about their contents to properly order our operations. In our work, however, the goal for adaptivity is to get the answers with high probabilities faster.

The architecture of the system is shown in Figure 1. The system contains a number of data sources and a mediated schema. When the user poses a query Q, which can be either a structured query on the mediated schema or a keyword query, the system returns a set of answer tuples, each with a probability. If Q is a keyword query, the system first performs keyword reformulation to translate it into a set of candidate structured queries on the mediated schema. Otherwise, the candidate query is Q itself.

Consider how the system answers the candidate queries, and assume the queries will not involve joins over multiple sources. For each candidate structured query Q_0 and a data source S, the system reformulates Q_0 according to the schema mapping (which can be uncertain) between S's schema and the mediated schema, sends the reformulated query (or queries) to S, retrieving the answers.

If the user asks for all the answers to the query, then the reformulated query is typically a query with grouping and aggregation, because the semantics of answers require aggregating the probabilities of answers from multiple sources. If S does not support grouping or aggregation, then grouping and aggregation needs be processed in the integration system.

If the user asks for top-k answers, then query processing is more complex. The system reformulates the query into a set of queries, uses a middle layer to decide at runtime which queries are critical to computing the top-k answers, and sends the appropriate queries to S. Note that we may need several iterations, where in each iteration we decide which are the promising reformulated queries to issue, and then retrieving answers. Furthermore, the system can even decide which data sources are more relevant and prioritize the queries to those data sources. Finally, if the data in the sources are uncertain, then the sources will return answers with probabilities attached to them.

After receiving answers from different data sources, the system combines them to get one single set of answer tuples. For example, if the data sources are known to be independent of each other, and we obtain tuple t from n data sources with probabilities $p_1, \ldots, p_n$ respectively, then in the final answer set t has probability $1 - \Pi_{i=1}^{n}(1 - p_i)$. If we know that some data sources are duplicates or extensions of others, a different combination function needs to be used.

2.3 Handling uncertainty in mappings

As a first step towards developing such a data integration system, we introduce in this paper *probabilistic schema mappings*, and show how to answer queries in their presence. Before the formal discussion, we illustrate the main ideas with an example.

Example 1 Consider a data source S, which describes a person by her email address, current address, and permanent address, and the mediated schema T, which describes a person by her name, email, mailing address, home address and office address:

```
S=(pname, email-addr, current-addr, permanent-addr)
T=(name, email, mailing-addr, home-addr, office-addr)
```

A semi-automatic schema-mapping tool may generate three possible mappings between S and T, assigning each a probability. Whereas the three mappings all map pname to name, they map other attributes in the source and the target differently. Figure 2(a) describes the three mappings using sets of attribute correspondences. For example, mapping m_1 maps pname to name, email-addr to email, current-addr to mailing-addr, and permanent-addr to home-addr. Because of the uncertainty about which mapping is correct, we consider all of these mappings in query answering.

Suppose the system receives a query Q formulated using the mediated schema and asking for people's mailing addresses:

```
Q: SELECT mailing-addr FROM T
```

Using the possible mappings, we can reformulate Q into different queries:

```
Q1: SELECT current-addr FROM S
Q2: SELECT permanent-addr FROM S
Q3: SELECT email-addr FROM S
```

If the user requires all possible answers, the system generates a single aggregation query based on Q_1, Q_2 and Q_3 to compute the probability of each returned tuple, and sends the query to the data source. Suppose the data source contains a table D_S as shown in Figure 2(b), the system will retrieve four answer tuples, each with a probability, as shown in Figure 2(c).

If the user requires only the top-1 answer (i.e., the answer tuple with the highest probability), the system decides at runtime which reformulated queries to execute. For example, after executing Q_1 and Q_2 at the source, the system can already conclude that ('Sunnyvale') is the top-1 answer and can skip query Q_3. □

2.4 Source of probabilities

A critical issue in any system that manages uncertainty is whether we have a reliable source of probabilities. Whereas obtaining reliable probabilities for such a system is one of the most interesting areas for future research, there is quite a bit to build on. For keyword reformulation, it is possible to train and test reformulators on large numbers of queries such that each reformulation result is given a probability based on its performance statistics. For information extraction, current techniques are often based on statistical machine learning methods and can be extended to compute probabilities of each extraction result. Finally, in the case of schema matching, it is standard practice for schema matchers to also associate numbers with the candidates they propose. The issue here is that the numbers are meant only as a ranking mechanism rather than true probabilities. However, as schema matching techniques start looking at a larger number of schemas, one can imagine ascribing probabilities (or estimations thereof) to their measures. Techniques on generating probabilistic mappings from schema matching results are presented in [32].

3 Probabilistic Schema Mapping

In this section we formally define the semantics of probabilistic schema mappings and the query answering problems we consider. Our discussion is in the context of the relational data model. A *schema* contains a finite set of relations. Each relation contains a finite set of *attributes* and is denoted by $R = \langle r_1, \ldots, r_n \rangle$. An *instance* D_R of R is a finite set of *tuples*, where each tuple associates a value with each attribute in the schema.

We consider select-project-join (SPJ) queries in SQL. Note that answering such queries is in PTIME in the size of the data.

	Possible Mapping	Prob
$m_1 =$	{(pname, name), (email-addr, email), (current-addr, mailing-addr), (permanent-addr, home-addr)}	0.5
$m_2 =$	{(pname, name), (email-addr, email), (permanent-addr, mailing-addr), (current-addr, home-addr)}	0.4
$m_3 =$	{(pname, name), (email-addr, mailing-addr), (current-addr, home-addr)}	0.1

(a)

*p*name	*e*mail-addr	*c*urrent-addr	*p*ermanent-addr
Alice	alice@	Mountain View	Sunnyvale
Bob	bob@	Sunnyvale	Sunnyvale

(b)

Tuple (mailing-addr)	Prob
('Sunnyvale')	0.9
('Mountain View')	0.5
('alice@')	0.1
('bob@')	0.1

(c)

Fig. 2 The running example: (a) a probabilistic schema mapping between S and T; (b) a source instance D_S; (c) the answers of Q over D_S with respect to the probabilistic mapping.

3.1 Schema mappings

We begin by reviewing non-probabilistic schema mappings. The goal of a schema mapping is to specify the semantic relationships between a *source schema* and a *target schema*. We refer to the source schema as $\bar{S}$, and a relation in $\bar{S}$ as $S = \langle s_1, \ldots, s_m \rangle$. Similarly, we refer to the target schema as $\bar{T}$, and a relation in $\bar{T}$ as $T = \langle t_1, \ldots, t_n \rangle$.

We consider a limited form of schema mappings that are also referred to as *schema matching* in the literature [30]. Specifically, a schema matching contains a set of *attribute correspondences*. An attribute correspondence is of the form $c_{ij} = (s_i, t_j)$, where s_i is a *source attribute* in the schema S and t_j is a *target attribute* in the schema T. Intuitively, c_{ij} specifies that there is a relationship between s_i and t_j. In practice, a correspondence also involves a function that transforms the value of s_i to the value of t_j. For example, the correspondence (c-degree, temperature) can be specified as temperature=c-degree $*1.8+32$, describing a transformation from Celsius to Fahrenheit. These functions are irrelevant to our discussion, and therefore we omit them. We consider this class of mappings because they already expose many of the novel issues involved in probabilistic mappings and because they are quite common in practice. We also note that many of the concepts we define apply to a broader class of mappings, which we will discuss in detail in Section 4.1.

Formally, we define relation mappings and schema mappings as follows.

Definition 1 (Schema Mapping) Let $\bar{S}$ and $\bar{T}$ be relational schemas. A *relation mapping* M is a triple (S, T, m), where S is a relation in $\bar{S}$, T is a relation in $\bar{T}$, and m is a set of attribute correspondences between S and T.

When each source and target attribute occurs in at most one correspondence in m, we call M a *one-to-one relation mapping.*

A *schema mapping* $\overline{M}$ is a set of one-to-one relation mappings between relations in $\bar{S}$ and in $\bar{T}$, where every relation in either $\bar{S}$ or $\bar{T}$ appears at most once. □

A pair of instances D_S and D_T *satisfies* a relation mapping m if for every source tuple $t_s \in D_S$, there exists a target tuple $t_t \in D_t$, such that for every attribute correspondence $(s, t) \in m$, the value of attribute s in t_s is the same as the value of attribute t in t_t.

Example 2 Consider the mappings in Example 1. The source database in Figure 2(b) (repeated in Figure 3(a)) and the target database in Figure 3(b) satisfy m_1. □

3.2 Probabilistic schema mappings

Intuitively, a probabilistic schema mapping describes a probability distribution of a set of *possible* schema mappings between a source schema and a target schema.

Definition 2 (Probabilistic Mapping) Let $\bar{S}$ and $\bar{T}$ be relational schemas. A *probabilistic mapping (p-mapping)*, pM, is a triple $(S, T, \mathbf{m})$, where $S \in \bar{S}$, $T \in \bar{T}$, and $\mathbf{m}$ is a set $\{(m_1, Pr(m_1)), \ldots, (m_l, Pr(m_l))\}$, such that

- for $i \in [1, l]$, m_i is a one-to-one mapping between S and T, and for every $i, j \in [1, l]$, $i \neq j \Rightarrow m_i \neq m_j$.
- $Pr(m_i) \in [0, 1]$ and $\sum_{i=1}^{l} Pr(m_i) = 1$.

A *schema p-mapping*, $\overline{pM}$, is a set of p-mappings between relations in $\bar{S}$ and in $\bar{T}$, where every relation in either $\bar{S}$ or $\bar{T}$ appears in at most one p-mapping. □

We refer to a non-probabilistic mapping as an *ordinary mapping*. A schema p-mapping may contain both p-mappings and ordinary mappings. Example 1 shows a p-mapping (see Figure 2(a)) that contains three possible mappings.

*p*name	*e*mail-addr	*c*urrent-addr	*p*ermanent-addr
Alice	alice@	Mountain View	Sunnyvale
Bob	bob@	Sunnyvale	Sunnyvale

(a)

*n*ame	*e*mail	*m*ailing-addr	*h*ome-addr	*o*ffice-addr
Alice	alice@	Mountain View	Sunnyvale	office
Bob	bob@	Sunnyvale	Sunnyvale	office

(b)

*n*ame	*e*mail	*m*ailing-addr	*h*ome-addr	*o*ffice-addr
Alice	alice@	Sunnyvale	Mountain View	office
Bob	email	bob@	Sunnyvale	office

(c)

Tuple (mailing-addr)	Prob
('Sunnyvale')	0.9
('Mountain View')	0.5
('alice@')	0.1
('bob@')	0.1

(d)

Tuple (mailing-addr)	Prob
('Sunnyvale')	0.94
('Mountain View')	0.5
('alice@')	0.1
('bob@')	0.1

(e)

Fig. 3 Example 3: (a) a source instance D_S; (b) a target instance that is by-table consistent with D_S and m_1; (c) a target instance that is by-tuple consistent with D_S and $< m_2, m_3 >$; (d) $Q^{table}(D_S)$; (e) $Q^{tuple}(D_S)$.

3.3 Semantics of probabilistic mappings

Intuitively, a probabilistic schema mapping models the uncertainty about which of the mappings in pM is the correct one. When a schema matching system produces a set of candidate matches, there are two ways to interpret the uncertainty: (1) a single mapping in pM is the correct one and it applies to all the data in S, or (2) several mappings are partially correct and each is suitable for a subset of tuples in S, though it is not known which mapping is the right one for a specific tuple. Example 1 illustrates the first interpretation and query rewriting under this interpretation. For the same example, the second interpretation is equally valid: some people may choose to use their current address as mailing address while others use their permanent address as mailing address; thus, for different tuples we may apply different mappings, so the correct mapping depends on the particular tuple.

This paper analyzes query answering under both interpretations. We refer to the first interpretation as the *by-table* semantics and to the second one as the *by-tuple* semantics of probabilistic mappings. We are not trying to argue for one interpretation over the other. The needs of the application should dictate the appropriate semantics. Furthermore, our complexity results, which will show advantages to by-table semantics, should not be taken as an argument in the favor of by-table semantics.

We next define query answering with respect to p-mappings in detail and the definitions for schema p-mappings are the obvious extensions. Recall that given a query and an ordinary mapping, we can compute *certain answers* to the query with respect to the mapping. Query answering with respect to p-mappings is defined as a natural extension of certain answers, which we next review.

A mapping defines a relationship between instances of S and instances of T that are *consistent* with the mapping.

Definition 3 (Consistent Target Instance) Let $M = (S, T, m)$ be a relation mapping and D_S be an instance of S.

An instance D_T of T is said to be *consistent with* D_S *and* M, if for each tuple $t_s \in D_S$, there exists a tuple $t_t \in D_T$, such that for every attribute correspondence $(a_s, a_t) \in m$, the value of a_s in t_s is the same as the value of a_t in t_t. □

For a relation mapping M and a source instance D_S, there can be an infinite number of target instances that are consistent with D_S and M. We denote by $Tar_M(D_S)$ the set of all such target instances. The set of answers to a query Q is the intersection of the answers on all instances in $Tar_M(D_S)$. The following definition is from [1].

Definition 4 (Certain Answer) Let $M = (S, T, m)$ be a relation mapping. Let Q be a query over T and let D_S be an instance of S.

A tuple t is said to be a *certain answer of* Q *with respect to* D_S *and* M, if for every instance $D_T \in Tar_M(D_S)$, $t \in Q(D_T)$. □

By-table semantics: We now generalize these notions to the probabilistic setting, beginning with the by-table semantics. Intuitively, a p-mapping pM describes a set of possible worlds, each with a possible mapping $m \in pM$. In by-table semantics, a source table can fall in one of the possible worlds; that is, the possible mapping associated with that possible world applies to the whole source table. Following this intuition, we define target instances that are *consistent with* the source instance.

Definition 5 (By-table Consistent Instance) Let $pM = (S, T, \mathbf{m})$ be a p-mapping and D_S be an instance of S.

An instance D_T of T is said to be *by-table consistent with D_S and pM*, if there exists a mapping $m \in \mathbf{m}$ such that D_S and D_T satisfy m. □

Given a source instance D_S and a possible mapping $m \in \mathbf{m}$, there can be an infinite number of target instances that are consistent with D_S and m. We denote by $Tar_m(D_S)$ the set of all such instances.

In the probabilistic context, we assign a probability to every answer. Intuitively, we consider the certain answers with respect to each possible mapping in isolation. The probability of an answer t is the sum of the probabilities of the mappings for which t is deemed to be a certain answer. We define by-table answers as follows:

Definition 6 (By-table Answer) Let $pM = (S, T, \mathbf{m})$ be a p-mapping. Let Q be a query over T and let D_S be an instance of S.

Let t be a tuple. Let $\bar{m}(t)$ be the subset of $\mathbf{m}$, such that for each $m \in \bar{m}(t)$ and for each $D_T \in Tar_m(D_S)$, $t \in Q(D_T)$.

Let $p = \sum_{m \in \bar{m}(t)} Pr(m)$. If $p > 0$, then we say (t, p) is a *by-table answer of Q with respect to D_S and pM*. □

By-tuple semantics: If we follow the possible-world notions, in by-tuple semantics, different tuples in a source table can fall in different possible worlds; that is, different possible mappings associated with those possible worlds can apply to the different source tuples.

Formally, the key difference in the definition of by-tuple semantics from that of by-table semantics is that a consistent target instance is defined by a mapping *sequence* that assigns a (possibly different) mapping in $\mathbf{m}$ to each source tuple in D_S. (Without losing generality, in order to compare between such sequences, we assign some order to the tuples in the instance).

Definition 7 (By-tuple Consistent Instance) Let $pM = (S, T, \mathbf{m})$ be a p-mapping and let D_S be an instance of S with d tuples.

An instance D_T of T is said to be *by-tuple consistent with D_S and pM*, if there is a sequence $\langle m^1, \ldots, m^d \rangle$, where d is the number of tuples in D_S, and for every $1 \leq i \leq d$,

– $m^i \in \mathbf{m}$, and
– for the i^{th} tuple of D_S, t_i, there exists a target tuple $t'_i \in D_T$ such that for each attribute correspondence $(a_s, a_t) \in m^i$, the value of a_s in t_i is the same as the value of a_t in t'_i . □

Given a mapping sequence $seq = \langle m^1, \ldots, m^d \rangle$, we denote by $Tar_{seq}(D_S)$ the set of all target instances that are consistent with D_S and seq. Note that if D_T is by-table consistent with D_S and m, then D_T is also by-tuple consistent with D_S and a mapping sequence in which each mapping is m.

We can think of every sequence of mappings $seq = \langle m^1, \ldots, m^d \rangle$ as a separate event whose probability is $Pr(seq) = \Pi_{i=1}^{d} Pr(m^i)$. (In Section 9 we relax this independence assumption and introduce *conditional mappings*.) If there are l mappings in pM, then there are l^d sequences of length d, and their probabilities add up to 1. We denote by $\mathbf{seq}_d(pM)$ the set of mapping sequences of length d generated from pM.

Definition 8 (By-tuple Answer) Let $pM = (S, T, \mathbf{m})$ be a p-mapping. Let Q be a query over T and D_S be an instance of S with d tuples.

Let t be a tuple. Let $\overline{seq}(t)$ be the subset of $\mathbf{seq}_d(pM)$, such that for each $seq \in \overline{seq}(t)$ and for each $D_T \in Tar_{seq}(D_S)$, $t \in Q(D_T)$.

Let $p = \sum_{seq \in \overline{seq}(t)} Pr(seq)$. If $p > 0$, we call (t, p) a *by-tuple answer of Q with respect to D_S and pM*. □

The set of by-table answers for Q with respect to D_S is denoted by $Q^{table}(D_S)$ and the set of by-tuple answers for Q with respect to D_S is denoted by $Q^{tuple}(D_S)$.

Example 3 Consider the p-mapping pM, the source instance D_S, and the query Q in the motivating example.

In by-table semantics, Figure 3(b) shows a target instance that is consistent with D_S (repeated in Figure 3(a)) and possible mapping m_1. Figure 3(d) shows the by-table answers of Q with respect to D_S and pM. As an example, for tuple $t =$('Sunnyvale'), we have $\bar{m}(t) = \{m_1, m_2\}$, so the possible tuple ('Sunnyvale', 0.9) is an answer.

In by-tuple semantics, Figure 3(c) shows a target instance that is by-tuple consistent with D_S and the mapping sequence $< m_2, m_3 >$. Figure 3(e) shows the by-tuple answers of Q with respect to D_S and pM. Note that the probability of tuple t=('Sunnyvale') in the by-table answers is different from that in the by-tuple answers. We describe how to compute the probabilities in detail in the next section. □

4 Complexity of Query Answering

This section considers query answering in the presence of probabilistic mappings. We describe algorithms for query answering and study the complexity of query answering in terms of the size of the data (*data complexity*) and the size of the p-mapping (*mapping complexity*). We note that the number of possible mappings in a p-mapping can be exponential in the number of source or target attributes; we discuss more compressive representations of p-mappings in Section 6. We also consider cases in which we are not interested in the actual probability of an answer, just whether or not a tuple is a possible answer.

We show that when the schema is fixed, returning all by-table answers is in PTIME for both complexity measures, whereas returning all by-tuple answers in general is #P-complete with respect to the data complexity. Recall that #P is the complexity class of some hard counting problems (*e.g.*, counting the number of variable assignments that satisfy a Boolean formula). It is believed that a #P-complete problem cannot be solved in polynomial time, unless $P = NP$. We show that computing the probabilities is the culprit here: even deciding the probability of a *single* answer tuple under by-tuple semantics is already #P-complete, whereas computing all by-tuple answers without returning the probabilities is in PTIME. Finally, we identify a large subclass of common queries where returning all by-tuple answers with their probabilities is still in PTIME.

We note that our complexity results are for ordinary databases (*i.e.*, deterministic data). Query answering on probabilistic data in itself can be #P-complete [33] and thus query answering on probabilistic data with respect to p-mappings is at least #P-hard. Extending our results for probabilistic data is rather involving and we leave it for future work.

4.1 By-table query answering

In the case of by-table semantics, answering queries is conceptually simple. Given a p-mapping $pM = (S, T, \mathbf{m})$ and an SPJ query Q, we can compute the certain answers of Q under each of the mappings $m \in \mathbf{m}$. We attach the probability $Pr(m)$ to every certain answer under m. If a tuple is an answer to Q under multiple mappings in $\mathbf{m}$, then we add up the probabilities of the different mappings.

Algorithm ByTable takes as input an SPJ query Q that mentions the relations $T_1, \ldots, T_l$ in the `FROM` clause. Assume that we have the p-mapping pM_i associated with the table T_i. The algorithm proceeds as follows.

Step 1: We generate the possible reformulations of Q (a reformulation query computes all certain answers when executed on the source data) by considering every combination of the form $(m^1, \ldots, m^l)$, where m^i is one of the possible mappings in pM_i. Denote the set of reformulations by $Q'_1, \ldots, Q'_k$. The probability of a reformulation $Q' = (m^1, \ldots, m^l)$ is $\Pi_{i=1}^{l} Pr(m^i)$.

Step 2: For each reformulation Q', retrieve each of the unique answers from the sources. For each answer obtained by $Q'_1 \cup \ldots \cup Q'_k$, its probability is computed by summing the probabilities of the Q''s in which it is returned.

Importantly, note that it is possible to express both steps as an SQL query with grouping and aggregation. Therefore, if the underlying sources support SQL, we can leverage their optimizations to compute the answers.

With our restricted form of schema mapping, the algorithm takes time polynomial in the size of the data

Tuple (mailing-addr)	Pr
('Sunnyvale')	0.94
('Mountain View')	0.5
('alice@')	0.1
('bob@')	0.1

(a)

Tuple (mailing-addr)	Pr
('Sunnyvale')	0.8
('Mountain View')	0.8

(b)

Fig. 4 Example 4: (a) $Q_1^{tuple}(D)$ and (b) $Q_2^{tuple}(D)$.

and the mappings. We thus have the following complexity result. We give full proofs for results in this paper in the appendix.

Theorem 1 *Let $\overline{pM}$ be a schema p-mapping and let Q be an SPJ query.*

Answering Q with respect to $\overline{pM}$ in by-table semantics is in PTIME in the size of the data and the mapping. □

This result holds for more general mappings, as we explain next.

GLAV mappings: The common formalism for schema mappings, GLAV, is based on expressions of the form

$$m : \forall \mathbf{x} (\varphi(\mathbf{x}) \rightarrow \exists \mathbf{y} \psi(\mathbf{x}, \mathbf{y})).$$

In the expression, φ is the body of a conjunctive query over $\bar{S}$ and ψ is the body of a conjunctive query over $\bar{T}$. A pair of instances D_S and D_T *satisfies* a GLAV mapping m if for every assignment of $\mathbf{x}$ in D_S that satisfies φ there exists an assignment of $\mathbf{y}$ in D_T that satisfies ψ.

The schema mapping we have considered so far is a limited form of GLAV mappings where each side of the mapping involves only projection queries on a single table. However, it is rather straightforward to extend the complexity results for this limited form of schema mappings to arbitrary GLAV mappings.

We define *general p-mappings* to be triples of the form $pGM = (\bar{S}, \bar{T}, \mathbf{gm})$, where $\mathbf{gm}$ is a set $\{(gm_i, Pr(gm_i)) \mid i \in [1, n]\}$, such that for each $i \in [1, n]$, gm_i is a general GLAV mapping. The definition of by-table semantics for such mappings is a simple generalization of Definition 6. The following result holds for general p-mappings.

Theorem 2 *Let pGM be a general p-mapping between a source schema $\bar{S}$ and a target schema $\bar{T}$. Let D_S be an instance of $\bar{S}$. Let Q be an SPJ query with only equality conditions over $\bar{T}$. The problem of computing $Q^{table}(D_S)$ with respect to pGM is in PTIME in the size of the data and the mapping.* □

4.2 By-tuple query answering

To extend the by-table query-answering strategy to by-tuple semantics, we would need to compute the certain answers for every *mapping sequence* generated by pM. However, the number of such mapping sequences is exponential in the size of the input data. The following example shows that for certain queries this exponential time complexity is inevitable.

Example 4 Suppose that in addition to the tables in Example 1, we also have U(city) in the source and V(hightech) in the target. The p-mapping for V contains two possible mappings: ({(city, hightech)}, .8) and ($\emptyset$, .2).

Consider the following query Q, which decides if there are any people living in a high-tech city.

```
Q: SELECT 'true'
   FROM T, V
   WHERE T.mailing-addr = V.hightech
```

An incorrect way of answering the query is to first execute the following two sub-queries Q_1 and Q_2, then join the answers of Q_1 and Q_2 and summing up the probabilities.

```
Q1: SELECT mailing-addr FROM T
Q2: SELECT hightech FROM V
```

Now consider the source instance D, where D_S is shown in Figure 2(a), and D_U has two tuples ('Mountain View') and ('Sunnyvale'). Figure 4(a) and (b) show $Q_1^{tuple}(D)$ and $Q_2^{tuple}(D)$. If we join the results of Q_1 and Q_2, we obtain for the true tuple the following probability: $0.94{*}0.8{+}0.5{*}0.8 = 1.152$. However, this is incorrect. By enumerating all consistent target tables, we in fact compute 0.864 as the probability. The reason for this error is that on some target instance that is by-tuple consistent with the source instance, the answers to both Q_1 and Q_2 contain tuple ('Sunnyvale') and tuple ('Mountain View'). Thus, generating the tuple ('Sunnyvale') as an answer for both Q_1 and Q_2 and generating the tuple ('Mountain View') for both queries are not independent events, and so simply adding up their probabilities leads to incorrect results.

Indeed, we do not know a better algorithm to answer Q than by enumerating all by-tuple consistent target instances and then answering Q on each of them. □

In fact, we show that in general, answering SPJ queries in by-tuple semantics with respect to schema p-mappings is hard.

Theorem 3 *Let Q be an SPJ query and let $\overline{pM}$ be a schema p-mapping. The problem of finding the probability for a by-tuple answer to Q with respect to $\overline{pM}$ is #P-complete with respect to data complexity and is in PTIME with respect to mapping complexity.* □

The lower bound in Theorem 3 is proved by reducing the problem of counting the number of variable assignments that satisfy a bipartite monotone 2DNF Boolean formula to the problem of finding the answers to Q.

In fact, the reason for the high complexity is exactly that we are asking for the probability of the answer. The following theorem shows that if we want to know only the possible by-tuple answers, we can do so in polynomial time.

Theorem 4 *Given an SPJ query and a schema p-mapping, returning all by-tuple answers without probabilities is in PTIME with respect to data complexity.* □

The key to proving the PTIME complexity is that we can find all by-tuple answer tuples (without knowing the probability) by answering the query on the *mirror target* of the source data. Formally, let D_S be the source data and $\overline{pM}$ be the schema p-mapping. The mirror target of D_S with respect to $\overline{pM}$ is defined as follows. If R is not involved in any mapping, the mirror target contains R itself; if R is the target of $pM = (S, T, \mathbf{m}) \in \overline{pM}$, the mirror target contains a relation R' where for each source tuple t_S of S and each $m \in \mathbf{m}$, there is a tuple t_T in R' that (1) is consistent with t_S and m and contains null value for each attribute that is not involved in m, (2) contains an id column with the value of the id column in t_S (we assume the existence of identifier attribute id for S and in practice we can use S's key attributes in place of id), and (3) contains a mapping column with the identifier of m. Meanwhile, we slightly modify a query Q into a *mirror query* Q_m with respect to $\overline{pM}$ as follows: Q_m is the same as Q except that for each relation R that is the target of a p-mapping in $\overline{pM}$ and occurs multiple times in Q's FROM clause, and for any of R's two aliases R_1 and R_2 in the FROM clause, Q' contains in addition the following predicates: (R_1.id <> R_2.id OR R_1.mapping=R_2.mapping).

Lemma 1 *Let $\overline{pM}$ be a schema p-mapping. Let Q be an SPJ query and Q_m be Q's mirror query with respect to $\overline{pM}$. Let D_S be the source database and D_T be the mirror target of D_S with respect to $\overline{pM}$.*

Then, $t \in Q^{tuple}(D_S)$ if and only if $t \in Q_m(D_T)$ and t does not contain null *value.* □

The size of the mirror target is polynomial in the size of the data and the p-mapping. The PTIME complexity bound follows from the fact that answering the mirror query on the mirror target takes only polynomial time.

GLAV mappings: Extending by-tuple semantics to arbitrary GLAV mappings is much trickier than by-table semantics. It would involve considering mapping sequences whose length is the product of the number of tuples in each source table, and the results are much less intuitive. Hence, we postpone by-tuple semantics to future work.

4.3 Two restricted cases

In this section we identify two restricted but common classes of queries for which by-tuple query answering takes polynomial time. While we do not have a necessary and sufficient condition for PTIME complexity of query answering, we do not know any other cases where it is possible to answer a query in polynomial time.

In our discussion we refer to *subgoals* of a query. The subgoals are tables that occur in the FROM clause of a query. Hence, even if the same table occurs twice in the FROM clause, each occurrence is a different subgoal.

Queries with a single p-mapping subgoal

The first class of queries we consider are those that include only a single subgoal being the target of a p-mapping. Relations in the other subgoals are either involved in ordinary mappings or do not require a mapping. Hence, if we only have uncertainty with respect to one part of the domain, our queries will typically fall in this class. We call such queries *non-p-join queries*. The query Q in the motivating example is an example non-p-join query.

Definition 9 (non-p-join queries) Let $\overline{pM}$ be a schema p-mapping and let Q be an SPJ query.

If at most one subgoal in the body of Q is the target of a p-mapping in $\overline{pM}$, we say Q is a *non-p-join query with respect to* $\overline{pM}$. □

For a non-p-join query Q, the by-tuple answers of Q can be generated from the by-table answers of Q over a set of databases, each containing a single tuple in the source table. Specifically, let $pM = (S, T, \mathbf{m})$ be the single p-mapping whose target is a relation in Q, and let D_S be an instance of S with d tuples. Consider the set of *tuple databases* $\mathbf{T}(D_S) = \{D_1, \ldots, D_d\}$, where for each $i \in [1, d]$, D_i is an instance of S and contains only the i-th tuple in D_S. The following lemma shows that $Q^{tuple}(D_S)$ can be derived from $Q^{table}(D_1), \ldots, Q^{table}(D_d)$.

Lemma 2 *Let $\overline{pM}$ be a schema p-mapping between $\bar{S}$ and $\bar{T}$. Let Q be a non-p-join query over $\bar{T}$ and let D_S be an instance of $\bar{S}$. Let $(t, Pr(t))$ be a by-tuple answer with respect to D_S and $\overline{pM}$. Let $\bar{T}(t)$ be the subset of $\mathbf{T}(D_S)$ such that for each $D \in \bar{T}(t)$, $t \in Q^{table}(D)$. The following two conditions hold:*

1. *$\bar{T}(t) \neq \emptyset$;*
2. *$Pr(t) = 1 - \Pi_{D\in\bar{T}(t),(t,p)\in Q^{table}(D)}(1 - p)$.* □

In practice, answering the query for each tuple database can be expensive. We next describe Algorithm NonPJoin, which computes the answers for all tuple databases in one step. The key of the algorithm is to distinguish answers generated by different source tuples. To do this, we assume there is an identifier attribute id for the source relation whose values are concatenations of values of the key columns. We now describe the algorithm in detail.

Algorithm NonPJoin takes as input a non-p-join query Q, a schema p-mapping $\overline{pM}$, and a source instance D_S, and proceeds in three steps to compute all by-tuple answers.

Step 1: Rewrite Q to Q' such that it returns T.id in addition. Revise the p-mapping such that each possible mapping contains the correspondence between S.id and T.id.

Step 2: Invoke ByTable with Q', $\overline{pM}$ and D_S. Note that each generated result tuple contains the id column in addition to the attributes returned by Q.

Step 3: Project the answers returned in Step 2 on Q's returned attributes. Suppose projecting $t_1, \ldots, t_n$ obtains the answer tuple t, then the probability of t is $1 - \Pi_{i=1}^{n}(1 - Pr(t_i))$.

Note that Algorithm NonPJoin is different from Algorithm ByTable in two ways. First, it considers an identifier column of the source and so essentially it can answer the query on all tuple databases parallelly. Second, whereas ByTable combines the results from rewritten queries simply by adding up the probabilities of each distinct tuple t, NonPJoin needs to in addition compute $1 - \Pi_{i=1}^{n}(1 - Pr(t_i))$ for each tuple t_i projecting which obtains answer tuple t.

Example 5 Consider rewriting Q in the motivating example, repeated as follows:

```
Q: SELECT mailing-addr FROM T
```

Step 1 rewrites Q into query Q' by adding the id column:

```
Q': SELECT id, mailing-addr FROM T
```

In Step 2, ByTable may generate the following *SQL* query to compute by-table answers for Q':

```
Qa: SELECT id, mailing-addr, SUM(pr)
    FROM (
      SELECT DISTINCT id, current-addr
              AS mailing-addr, 0.5 AS pr
      FROM S
      UNION ALL
      SELECT DISTINCT id, permanent-addr
              AS mailing-addr, 0.4 AS pr
      FROM S
      UNION ALL
      SELECT DISTINCT id, email-addr
              AS mailing-addr, 0.1 AS pr
      FROM S)
    GROUP BY id, mailing-addr
```

Step 3 then generates the results using the following query.

```
Qu:  SELECT mailing-addr, NOR(pr) AS pr
     FROM Qa
     GROUP BY mailing-addr
```

where for a set of probabilities $pr_1, \ldots, pr_n$, NOR computes $1 - \Pi_{i=1}^{n}(1 - pr_i)$. □

An analysis of Algorithm NonPJoin leads to the following complexity result for non-p-join queries.

Theorem 5 *Let $\overline{pM}$ be a schema p-mapping and let Q be a non-p-join query with respect to $\overline{pM}$.*

Answering Q with respect to $\overline{pM}$ in by-tuple semantics is in PTIME in the size of the data and the mapping. □

Projected p-join queries

We now show that query answering can be done in polynomial time for a class of queries, called *projected p-join queries*, that include multiple subgoals involved in p-mappings. In such a query, we say that a join predicate is a *p-join predicate* with respect to a schema p-mapping $\overline{pM}$, if at least one of the involved relations is the target of a p-mapping in $\overline{pM}$. We define projected p-join queries as follows.

Definition 10 (projected p-join query) Let $\overline{pM}$ be a schema p-mapping and Q be an SPJ query over the target of $\overline{pM}$. If the following conditions hold, we say Q is a *projected p-join query with respect to* $\overline{pM}$:

- at least two subgoals in the body of Q are targets of p-mappings in $\overline{pM}$.
- for every p-join predicate, the join attribute (or an equivalent attribute implied by the predicates in Q) is returned in the SELECT clause. □

Example 6 Consider the schema p-mapping in Example 4. A slight revision of Q, shown as follows, is a projected-p-join query.

```
Q': SELECT V.hightech
    FROM T, V
    WHERE T.mailing-addr = V.hightech
```

□

Note that in practice, when joining data from multiple tables in a data integration scenario, we typically project the join attributes, thereby leading to projected p-join queries.

The key to answering a projected-p-join query Q is to divide Q into multiple subqueries, each of which is a non-p-join query, and compute the answer to Q from the answers to the subqueries. We proceed by considering partitions of the subgoals in Q. We say that a partitioning $\bar{J}$ is a *refinement* of a partitioning $\bar{J}'$, denoted $\bar{J} \preceq \bar{J}'$, if for each partition $J \in \bar{J}$, there is a partition $J' \in \bar{J}'$, such that $J \subseteq J'$. We consider the following partitioning of Q, the generation of which will be described in detail in the algorithm.

Definition 11 (Maximal P-Join Partitioning) Let $\overline{pM}$ be a schema p-mapping. Let Q be an SPJ query and $\bar{J}$ be a partitioning of the subgoals in Q.

We say that $\bar{J}$ is a *p-join partitioning of* Q, if (1) each partition $J \in \bar{J}$ contains at most one subgoal that is the target of a p-mapping in $\overline{pM}$, and (2) if neither subgoal in a join predicate is involved in p-mappings in $\overline{pM}$, the two subgoals belong to the same partition.

We say that $\bar{J}$ is a *maximal p-join partitioning of* Q, if there does not exist a p-join partitioning $\bar{J}'$, such that $\bar{J} \preceq \bar{J}'$. □

For each partition $J \in \bar{J}$, we can define a query Q_J as follows. The FROM clause includes the subgoals in J. The SELECT clause includes J's attributes that occur in (1) Q's SELECT clause or (2) Q's join predicates that join subgoals in J with subgoals in other partitions. The WHERE clause includes Q's predicates that contain only subgoals in J. When J is a partition in a maximal p-join partitioning of Q, we say that Q_J is a *p-join component* of Q.

The following is the main lemma underlying our algorithm. It shows that we can compute the answers of Q from the answers to its p-join components.

Lemma 3 *Let $\overline{pM}$ be a schema p-mapping. Let Q be a projected p-join query with respect to $\overline{pM}$ and let $\bar{J}$ be a maximal p-join partitioning of Q. Let $Q_{J1}, \ldots, Q_{Jn}$ be the p-join components of Q with respect to $\bar{J}$.*

For any instance D_S of the source schema of $\overline{pM}$ and result tuple $t \in Q^{tuple}(D_S)$, the following two conditions hold:

1. *For each $i \in [1, n]$, there exists a single tuple $t_i \in Q_{Ji}^{tuple}(D_S)$, such that $t_1, \ldots, t_n$ generate t when joined together.*
2. *Let $t_1, \ldots, t_n$ be the above tuples. Then $Pr(t) = \Pi_{i=1}^{n} Pr(t_i)$.* □

Lemma 3 leads naturally to the query-answering algorithm ProjectedPJoin, which takes as input a projected-p-join query Q, a schema p-mapping $\overline{pM}$, and a source instance D_S, outputs all by-tuple answers, and proceeds in three steps.

Step 1: Generate maximum p-join partitions $J_1, \ldots, J_n$ as follows. First, initialize each partition to contain one subgoal in Q. Then, for each join predicate with subgoals S_1 and S_2 that are not involved in p-mappings in $\overline{pM}$, merge the partitions that S_1 and S_2 belong to. Finally, for each partition that contains no subgoal involved in $\overline{pM}$, merge it with another partition.

Step 2: For each p-join partition $J_i, i \in [1, n]$, generate the p-join component Q_{Ji} and invoke Algorithm NonPJoin with Q_{Ji}, $\overline{pM}$ and D_S to compute answers for Q_{Ji}.

Step 3: Join the results of $Q_{J1}, \ldots, Q_{Jn}$. If an answer tuple t is obtained by joining $t_1, \ldots, t_n$, then the probability of t is computed by $\Pi_{i=1}^{n} Pr(t_i)$.

We illustrate the algorithm using the following example.

Example 7 Consider query Q' in Example 6. Its two p-join components are Q_1 and Q_2 shown in Example 4. Suppose we compute Q_1 with query Q_u (shown in Example 5) and compute Q_2 with query Q'_u. We can compute by-tuple answers of Q' as follows:

```
SELECT Qu'.hightech, Qu.pr*Qu'.pr
FROM Qu, Qu'
WHERE Qu.mailing-addr = Qu'.hightect
```

□

Since the number of p-join components is bounded by the number of subgoals in a query, and for each of them we invoke Algorithm NonPJoin, query answering for projected p-join queries takes polynomial time.

Theorem 6 *Let $\overline{pM}$ be a schema p-mapping and let Q be a projected-p-join query with respect to $\overline{pM}$.*

Answering Q with respect to $\overline{pM}$ in by-tuple semantics is in PTIME in the size of the data and the mapping. □

Other SPJ queries

A natural question is whether the two classes of queries we have identified are the only ones for which query answering is in PTIME for by-tuple semantics. If Q contains multiple subgoals that are involved in a schema p-mapping, but Q is not a projected-p-join query, then Condition 1 in Lemma 3 does not hold and the technique for answering projected-p-join queries do not apply any more. We do not know any better algorithm to answer such queries than enumerating all mapping sequences.

We believe that the complexity of the border case, where a query joins two relations involved in p-mappings but does not return the join attribute, is #P-hard, but currently it remains an open problem.

5 Top-*K* Query Answering

In this section, we consider returning the top-k query answers, which are the k answer tuples with the top probabilities. The main challenge in designing the algorithm is to only perform the necessary reformulations at every step and halt when the top-k answers are found. We first describe our algorithm for by-table semantics. We then show the challenges for by-tuple semantics and outline our solution.

5.1 Returning top-k by-table answers

Recall that in by-table query answering, the probability of an answer is the sum of the probabilities of the reformulated queries that generate the answer. Our goal is to reduce the number of reformulated queries we execute. Our algorithm proceeds in a greedy fashion: we execute queries in descending order of probabilities. For each tuple t, we maintain the upper bound $p_{max}(t)$ and lower bound $p_{min}(t)$ of its probability. This process halts when we find k tuples whose p_{min} values are higher than p_{max} of the rest of the tuples.

TopKByTable takes as input an SPJ query Q, a schema p-mapping $\overline{pM}$, an instance D_S of the source schema, and an integer k, and outputs the top-k answers in $Q^{table}(D_S)$. The algorithm proceeds in three steps.

Step 1: Rewrite Q according to $\overline{pM}$ into a set of queries $Q_1, \ldots, Q_n$, each with a probability assigned in a similar way as stated in Algorithm ByTable.

Step 2: Execute $Q_1, \ldots, Q_n$ in descending order of their probabilities. Maintain the following measures:

- The highest probability, $PMax$, for the tuples that have not been generated yet. We initialize $PMax$ to 1; after executing query Q_i and updating the list of answers (see third bullet), we decrease $PMax$ by $Pr(Q_i)$;
- The threshold th determining which answers are potentially in the top-k. We initialize th to 0; after executing Q_i and updating the answer list, we set th to the k-th largest p_{min} for tuples in the answer list;
- A list L of answers whose p_{max} is no less than th, and bounds p_{min} and p_{max} for each answer in L. After executing query Q_i, we update the list as follows: (1) for each $t \in L$ and $t \in Q_i(D_S)$, we increase $p_{min}(t)$ by $Pr(Q_i)$; (2) for each $t \in L$ but $t \notin Q_i(D_S)$, we decrease $p_{max}(t)$ by $Pr(Q_i)$; (3) if $PMax \geq th$, for each $t \notin L$ but $t \in Q_i(D_S)$, insert t to L, set p_{min} to $Pr(Q_i)$ and $p_{max}(t)$ to $PMax$.
- A list T of k tuples with top p_{min} values.

Step 3: When $th > PMax$ and for each $t \notin T$, $th > p_{max}(t)$, halt and return T.

Example 8 Consider Example 1 where we seek for top-1 answer. We answer the reformulated queries in order of Q_1, Q_2, Q_3. After answering Q_1, for tuple ("Sunnyvale") we have $p_{min} = .5$ and $p_{max} = 1$, and for tuple ("Mountain View") we have the same bounds. In addition, $PMax = .5$ and $th = .5$.

In the second round, we answer Q_2. Then, for tuple ("Sunnyvale") we have $p_{min} = .9$ and $p_{max} = 1$, and for tuple ("Mountain View") we have $p_{min} = .5$ and $p_{max} = .6$. Now $PMax = .1$ and $th = .9$.

Because $th > PMax$ and th is above the p_{max} for the ("Mountain View") tuple, we can halt and return ("Sunnyvale") as the top-1 answer. □

The next theorem states the correctness of ByTableTopK.

Theorem 7 *For any schema mapping $\overline{pM}$, SPJ query Q, instance D_S of the source schema of $\overline{pM}$, and integer k, Algorithm* ByTableTopK *correctly computes the top-k answers in $Q^{table}(D_S)$.* □

Our algorithm differs from previous top-k algorithms in the literature in two aspects. First, we execute the reformulated queries only when necessary, so we can return the top-k answers without executing all reformulated queries thereby leading to significant performance improvements. Fagin et al. [13] have proposed several algorithms for finding instances with top-k scores, where each instance has m attributes and the score of the instance is an aggregation over values of these m attributes. However, these algorithms assume for each attribute there exists a sorted list on its values, and they access the lists in parallel. In our context, this would require executing all reformulated queries upfront. Li et al. [24] have studied computing top-k answers for aggregation and group-by queries and optimizing query answering by generating the groups incrementally. Although we can also compute by-table answers using an aggregation query, this query is different from those considered in [24] in that the `WHERE` clause contains a set of sub-queries rather than database tables. Therefore, applying [24] here also requires evaluating all reformulated queries at the beginning.

Second, whereas maintaining upper bounds and lower bounds for instances has been explored in the literature, such as in Fagin's NRA (Non-Random Access) algorithm and in [24], our algorithm is different in that it keeps these bounds only for tuples that have already been generated by an executed reformulated query and that are potential top-k answers (by judging if the upper bound is above the threshold th).

5.2 By-tuple top-K query answering

We next consider returning top-k answers in by-tuple semantics. In general, we need to consider each mapping sequence and answer the query on the target instance that is consistent with the source and the mapping sequence. Algorithm TopKByTable can be modified to compute top-k by-tuple answers by deciding at runtime the mapping sequence to consider next. However, for non-p-join queries and projected-p-join queries, we can return top-k answers more efficiently. We outline our method for answering non-p-join queries here.

For non-p-join queries the probability of an answer tuple t to query Q cannot be expressed as a function of t's probabilities in executing reformulations of Q; rather, it is a function of t's probabilities in answering Q on each tuple database of the source table. However, retrieving answers on a tuple base is expensive. Algorithm NonPJoin provides a method that computes by-tuple answers on the tuple databases in a batch mode by first rewriting Q into Q' by returning the id column and then executing Q''s reformulated queries. We find top-k answers in a similar fashion. Here, after executing each reformulated query, we need to maintain two answer lists, one for Q and one for Q', and compute p_{min} and p_{max} for answers in different lists differently.

6 Representation of Probabilistic Mappings

Thus far, a p-mapping was represented by listing each of its possible mappings, and the complexity of query answering was polynomial in the size of that representation. Such a representation can be quite lengthy since it essentially enumerates a probability distribution by listing every combination of events in the probability space. Hence, an interesting question is whether there are more concise representations of p-mappings and whether our algorithms can leverage them.

We consider three representations that can reduce the size of the p-mapping exponentially. In Section 6.1 we consider a representation in which the attributes of the source and target tables are partitioned into groups and p-mappings are specified for each group separately. We show that query answering can be done in time polynomial in the size of the representation. In Section 6.2 we consider probabilistic correspondences, where we specify the marginal probability of each attribute correspondence. However, we show that such a representation can only be leveraged in limited cases. Finally, we consider Bayes Nets, the most common method for concisely representing probability distributions, in Section 6.3, and show that even though some p-mappings can be represented by them, query answering does not necessarily benefit from the representation.

6.1 Group probabilistic mapping

In practice, the uncertainty we have about a p-mapping can often be represented as a few localized choices, especially when schema mappings are created by semi-automatic methods. To represent such p-mappings more concisely, we can partition the source and target attributes and specify p-mappings for each partition.

Definition 12 (Group P-Mapping) An *n-group p-mapping gpM* is a triple $(S, T, \overline{pM})$, where

– S is a source relation schema and $S_1, \dots, S_n$ is a set of disjoint subsets of attributes in S;
– T is a target relation schema and $T_1, \dots, T_n$ is a set of disjoint subsets of attributes in T;
– $\overline{pM}$ is a set of p-mappings $\{pM_1, \dots, pM_n\}$, where for each $1 \le i \le n$, pM_i is a p-mapping between S_i and T_i. □

The semantics of an n-group p-mapping $gpM = (S, T, \overline{pM})$ is a p-mapping that includes the Cartesian product of the mappings in each of the pM_i's. The probability of the mapping composed of $m_1 \in pM_1, \dots, m_n \in pM_n$ is $\Pi_{i=1}^{n} Pr(m_i)$.

Example 9 Figure 5(a) shows p-mapping pM between the schemas $S(a, b, c)$ and $T(a', b', c')$. Figure 5(b) and (c) show two independent mappings that together form a 2-group p-mapping equivalent to pM. □

Note that a group p-mapping can be considerably more compact than an equivalent p-mapping. Specifically, if each pM_i includes l_i mappings, then a group p-mapping can describe $\Pi_{i=1}^{n} l_i$ possible mappings with $\sum_{i=1}^{n} l_i$ sub-mappings. The important feature of n-group p-mappings is that query answering can be done in time polynomial in their size.

Mapping	Prob
{(a,a'), (b,b'), (c,c')}	0.72
{(a,b'), (c,c')}	0.18
{(a,a'), (b,b')}	0.08
{(a,b')}	0.02

(a)

Mapping	Prob
{(a,a'), (b,b')}	0.8
{(a,b')}	0.2

(b)

Mapping	Prob
{(c,c')}	0.9
$\emptyset$	0.1

(c)

Fig. 5 Example 9: the p-mapping in (a) is equivalent to the 2-group p-mapping in (b) and (c).

Theorem 8 *Let $\overline{gpM}$ be a schema group p-mapping and let Q be an SPJ query. The mapping complexity of answering Q with respect to $\overline{gpM}$ in both by-table semantics and by-tuple semantics is in PTIME.* □

Note that as n grows, fewer p-mappings can be represented with n-group p-mappings. Formally, suppose we denote by $\mathcal{M}_{ST}^n$ the set of all n-group p-mappings between S and T, then:

Proposition 1 *For each $n \geq 1$, $\mathcal{M}_{ST}^{n+1} \subset \mathcal{M}_{ST}^n$.* □

We typically expect that when possible, a mapping would be given as a group p-mapping. The following theorem shows that we can find the best group p-mapping for a given p-mapping in polynomial time.

Proposition 2 *Given a p-mapping pM, we can find in polynomial time in the size of pM the maximal n and an n-group p-mapping gpM, such that gpM is equivalent to pM.* □

6.2 Probabilistic correspondences

The second representation we consider, *probabilistic correspondences*, represents a p-mapping with the marginal probabilities of attribute correspondences. This representation is the most compact one as its size is proportional to the product of the schema size of S and the schema size of T.

Definition 13 (Probabilistic Correspondences) A *probabilistic correspondence mapping (p-correspondence)* is a triple $pC = (S, T, \mathbf{c})$, where $S = \langle s_1, \ldots, s_m \rangle$ is a source relation schema, $T = \langle t_1, \ldots, t_n \rangle$ is a target relation schema, and

- $\mathbf{c}$ is a set $\{(c_{ij}, Pr(c_{ij})) | i \in [1, m], j \in [1, n]\}$, where $c_{ij} = (s_i, t_j)$ is an attribute correspondence, and $Pr(c_{ij}) \in [0, 1]$;
- for each $i \in [1, m], \sum_{j=1}^{n} Pr(c_{ij}) \leq 1$;
- for each $j \in [1, n], \sum_{i=1}^{m} Pr(c_{ij}) \leq 1$. □

Note that for a source attribute s_i, we allow

$$\sum_{j=1}^{n} Pr(c_{ij}) < 1.$$

Mapping	Prob
{(a,a'), (b,b'), (c,c')}	0.8
{(a,b'), (c,c')}	0.1
{(a,b')}	0.1

(a)

Corr	Prob
{(a,a')}	0.8
{(a,b')}	0.2
{(b,b')}	0.8
{(c,c')}	0.9

(b)

Fig. 6 Example 10: the p-mapping in (a) corresponds to the p-correspondence in (b).

This is because in some of the possible mappings, s_i may not be mapped to any target attribute. Similarly, for a target attribute t_j, we allow

$$\sum_{i=1}^{m} Pr(c_{ij}) < 1.$$

From each p-mapping, we can infer a p-correspondence by calculating the marginal probabilities of each attribute correspondence. Specifically, for a p-mapping $pM = (S, T, \mathbf{m})$, we denote by $pC(pM)$ the p-correspondence where each marginal probability is computed as follows:

$$Pr(c_{ij}) = \sum_{c_{ij} \in m, m \in \mathbf{m}} Pr(m)$$

However, as the following example shows, the relationship between p-mappings and p-correspondences is many-to-one.

Example 10 The p-correspondence in Figure 6(b) is the one computed for both the p-mapping in Figure 6(a) and the p-mapping in Figure 5(a). □

Given the many-to-one relationship, the question is when it is possible to compute the correct answer to a query based only on the p-correspondence. That is, we are looking for a class of queries $\bar{Q}$, called *p-mapping independent queries*, such that for every $Q \in \bar{Q}$ and every database instance D_S, if $pC(pM_1) = pC(pM_2)$, then the answer of Q with respect to pM_1 and D_S is the same as the answer of Q with respect to pM_2 and D_S. Unfortunately, this property holds for a very restricted class of queries, defined as follows:

Definition 14 (Single-Attribute Query) Let $pC = (S, T, \mathbf{c})$ be a p-correspondence. An SPJ query Q is said to be a *single-attribute* query with respect to pC if T has one single attribute occurring in the `SELECT` and `WHERE` clauses of Q. This attribute of T is said to be a *critical attribute.* □

Theorem 9 *Let $\overline{pC}$ be a schema p-correspondence, and Q be an SPJ query. Then, Q is p-mapping independent with respect to $\overline{pC}$ if and only if for each $pC \subseteq \overline{pC}$, Q is a single-attribute query with respect to pC.* □

Example 11 Continuing with Example 10, consider the p-correspondence pC in Figure 6(b) and the following two queries Q_1 and Q_2. Query Q_1 is mapping independent with respect to pC, but Q_2 is not.

```
Q1: SELECT T.a FROM T,U WHERE T.a=U.a'
Q2: SELECT T.a, T.c FROM T
```

□

Theorem 9 simplifies query answering for p-mapping independent queries. Wherever we needed to consider every possible mapping in previous algorithms, we consider only every attribute correspondence for the critical attribute.

Corollary 1 *Let $\overline{pC}$ be a schema p-correspondence, and Q be a p-mapping independent SPJ query with respect to $\overline{pC}$. The mapping complexity of answering Q with respect to $\overline{pC}$ in both by-table semantics and by-tuple semantics is in PTIME.* □

The result in Theorem 9 can be generalized to cases where we know the p-mapping is an n-group p-mapping. Specifically, as long as Q includes at most a single attribute in *each of the groups* in the n-group p-mapping, query answering can still be done with the correspondence mapping. We omit the details of this generalization.

6.3 Bayes nets

Bayes Nets are a powerful mechanism for concisely representing probability distributions and reasoning about probabilistic events [29]. The following example shows how Bayes Nets can be used in our context.

Example 12 Consider two schemas $S = (s_1, \ldots, s_n, s'_1, \ldots, s'_n)$ and $T = (t_1, \ldots, t_n)$. Consider the p-mapping $pM = (S, T, \mathbf{m})$, which describes the following probability distribution: if s_1 maps to t_1 then it is more likely that $\{s_2, \ldots, s_n\}$ maps to $\{t_2, \ldots, t_n\}$, whereas if s'_1 maps to t_1 then it is more likely that $\{s'_2, \ldots, s'_n\}$ maps to $\{t_2, \ldots, t_n\}$.

We can represent the p-mapping using a Bayes Net as follows. Let c be an integer constant. Then,

1. $Pr((s_1, t_1)) = Pr((s'_1, t_1)) = 1/2$;
2. for each $i \in [1, n]$, $Pr((s_i, t_i)|(s_1, t_1)) = 1 - \frac{1}{c}$ and $Pr((s'_i, t_i)|(s_1, t_1)) = \frac{1}{c}$;
3. for each $i \in [1, n]$, $Pr((s_i, t_i)|(s'_1, t_1)) = \frac{1}{c}$ and $Pr((s'_i, t_i)|(s'_1, t_1)) = 1 - \frac{1}{c}$.

Since the p-mapping contains 2^n possible mappings, the original representation would take space $O(2^n)$; however, the Bayes-Net representation takes only space $O(n)$. □

Although the Bayes-Net representation can reduce the size exponentially for some p-mappings, this conciseness may not help reduce the complexity of query answering. In Example 12, a query that returns all attributes in S will have 2^n answer tuples in by-table semantics and enumerating all these answers already takes exponential time in the size of pM's Bayes-Net representation.

7 Probabilistic Data Exchange

In this section we consider the use of probabilistic schema mappings in another common form of data integration, namely, data exchange. In doing so, we establish a close relationship between probabilistic mappings and probabilistic databases.

Unlike virtual data integration, in data exchange our goal is to create an instance of the target schema, given instances of the source schema. As discussed in previous work on data exchange [11], our goal is to create the *core universal solution*, which is an instance of the target schema that is minimal and from which we can derive all and only the certain answers to a query. In our context, we show that we can create a probabilistic database that serves as the core universal solution.

Probabilistic databases: We begin by briefly reviewing probabilistic databases (the reader is referred to [33] for further details).

A *probabilistic database (p-database)* pD over a schema $\bar{R}$ is a set $\{(D_1, Pr(D_1)), \ldots, (D_n, Pr(D_n))\}$, such that

– for $i \in [1, n]$, D_i is an instance of $\bar{R}$, and for every $i, j \in [1, n]$, $i \neq j \Rightarrow D_i \neq D_j$;
– $Pr(D_i) \in [0, 1]$ and $\sum_{i=1}^{n} Pr(D_i) = 1$.

Answers to queries over p-databases have probabilities associated with them. Specifically, let Q be a query over pD, and let t be a tuple. We denote by $\bar{D}(t)$ the subset of pD such that for each $D \in \bar{D}(t)$, $t \in Q(D)$. Let $p = \sum_{D \in \overline{D}(t)} Pr(D)$. If $p > 0$, we call (t, p) a *possible tuple in the answer of Q on pD*.

Given a query Q and a p-database pD, we denote by $Q(pD)$ the set of all possible tuples in the answer of Q on pD. We next show that data-exchange solutions can be represented as p-databases.

Data-exchange solutions: The data-exchange problem for a p-mapping $pM = (S, T, \mathbf{m})$ and an instance D_S of S is to find an instance of T that is consistent with D_S and pM. We distinguish between by-table solutions and by-tuple solutions.

Definition 15 (By-table Solution) Let $pM = (S, T, \mathbf{m})$ be a p-mapping and D_S be an instance of S.

A p-database $pD_T = \{(D_1, Pr(D_1)), \ldots, (D_n, Pr(D_n))\}$ is a *by-table solution for D_S under pM*, if for each $i \in [1, n]$, there exists a subset $\overline{m_i} \subseteq \mathbf{m}$, such that

– for each $m \in \overline{m_i}$, D_i is by-table consistent with D_S and m;
– $Pr(D_i) = \sum_{m \in \overline{m_i}} Pr(m)$;
– $\bar{m}_1, \ldots, \bar{m}_n$ form a partition of $\mathbf{m}$. □

Intuitively, for each possible mapping m, there should be a target instance that is consistent with the source instance and m, and the probability of the target instance should be the same as the probability of m. However, there can be a set of possible mappings $\bar{m}_i$ such that there exists a target instance, D_i, that is consistent with the source instance and each of the mapping in $\bar{m}_i$; hence, the probability of D_i should be the sum of the probabilities of the mappings in $\bar{m}_i$. Finally, the solution should have one and only one target for each possible mapping, so $\bar{m}_1, \ldots, \bar{m}_n$ should form a partition of the mappings in $\mathbf{m}$. In the definition for by-tuple semantics, the same intuition applies, except that we need to consider subsets of sequences.

Definition 16 (By-tuple Solution) Let $pM = (S, T, \mathbf{m})$ be a p-mapping and D_S be an instance of S with d tuples.

A p-database $pD_T = \{(D_1, Pr(D_1)), \ldots, (D_n, Pr(D_n))\}$ is a *by-tuple solution for D_S under pM* if for each $i \in [1, n]$, there exists a subset $\overline{seq_i} \subseteq \mathbf{seq}_d(pM)$, such that

– for each $seq \in \overline{seq_i}$, D_i is by-tuple consistent with D_S and seq;
– $Pr(D_i) = \sum_{seq \in \overline{seq_i}} Pr(seq)$;
– $\overline{seq_1}, \ldots, \overline{seq_n}$ form a partition of $\mathbf{seq}_d(pM)$. □

We illustrate by-table solutions and by-tuple solutions in the following example.

Example 13 Consider the p-mapping pM and the source instance D_S in Example 1 (repeated in Figure 7(a)(b)). Figure 7(c) shows a by-table solution for D_S under pM. Figure 7(d) and (e) show two by-tuple solutions for D_S under pM. Note that in (d), the first possible database is consistent with both sequence $< m_1, m_1 >$ and $< m_1, m_2 >$, so its probability is $0.5 * 0.5 + 0.5 * 0.4 = 0.45$. □

Core universal solution: Among all solutions, we would like to identify the *core universal solution*, because it is unique up to isomorphism and because we can use it to find all the answers to a query. We define the core universal solution for p-databases, but first we need to define *homomorphism* and *isomorphism* on such databases.

The definition of homomorphism on p-databases is an extension of homomorphism on traditional databases, which we review now. Let $\mathcal{C}$ be the set of all constant values that occur in source instances, called *constants*, and let $\mathcal{V}$ be an infinite set of variables, called *labeled nulls*. $\mathcal{C} \cap \mathcal{V} = \emptyset$. Let D be a database instance. We denote by $V(D) \subseteq \mathcal{V}$ the set of labeled nulls occurring in D.

Definition 17 (Instance Homomorphism) Let D_R and D'_R be two instances of schema R with values in $\mathcal{C} \cup \mathcal{V}$.

A *homomorphism* $h : D_R \to D'_R$ is a mapping from $\mathcal{C} \cup V(D_R)$ to $\mathcal{C} \cup V(D'_R)$ such that

– $h(c) = c$ for every $c \in \mathcal{C}$;
– for every tuple $t = (v_1, \ldots, v_n)$ in D_R, we have that $h(t) = (h(v_1), \ldots, h(v_n))$ is in D'_R. □

We next extend the definition of homomorphism for traditional databases to homomorphism for p-databases. Consider two p-databases pD and pD'. Intuitively, for pD to be homomorphic to pD', each possible database in pD should be homomorphic to some possible database in pD'. However, one possible database in pD can be homomorphic to several possible databases in pD'. We thus partition the databases in pD' and each database in pD should be homomorphic to the databases in one partition of pD'. We note that it can also happen that multiple databases in pD are homomorphic to the same possible database in pD'. Our definition requires that each database in pD is homomorphic to at least one distinct database in pD' and so for pD to be homomorphic to pD', the number of databases in pD should be no more than that in pD'. As we will see in the definition of *core universal solution*, with our definition of homomorphism, the core universal solution would be the solution with the least number of possible databases.

Definition 18 (Homomorphism of P-Databases) Let $pD = \{(D_i, Pr(D_i)) \mid i \in [1, n]\}$ and $pD' = \{(D'_i, Pr(D'_i)) \mid i \in [1, l]\}$ be two p-databases of the same schema. Let $\mathcal{P}(pD')$ be the powerset of the possible databases in pD'.

A *homomorphism* $h : pD \to pD'$ is a mapping from pD to $\mathcal{P}(pD')$, such that

– for every $D \in pD$ and $D' \in h(D)$, there exists a homomorphism $g : D \to D'$;
– for every $D \in pD$, $Pr(D) = \sum_{D' \in h(D)} Pr(D')$;
– $h(D_1), \ldots, h(D_n)$ form a partition of pD'. □

According to this definition, in Figure 7, p-database pD_2 is homomorphic to pD_3, but the homomorphism in the opposite direction does not hold.

We next define isomorphism for p-databases, where we require one-to-one mappings between possible databases.

Definition 19 (Isomorphism of P-Databases) Let $pD = \{(D_i, Pr(D_i)) \mid i \in [1, n]\}$ and $pD' = \{(D'_i, Pr(D'_i)) \mid i \in [1, m]\}$ be two p-databases of the same schema.

An *isomorphism* $i : pD \to pD'$ is a bijective mapping from pD to pD', such that if $h(D) = D'$,

	Possible Mapping	Prob
$m_1 =$	{(pname, name), (email-addr, email), (current-addr, mailing-addr), (permanent-addr, home-addr)}	0.5
$m_2 =$	{(pname, name), (email-addr, email), (permanent-addr, mailing-addr), (current-addr, home-addr)}	0.4
$m_3 =$	{(pname, name), (email-addr, mailing-addr), (current-addr, home-addr)}	0.1

(a)

pname	email-addr	current-addr	permanent-addr
Alice	alice@	Mountain View	Sunnyvale
Bob	bob@	Sunnyvale	Sunnyvale

(b)

name	email	mailing-addr	home-addr	office-addr
Alice	alice@	Mountain View	Sunnyvale	*O1*
Bob	bob@	Sunnyvale	Sunnyvale	*O2*

p=0.5

name	email	mailing-addr	home-addr	office-addr
Alice	alice@	Sunnyvale	Mountain View	*O1*
Bob	bob@	Sunnyvale	Sunnyvale	*O2*

p=0.4

name	email	mailing-addr	home-addr	office-addr
Alice	*E1*	alice@	Mountain View	*O1*
Bob	*E2*	bob@	Sunnyvale	*O2*

p=0.1

(c)

name	email	mailing-addr	home-addr	office-addr
Alice	alice@	Mountain View	Sunnyvale	*O1*
Bob	bob@	Sunnyvale	Sunnyvale	*O2*

p=0.45

name	email	mailing-addr	home-addr	office-addr
Alice	alice@	Mountain View	Sunnyvale	*O1*
Bob	*E2*	bob@	Sunnyvale	*O2*

p=0.05

name	email	mailing-addr	home-addr	office-addr
Alice	alice@	Sunnyvale	Mountain View	*O1*
Bob	bob@	Sunnyvale	Sunnyvale	*O2*

p=0.36

name	email	mailing-addr	home-addr	office-addr
Alice	alice@	Sunnyvale	Mountain View	*O1*
Bob	*E2*	bob@	Sunnyvale	*O2*

p=0.04

name	email	mailing-addr	home-addr	office-addr
Alice	*E1*	alice@	Mountain View	*O1*
Bob	bob@	Sunnyvale	Sunnyvale	*O2*

p=0.09

name	email	mailing-addr	home-addr	office-addr
Alice	*E1*	alice@	Mountain View	*O1*
Bob	*E2*	bob@	Sunnyvale	*O2*

p=0.01

(d)

name	email	mailing-addr	home-addr	office-addr
Alice	alice@	Mountain View	Sunnyvale	*O1*
Bob	bob@	Sunnyvale	Sunnyvale	*O2*

p=0.25

name	email	mailing-addr	home-addr	office-addr
Alice	alice@	Mountain View	Sunnyvale	*O1*
Bob	bob@	Sunnyvale	Sunnyvale	*O2*

p=0.20

name	email	mailing-addr	home-addr	office-addr
Alice	alice@	Mountain View	Sunnyvale	*O1*
Bob	*E2*	bob@	Sunnyvale	*O2*

p=0.05

name	email	mailing-addr	home-addr	office-addr
Alice	alice@	Sunnyvale	Mountain View	*O1*
Bob	bob@	Sunnyvale	Sunnyvale	*O2*

p=0.20

name	email	mailing-addr	home-addr	office-addr
Alice	alice@	Sunnyvale	Mountain View	*O1*
Bob	bob@	Sunnyvale	Sunnyvale	*O2*

p=0.16

name	email	mailing-addr	home-addr	office-addr
Alice	alice@	Sunnyvale	Mountain View	*O1*
Bob	*E2*	bob@	Sunnyvale	*O2*

p=0.04

name	email	mailing-addr	home-addr	office-addr
Alice	*E1*	alice@	Mountain View	*O1*
Bob	bob@	Sunnyvale	Sunnyvale	*O2*

p=0.05

name	email	mailing-addr	home-addr	office-addr
Alice	*E1*	alice@	Mountain View	*O1*
Bob	bob@	Sunnyvale	Sunnyvale	*O2*

p=0.04

name	email	mailing-addr	home-addr	office-addr
Alice	*E1*	alice@	Mountain View	*O1*
Bob	*E2*	bob@	Sunnyvale	*O2*

p=0.01

(e)

Fig. 7 The running example: (a) a probabilistic schema mapping between S and T; (b) a source instance D_S; (c) a by-table solution pD_1 for D_S under pM; (d) a by-tuple solution pD_2 for D_S under pM; (e) another by-tuple solution pD_3 for D_S under pM. In (c)(d)(e), *O1, O2, E1,* and *E2* are labeled nulls.

– there exists an isomorphism $g : D \to D'$;
– $Pr(D) = Pr(D')$. □

We can now define core universal solutions.

Definition 20 (Core Universal Solution) Let $pM = (S, T, \mathbf{m})$ be a p-mapping and D_S be an instance of S.

A p-database instance pD_T of T is called a *by-table (resp. by-tuple) universal solution for* D_S *under* pM, if (1) pD_T is a by-table (resp. by-tuple) solution for D_S, and (2) for every by-table (resp. by-tuple) solution pD'_T for D_S, there exists a homomorphism $h : pD_T \to pD'_T$.

Further, pD_T is called a *by-table (resp. by-tuple) core universal solution for* D_S if for each possible database $D_T \in pD_T$, there is no homomorphism from D_T to a proper subset of tuples in D_T. □

Intuitively, a core universal solution is the *smallest* and *most general* solution. In Example 13, pD_1 is the core universal solution in by-table semantics and pD_2 is the core universal solution in by-tuple semantics.

The following theorem establishes the key properties of core universal solutions in our context.

Theorem 10 *Let $pM = (S, T, \mathbf{m})$ be a p-mapping and D_S be an instance of S.*

1. *There is a unique by-table core universal solution and a unique by-tuple core universal solution up to isomorphism for D_S with respect to pM.*
2. *Let Q be a conjunctive query over T. We denote by $Q(pD)$ the results of answering Q on pD and discarding all answer tuples containing* null *values (labeled nulls). Then,*

   $$Q^{table}(D_S) = Q(pD_T^{table}).$$

   Similarly, let pD_T^{tuple} be the by-tuple core universal solution for D_S under pM. Then,

   $$Q^{tuple}(D_S) = Q(pD_T^{tuple}). \qquad \square$$

Complexity of data exchange: Recall that query answering is in PTIME in by-table semantics, and in #P in by-tuple semantics in general. However, data exchange in both semantics is in PTIME in the size of the data and in the size of the mapping. The complexity of computing the core universal solution is established by the following theorem:

Theorem 11 *Let $pM = (S, T, \mathbf{m})$ be a p-mapping and D_S be an instance of S.*

Generating the by-table or by-tuple core universal solution for D_S under pM takes polynomial time in the size of the data and the mapping. $\square$

For by-table semantics the proof is rather straightforward. For by-tuple semantics the proof requires a special representation of p-databases, called *disjunctive p-database.*

Definition 21 (Disjunctive P-Database) Let R be a relation schema where there exists a set of attributes that together form the *key* of the relation. Let $pD_R^\vee$ be a set of tuples of R, each attached with a probability.

We say that $pD_R^\vee$ is a *disjunctive p-database* if for each key value that occurs in $pD_R^\vee$, the probabilities of the tuples with this key value sum up to 1. $\square$

In a disjunctive p-database, we consider tuples with the same key value as disjoint and those with different key values as independent. Formally, let $key_1, \ldots, key_n$ be the set of all distinct key values in $pD_R^\vee$. For each $i \in [1, n]$, we denote by d_i the number of tuples whose key value is key_i. Then, with a set of $\Sigma_{i=1}^n d_i$ tuples, $pD_R^\vee$ can define a set of $\Pi_{i=1}^n d_i$ possible databases, where each possible database $(D, Pr(D))$ contains n tuples $t_1, \ldots, t_n$, such that (1) for each $i \in [1, n]$, the key value of t_i is key_i; and (2) $Pr(D) = \Pi_{i=1}^n Pr(t_i)$. Figure 8 shows the disjunctive p-database that is equivalent to pD_2 in Figure 7(d).

Theorem 11 is based on the following lemma.

Lemma 4 *Let $pM = (S, T, \mathbf{m})$ be a p-mapping and D_S be an instance of S.*

The by-tuple core universal solution for D_S under pM can be represented as a disjunctive p-database. $\square$

The complexity of answering queries over the core universal solutions is the same as that of the corresponding results for probabilistic databases. Specifically, the following theorem follows from [31].

Theorem 12 *Let Q be a conjunctive query.*

- *Let pD be a p-database instance. Computing $Q(pD)$ is in PTIME in the size of the data.*
- *Let $pD^\vee$ be a disjunctive p-database instance. Computing $Q(pD^\vee)$ is #P-complete in the size of the data.* $\square$

Finally, we note that when the p-mapping is a group p-mapping, we can compute the core universal solution in time that is polynomial in the size of the data and in the size of the group p-mapping by representing the solution as a set of p-databases.

GLAV mappings: The complexity results for data exchange under our limited form of p-mappings carry over to GLAV mappings. For by-table semantics, generating the core universal solution takes polynomial time; for by-tuple semantics, defining the core universal solution is tricky and we leave it for future work.

Theorem 13 *Let pGM be a GLAV p-mapping between a source schema $\bar{S}$ and a target schema $\bar{T}$. Let D_S be an instance of $\bar{S}$.*

Generating the by-table core universal solution for D_S under pM takes polynomial time in the size of the data and the mapping. $\square$

8 Composition and Inversion

Composition and inversion of mappings have received significant attention recently [5, 10, 12, 26] because they are fundamental operations on mappings and they are important for data exchange, integration and peer data management. In this section, we study composition and inversion of probabilistic mappings. We show that probabilistic mappings are closed under composition but not under inversion, and we can compose two p-mappings in polynomial time.

Composition: Intuitively, composing two p-mappings derives a p-mapping between the source schema of the first p-mapping and the target schema of the second p-mapping, such that the composition p-mapping has the same effect as applying the two p-mappings successively. We formally define mapping compositions as follows.

*k*ey	*n*ame	*e*mail	*m*ailing-addr	*h*ome-addr	*o*ffice-addr	*p*robability
1	Alice	alice@	Mountain View	Sunnyvale	*O1*	0.5
1	Alice	alice@	Sunnyvale	Mountain View	*O1*	0.4
1	Alice	*E1*	alice@	Mountain View	*O1*	0.1
2	Bob	bob@	Sunnyvale	Sunnyvale	*O2*	0.9
2	Bob	*E2*	bob@	Sunnyvale	*O2*	0.1

Fig. 8 Disjunctive P-database that is equivalent to pD_2 in Figure 7(d).

Definition 22 (Composition of P-Mappings) Let $pM_1 = (R, S, \mathbf{m_1})$ and $pM_2 = (S, T, \mathbf{m_2})$ be two p-mappings.

We call $pM = (R, T, \mathbf{m})$ a *by-table (resp. by-tuple) composition of* pM_1 *and* pM_2, denoted by $pM = pM_1 \circ pM_2$, if for each D_R of R and D_T of T, D_T is by-table (resp. by-tuple) consistent with D_R with probability p under pM, if and only if there exists a set of possible databases $\bar{D}_S$ of S, such that

– for each $D \in \bar{D}_S$, D is by-table (resp. by-tuple) consistent with D_R with probability $p_1(D)$ under pM_1;
– for each $D \in \bar{D}_S$, D_T is by-table (resp. by-tuple) consistent with D with probability $p_2(D)$ under pM_2;
– $p = \sum_{D \in \bar{D}_S} p_1(D) \cdot p_2(D)$. □

When we have two p-mappings and need to apply them successively, a natural thought is to compute their composition and apply the result mapping directly. Indeed, the following theorem shows that for any two p-mappings, there is a unique composition p-mapping and we can generate it in polynomial time. Thus, the above strategy is feasible and efficient.

Theorem 14 *Let* $pM_1 = (R, S, \mathbf{m_1})$ *and* $pM_2 = (S, T, \mathbf{m_2})$ *be two p-mappings. Between* R *and* T *there exists a unique p-mapping,* pM*, that is the composition of* pM_1 *and* pM_2 *in both by-table and by-tuple semantics and we can generate* pM *in polynomial time.* □

Whereas probabilistic mappings in general are closed under composition, the following theorem shows that n-group p-mappings are not closed under composition when $n > 1$.

Theorem 15 *N-group* ($n > 1$) *p-mappings are not closed under mapping composition.* □

Inversion: The intuition for inverse mappings is as follows: if we compose a p-mapping, pM, and its inverse mapping, we obtain an *identity mapping*, which deterministically maps each attribute to itself. Given a schema R, we denote the identity p-mapping for R as $IM(R)$.

Definition 23 (Inversion of P-Mapping) Let $pM = (S, T, \mathbf{m})$ be a p-mapping. We say $pM' = (T, S, \mathbf{m}')$ is an *inverse of* pM_{ST}, if $pM \circ pM' = IM(S)$. □

Note that our definition of inversion corresponds to *global inverse* in [10], which can be applied to the class of all source instances. In [10] Fagin shows that for a traditional deterministic mapping to have a global inverse, it needs to satisfy the *unique solutions property*; that is, no two distinct source instances have the same set of solutions. In our context, as shown in the following theorem, only p-mappings in a very limited form have inverse p-mappings but the vast majority of p-mappings as illustrated in this paper do not have inverse p-mappings.

Theorem 16 *Let* $pM = (S, T, \mathbf{m})$ *be a p-mapping. Then,* pM *has an inverse p-mapping if and only if*

– $\mathbf{m}$ *contains a single possible mapping* $(m, 1)$;
– *each attribute in* S *is involved in an attribute correspondence in* m. □

9 Broader Classes of Mappings

In this section we briefly show how our results can be extended to capture three common practical extensions to our mapping language.

Complex mappings: Complex mappings map a set of attributes in the source to a set of attributes in the target. For example, we can map the attribute address to the concatenation of street, city, and state.

Formally, a *set correspondence* between S and T is a relationship between a subset of attributes in S and a subset of attributes in T. Here, the function associated with the relationship specifies a single value for each of the target attributes given a value for each of the source attributes. Again, the actual functions are irrelevant to our discussion. A *complex mapping* is a triple (S, T, cm), where cm is a set of set correspondences, such that each attribute in S or T is involved in at most one set correspondence. A *complex p-mapping* is of the form $pCM = \{(cm_i, Pr(cm_i)) \mid i \in [1, n]\}$, where $\sum_{i=1}^{n} Pr(cm_i) = 1$.

Theorem 17 *Let* $\overline{pCM}$ *be a complex schema p-mapping between schemas* $\bar{S}$ *and* $\bar{T}$*. Let* D_S *be an instance of* $\bar{S}$*.*

1. *Let* Q *be an SPJ query over* $\bar{T}$*. The data complexity and mapping complexity of computing* $Q^{table}(D_S)$ *with respect to* $\overline{pCM}$ *are PTIME. The data complexity of computing* $Q^{tuple}(D_S)$ *with respect to* $\overline{pCM}$ *is #P-complete. The mapping complexity of computing* $Q^{tuple}(D_S)$ *with respect to* $\overline{pCM}$ *is in PTIME.*

2. *Generating the by-table or by-tuple core universal solution for D_S under $\overline{pCM}$ takes polynomial time in the size of the data and the mapping.* □

Union mapping: *Union mappings* specify relationships such as both attribute home-address and attribute office-address can be mapped to address. Formally, a *union mapping* is a triple $(S, T, \bar{m})$, where $\bar{m}$ is a set of mappings between S and T. Given a source relation D_S and a target relation D_T, we say D_S and D_T are consistent with respect to the union mapping if for each source tuple t and $m \in \bar{m}$, there exists a target tuple t', such that t and t' satisfy m. A *union p-mapping* is of the form $pUM = \{(\bar{m}_i, Pr(\bar{m}_i)) \mid i \in [1, n]\}$, where $\sum_{i=1}^{n} Pr(\bar{m}_i) = 1$.

The results in this paper carry over, except that for by-tuple data exchange, we need a new representation for the core universal solution.

Theorem 18 *Let $\overline{pUM}$ be a union schema p-mapping between a source schema $\bar{S}$ and a target schema $\bar{T}$. Let D_S be an instance of $\bar{S}$.*

1. *Let Q be a conjunctive query over $\bar{T}$. The problem of computing $Q^{table}(D_S)$ with respect to $\overline{pUM}$ is in PTIME in the size of the data and the mapping; the problem of computing $Q^{tuple}(D_S)$ with respect to $\overline{pUM}$ is in PTIME in the size of the mapping and #P-complete in the size of the data.*
2. *Generating the by-table or by-tuple core universal solution for D_S under $\overline{pUM}$ takes polynomial time in the size of the data and the mapping.* □

Conditional mappings: In practice, our uncertainty is often conditioned. For example, we may want to state that daytime-phone maps to work-phone with probability 60% if age ≤ 65, and maps to home-phone with probability 90% if age > 65.

We define a *conditional p-mapping* as a set $cpM = \{(pM_1, C_1), \ldots, (pM_n, C_n)\}$, where $pM_1, \ldots, pM_n$ are p-mappings, and $C_1, \ldots, C_n$ are pairwise disjoint conditions. Intuitively, for each $i \in [1, n]$, pM_i describes the probability distribution of possible mappings when condition C_i holds. Conditional mappings make more sense for by-tuple semantics. The following theorem shows that our results carry over to such mappings.

Theorem 19 *Let $\overline{cpM}$ be a conditional schema p-mapping between $\bar{S}$ and $\bar{T}$. Let D_S be an instance of $\bar{S}$.*

1. *Let Q be an SPJ query over $\bar{T}$. The problem of computing $Q^{tuple}(D_S)$ with respect to $\overline{cpM}$ is in PTIME in the size of the mapping and #P-complete in the size of the data.*
2. *Generating the by-tuple core universal solution for D_S under $\overline{cpM}$ takes linear time in the size of the data and the mapping.* □

10 Related Work

We are not aware of any previous work studying the semantics and properties of probabilistic schema mappings. Florescu et al. [14] were the first to advocate the use of probabilities in data integration. Their work used probabilities to model (1) a mediated schema with overlapping classes (*e.g.*, DatabasePapers and AIPapers), (2) source descriptions stating the probability of a tuple being present in a source, and (3) overlap between data sources. While these are important aspects of many domains and should be incorporated into a data integration system, our focus here is different. Magnani and Montesi [27] have empirically shown that top-k schema mappings can be used to increase the recall of a data integration process and Gal [15] described how to generate top-k schema matchings by combining the matching results generated by various matchers. The probabilistic schema mappings we propose are different as it contains all possible schema mappings and has probabilities on these mappings to reflect the likelihood that each mapping is correct. Nottelmann and Straccia [28] proposed generating probabilistic schema matchings that capture the uncertainty in each matching step. The probabilistic schema mappings we consider in addition takes into consideration various combinations of attribute correspondences and describe a *distribution* of possible schema mappings where the probabilities of all mappings sum up to 1. Finally, De Rougement and Vieilleribiere [7] considered approximate data exchange in that they relaxed the constraints on the target schema, which is a different approach from ours.

There has been a flurry of activity around probabilistic and uncertain databases lately [4,33,6,3]. Our intention is that a data integration system will be based on a probabilistic data model, and we leverage concepts from that work as much as possible. We also believe that uncertainty and lineage are closely related, in the spirit of [4], and that relationship will play a key role in data integration. We leave exploring this topic to future work.

11 Conclusions and future work

We introduced probabilistic schema mappings, which are a key component of data integration systems that handle uncertainty. In particular, probabilistic schema mappings enable us to answer queries on heterogeneous data sources even if we have only a set of candidate mappings that may not be precise. We identified two possible semantics for such mappings, by-table and by-tuple, and presented query answering algorithms and computational complexity for both semantics. We also considered concise encoding of probabilistic mappings, with which we are able to improve the efficiency of query answering. Finally, we studied the application of probabilistic schema mappings in the context of data exchange and extended our definition to more powerful schema mapping languages to show the extensibility of our approach.

We are currently working on several extensions to this work. First, we have built a system that automatically creates a mediated schema from a set of given data sources. As an intermediate step in doing so, we create probabilistic schema mappings between the data sources and several candidate mediated schemas. We use these mappings to choose a mediated schema that appears to be the best fit.

Second, to employ probabilistic mappings in resolving heterogeneity at the schema level, we must have a good method of generating probabilities for the mappings. This is possible as techniques for semi-automatic schema mapping are often based on Machine Learning techniques that at their core compute the confidence of correspondences they generate. However, such confidence is meant more as a ranking mechanism than true probabilities between candidates and is associated with attribute correspondences rather than candidate mappings. We plan to study how to generate from them probabilities for candidate mappings.

Third, we would like to reason about the uncertainty in schema mappings in order to improve the schema mappings. Specifically, by analyzing the probabilities of the candidate mappings, we would like to find the critical parts (*i.e.*, attribute correspondences) where it is most beneficial to expand more resources (human or otherwise) to improve schema mapping.

Finally, we would like to extend our current results to probabilistic data and probabilistic queries and build a full-fledged data integration system that can handle uncertainty at various levels. Studying the theoretical underpinning of probabilistic mappings is the first step towards building such a system. In addition, we need to extend the current work in the community on probabilistic databases [33] to study how to efficiently answer queries in the presence of uncertainties in schemas and in data, and study how to translate a keyword query into structured queries by exploiting evidence obtained from the existing data and users' search and querying patterns.

References

1. S. Abiteboul and O. Duschka. Complexity of answering queries using materialized views. In *PODS*, 1998.
2. S. Agrawal, S. Chaudhuri, and G. Das. DBXplorer: A system for keyword-based search over relational databases. In *ICDE*, 2002.
3. L. Antova, C. Koch, and D. Olteanu. World-set decompositions: Expressiveness and efficient algorithms. In *ICDT*, 2007.
4. O. Benjelloun, A. D. Sarma, A. Y. Halevy, and J. Widom. ULDBs: Databases with uncertainty and lineage. In *VLDB*, 2006.
5. P. A. Bernstein, T. J. Green, S. Melnik, and A. Nash. Implementing mapping composition. In *Proc. of VLDB*, pages 55–66, 2006.
6. R. Cheng, S. Prabhakar, and D. V. Kalashnikov. Querying imprecise data in moving object environments. In *ICDE*, 2003.
7. M. de Rougemont and A. Vieilleribiere. Approximate data exchange. In *ICDT*, 2007.
8. X. Dong and A. Halevy. A platform for personal information management and integration. In *CIDR*, 2005.
9. X. L. Dong, A. Y. Halevy, and C. Yu. Data integration with uncertainty. In *Proc. of VLDB*, 2007.
10. R. Fagin. Inverting schema mappings. In *Proc. of PODS*, 2006.
11. R. Fagin, P. G. Kolaitis, and L. Popa. Data exchange: Getting to the core. *ACM Transactions on Database Systems*, 30(1):174–201, 2005.
12. R. Fagin, P. G. Kolaitis, L. Popa, and W. C. Tan. Composing schema mappings: Second-order dependencies to the rescue. *ACM Transactions on Database Systems*, 30(4):994–1055, 2005.
13. R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. In *PODS*, 2001.
14. D. Florescu, D. Koller, and A. Levy. Using probabilistic information in data integration. In *Proc. of VLDB*, 1997.
15. A. Gal. Why is schema matching tough and what can we do about it? *SIGMOD Record*, 35(4):2–5, 2007.
16. GoogleBase. http://base.google.com/, 2005.
17. A. Y. Halevy. Answering queries using views: A survey. *VLDB Journal*, 10(4), 2001.
18. A. Y. Halevy, N. Ashish, D. Bitton, M. J. Carey, D. Draper, J. Pollock, A. Rosenthal, and V. Sikka. Enterprise information integration: successes, challenges and controversies. In *SIGMOD*, 2005.
19. A. Y. Halevy, M. J. Franklin, and D. Maier. Principles of dataspace systems. In *PODS*, 2006.
20. A. Y. Halevy, A. Rajaraman, and J. J. Ordille. Data integration: The teenage years. In *VLDB*, 2006.
21. V. Hristidis and Y. Papakonstantinou. DISCOVER: Keyword search in relational databases. In *VLDB*, 2002.
22. M. Lenzerini. Data integration: A theoretical perspective. In *Proc. of PODS*, 2002.
23. A. Y. Levy. Special issue on adaptive query processing. *IEEE Data Eng. Bull.*, 23(2), 2000.
24. C. Li, K. C.-C. Chang, and I. F. LLyas. Supporting adhoc ranking aggregates. In *SIGMOD*, 2006.
25. J. Madhavan, S. Cohen, X. Dong, A. Halevy, S. Jeffery, D. Ko, and C. Yu. Navigating the seas of structured web data. In *CIDR*, 2007.
26. J. Madhavan and A. Halevy. Composing mappings among data sources. In *Proc. of VLDB*, 2003.
27. M. Magnani and D. Montesi. Uncertainty in data integration: current approaches and open problems. In *VLDB workshop on Management of Uncertain Data*, pages 18–32, 2007.
28. H. Nottelmann and U. Straccia. Information retrieval and machine learning for probabilistic schema matching. *Information Processing and Management*, 43(3):552–576, 2007.
29. J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* Morgan Kaufmann Publishers, Inc., California, 1988.
30. E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. *VLDB Journal*, 10(4):334–350, 2001.
31. C. Re, D. Suciu, and N. N. Dalvi. Query evaluation on probabilistic databases. *IEEE Data Eng. Bull.*, 29(1), 2006.
32. A. D. Sarma, X. L. Dong, and A. Y. Halevy. Bootstrapping pay-as-you-go data integration systems. In *Proc. of SIGMOD*, 2008.
33. D. Suciu and N. N. Dalvi. Foundations of probabilistic answers to queries. In *SIGMOD*, 2005.

12 Appendix: Proofs

Theorem 1. Let $\overline{pM}$ be a schema p-mapping and let Q be an SPJ query. Answering Q with respect to $\overline{pM}$ in by-table semantics is in PTIME in the size of the data and the mapping. □

Proof It is trivial that Algorithm ByTable computes all by-table answers. We now consider its time complexity by examining the time complexity of each step.
Step 1: Assume for each target relation $T_i, i \in [1, l]$, the involved p-mapping contains n_i possible mappings. Then, the number of reformulated queries is $\Pi_{i=1}^{l} n_i$, polynomial in the size of the mapping.

Given the restricted class of mappings we consider, we can reformulate the query as follows. For each of T_i's attributes t, if there exists an attribute correspondence $(S.s, T.t)$ in m^i, we replace t everywhere with s; otherwise, the reformulated query returns an empty result. Let $|Q|$ be the size of Q. Thus, reformulating a query takes time $O(|Q|)$, and the size of the reformulated query does not exceed the size of Q.

Therefore, Step 1 takes time $O(\Pi_{i=1}^{l} n_i \cdot |Q|)$, which is polynomial in the size of the p-mapping and does not depend on the size of the data.

Step 2: Answering each reformulated query takes polynomial time in the size of the data and the number of answer tuples is polynomial in the size of the data. Because there is a polynomial number of answer tuples and each occurs in the answers of no more than $\Pi_{i=1}^{l} n_i$ queries, summing up the probabilities for each answer tuple takes time $O(\Pi_{i=1}^{l} n_i)$. Thus, Step 2 takes polynomial time in the size of the mapping and the data. □

Theorem 2. Let pGM be a general p-mapping between a source schema $\bar{S}$ and a target schema $\bar{T}$. Let D_S be an instance of $\bar{S}$. Let Q be an SPJ query with only equality conditions over $\bar{T}$.

The problem of computing $Q^{table}(D_S)$ with respect to pGM is in PTIME in the size of the data and the mapping. □

Proof We proceed in two steps to return all by-table answers. In the first step, for each $gm_i, i \in [1, n]$, we answer Q according to gm_i on D_S. The certain answer with regard to gm_i has probability $Pr(gm_i)$. SPJ queries with only equality conditions are conjunctive queries. According to [1], we can return all certain answers in polynomial time in the size of the data, and the number of certain answers is polynomial in the size of the data. Thus, the first step takes polynomial time in the size of the data and the mapping.

In the second step, we sum up the probabilities of each answer tuple. Because there are a polynomial number of answer tuples and each occurs in the answers of no more than n reformulated queries, this step takes polynomial time in the size of the data and the mapping. □

Lemma 1. Let $\overline{pM}$ be a schema p-mapping. Let Q be an SPJ query and Q_m be Q's mirror query with respect to $\overline{pM}$. Let D_S be the source database and D_T be the mirror target of D_S with respect to $\overline{pM}$.

Then, $t \in Q^{tuple}(D_S)$ if and only if $t \in Q_m(D_T)$ and t does not contain null value. □

Proof *If:* We prove $t \in Q^{tuple}(D_S)$ by showing that we can construct a mapping sequence *seq* such that for each target instance D'_T that is consistent with D_S and *seq*, $t \in Q(D'_T)$.

Assume query Q (and so Q_m) contains n subgoals (*i.e.*, occurrences of tables in the `FROM` clause). Assume we obtain t by joining n tuples $t_1, \ldots, t_n \in D_T$, each in the relation of a subgoal. Consider a relation R that occurs in Q. Assume $t_{k_1}, \ldots, t_{k_l}, (k_1, \ldots, k_l \in [1, n])$ are tuples of R (for different subgoals). Let $pM \in \overline{pM}$ be the p-mapping where R is the target and let S be the source relation of pM. For each $j \in [1, l]$, we denote the id value of t_{k_j} by t_{k_j}.id, and the mapping value of t_{k_j} by t_{k_j}.mapping. Then, t_{k_j} is consistent with the t_{k_j}.id-th source tuple in S and the mapping t_{k_j}.mapping.

We construct the mapping sequence of R for *seq* as follows: (1) for each $j \in [1, l]$, the mapping for the t_{k_j}.id-th tuple is t_{k_j}.mapping; (2) the rest of the mappings are arbitrary mappings in pM. To ensure the construction is valid, we need to prove that all tuples with the same id value have the same mapping value. Indeed, for every $j, h \in [1, l], j \neq h$, because t_{k_j} and t_{k_h} satisfy the predicate (R_1.id <> R_2.id `OR` R_1.mapping=R_2.mapping) in Q_m, if t_{k_j}.id=t_{k_h}.id then t_{k_j}.mapping=t_{k_h}.mapping.

We now prove for each target instance D'_T that is consistent with D_S and *seq*, $t \in Q(D'_T)$. For each $t_i, i \in [1, n]$, we denote by t'_i the tuple in D'_T that is consistent with the t_i.id-th source tuple and the t_i.mapping mapping. We denote by $R(t_i), i \in [1, n]$, the subgoal that t_i belongs to. By the definition of mirror target and also because t does not contain null value, for each attribute of $R(t_i)$ that is involved in Q, t_i has non-null value, and so they are involved in the mapping t_i.mapping. Thus, t'_i has the same value for these attributes. So t can be obtained by joining $t'_1, \ldots, t'_n$ and $t \in Q(D'_T)$.

Only if: $t \in Q^{tuple}(D_S)$, so there exists a mapping sequence *seq*, such that for each D'_T that is consistent with D_S and *seq*, $t \in Q(D'_T)$. Consider such a D'_T. Assume t is obtained by joining tuples $t_1, \ldots, t_n \in D'_T$, and for each $i \in [1, n]$, t_i is a tuple of subgoal R_i. Assume t_i is consistent with source tuple s_i and m_i. We denote by t'_i the instance in D_T whose id value refers to s_i and mapping value refers to m_i. Let $\bar{A}_i$ be the set of attributes of the subgoal R_i that are involved in the query. Since t is a "certain answer", all attributes in $\bar{A}_i$ must be involved in m_i. Thus, t_i and t'_i have the same value for these attributes, and all predicates in Q hold on $t'_1, \ldots, t'_n$.

Because D'_T is consistent with D_S, for every pair of tuples t_i and $t_j, i, j \in [1, n]$, of the same relation, t_i and t_j are either consistent with different source tuples in D_S, or are consistent with the same source tuple and the same possible mapping. Thus, predicate R_1.id <> R_2.id OR R_1.mapping=R_2.mapping in the mirror query must hold on t'_i and t'_j. Thus, $t \in Q_m(D_T)$. □

Theorem 3. Let Q be an SPJ query and let $\overline{pM}$ be a schema p-mapping. The problem of finding the probability for a by-tuple answer to Q with respect to $\overline{pM}$ is #P-complete with respect to data complexity and is in PTIME with respect to mapping complexity. □

Proof We prove the theorem by establishing three lemmas, stating that (1) the problem is in PTIME in the size of the mapping; (2) the problem is in #P in the size of the data; (3) the problem is #P-hard in the size of the data.

Lemma 5 *Let Q be an SPJ query and let $\overline{pM}$ be a schema p-mapping. The problem of finding the probability for a by-tuple answer to Q with respect to $\overline{pM}$ is in PTIME in the size of the mapping.* □

Proof We can generate all answers in three steps. Let $T_1, \ldots, T_l$ be the relations mentioned in Q's FROM clause. Let pM_i be the p-mapping associated with table T_i. Let d_i be the number of tuples in the source table of pM_i.

1. For each $seq^1 \in \mathbf{seq}_{d1}(pM_1), \ldots, seq^l \in \mathbf{seq}_{dl}(pM_l)$, generate a target instance that is consistent with the source instance and $\overline{pM}$ as follows. For each $i \in [1, l]$, the target relation T_i contains d_i tuples, where the j-th tuple (1) is consistent with the j-th source tuple and the j-th mapping m^j in seq^i, and (2) contains null as the value of each attribute that is not involved in m^j.
2. For each target instance, answer Q on the instance. Consider only the answer tuples that do not contain the null value and assign probability $\Pi_{i=1}^{l} Pr(seq^i)$ to the tuple.
3. For each distinct answer tuple, sum up its probabilities.

According to the definition of by-tuple answers, the algorithm generates all by-tuple answers. We now prove it takes polynomial time in the size of the mapping. Assume each p-mapping pM_i contains l_i mappings. Then, the number of instances generated in step 1 is $\Pi_{i=1}^{l} l_i^{d_i}$, polynomial in the size of $\overline{pM}$. In addition, the size of each generated target instance is linear in the size of the source instance. So the algorithm takes polynomial time in the size of the mapping. □

Lemma 6 *Let Q be an SPJ query and let $\overline{pM}$ be a schema p-mapping. The problem of finding the probability for a by-tuple answer to Q with respect to $\overline{pM}$ is in #P in the size of the data.* □

Proof According to Theorem 10, we can reduce the problem to answering queries on disjunctive p-databases, which is proved to be in #P [31]. Also, Theorem 11 shows we can do the reduction in polynomial time. Thus, the problem is in #P in the size of the data. □

Lemma 7 *Consider the following query*

```
Q: SELECT 'true'
   FROM T, J, T'
   WHERE T.a = J.a AND J.b = T'.b
```

Answering Q with respect to $\overline{pM}$ is #P-hard in the size of the data. □

Proof We prove the lemma by reducing the *bipartite monotone 2-DNF problem* to the above problem.

Consider a bipartite monotone 2-DNF problem where variables can be partitioned into $X = \{x_1, \ldots, x_m\}$ and $Y = \{y_1, \ldots, y_n\}$, and $\varphi = C_1 \vee \ldots \vee C_l$, where each clause C_i has the form $x_j \wedge y_k, x_j \in X, y_k \in Y$. We construct the following query-answering problem.

P-mapping: Let $\overline{pM}$ be a schema p-mapping containing pM and pM'. Let $pM = (S, T, \mathbf{m})$ be a p-mapping where $S =< a >, T =< a' >$ and

$\mathbf{m} = \{(\{(a, a')\}, .5), (\emptyset, .5)\}$.

Let $pM' = (S', T', \mathbf{m}')$ be a p-mapping where $S' =< b >$, $T' =< b' >$ and

$\mathbf{m}' = \{(\{(b, b')\}, .5), (\emptyset, .5)\}$.

Source data: The source relation S contains m tuples: $x_1, \ldots, x_m$. The source relation S' contains n tuples: $y_1, \ldots, y_n$. The relation J contains l tuples. For each clause $C_i = x_j \wedge y_k$, there is a tuple (x_j, y_k) in J.

Obviously the construction takes polynomial time. We now prove the answer to the query is tuple true with probability $\frac{\#\varphi}{2^{m+n}}$, where $\#\varphi$ is the number of variable assignments that satisfy φ. We prove by showing that for each variable assignment $v_{x1}, \ldots, v_{xm}, v_{y1}, \ldots, v_{yn}$ that satisfies φ, there exists a mapping sequence seq such that true is a certain answer with respect to seq and the source instance, and vice versa.

For each variable assignment $v_{x1}, \ldots, v_{xm}, v_{y1}, \ldots, v_{yn}$ that satisfies φ, there must exist j and k such that $v_{xj} =$ true, $v_{yk} =$true, and there exists $C_i = x_j \wedge y_k$ in φ. We construct the mapping sequence for pM such that for each $j \in [1, m]$, if $v_{xj} =$true, $m^j = (\{(a, a')\}, .5)$, and if $v_{xk} =$false, $m^j = (\emptyset, .5)$. We construct the mapping sequence for pM' such that for each $k \in [1, n]$, if $v_{yk} =$true, $m'^k = (\{(b, b')\}, .5)$, and if $v_{yk} =$false, $m'^k = (\emptyset, .5)$. Any target instance that is consistent with the source instance and $\{seq, seq'\}$ contains x_j in T and y_k in T'. Since $C_i \in \varphi$, J contains tuple (x_j, y_k) and so true is a certain answer.

For each mapping sequence seq for pM and seq' for pM', if true is a certain answer, there must exist $j \in [1, m]$ and $k \in [1, n]$, such that x_j is in any target instance that is consistent with S and seq, y_k is in any target instance that is consistent with S' and seq', and there exists a tuple (x_j, y_k) in J. Thus, $m^j \in seq$ must be $(\{(a, a')\}, .5)$ and $m'^k \in seq'$ must be $(\{(b, b')\}, .5)$. We construct the assignments $v_{x1}, \ldots, v_{xm}, v_{y1}, \ldots, v_{yn}$ as follows. For each $j \in [1, m]$, if we have $m^j = (\{(a, a')\}, .5)$

in *seq*, x_j =true; otherwise, x_j =false. For each $k \in [1, n]$, if $m^k = (\{(b, b')\}, .5)$ in *seq*, y_k =true; otherwise, y_k =false. Obviously, the values of x_j and y_k are true, φ contains a term $x_j \wedge y_k$, and so φ is satisfied.

Counting the number of variable assignments that satisfy a bipartite monotone 2DNF Boolean formula is #P-complete. Thus, answering query Q is #P-hard. □

Note that in Lemma 7 Q contains two joins. Indeed, as stated in the following conjecture, we suspect that even for a query that contains a single join, query answering is also #P-complete. The proof is still an open problem.

Conjecture 1 Let $\overline{pM}$ be a schema p-mapping containing pM and pM'. Let $pM = (S, T, \mathbf{m})$ be a p-mapping where $S =$
$< a, b >, T =< c >$ and

$\mathbf{m} = \{(\{(a, c)\}, .5), (\{(b, c)\}, .5)\}$.

Let $pM' = (S', T', \mathbf{m}')$ be a p-mapping where $S' =< d >$, $T' =< e >$ and

$\mathbf{m}' = \{(\{(d, e)\}, .5), (\emptyset, .5)\}$.

Consider the following query

```
Q: SELECT 'true'
   FROM T1, T2
   WHERE T1.c=T2.e
```

Answering Q with respect to $\overline{pM}$ is #P-hard in the size of the data.

Theorem 4: Given an SPJ query and a schema p-mapping, returning all by-tuple answers without probabilities is in PTIME with respect to data complexity. □

Proof According to the previous lemma, we can generate all by-tuple answers by answering the mirror query on the mirror target. The size of the mirror target is polynomial in the size of the data and the size of the p-mapping, so answering the mirror query on the mirror target takes polynomial time. □

Lemma 2. Let $\overline{pM}$ be a schema p-mapping between $\bar{S}$ and $\bar{T}$. Let Q be a non-p-join query over $\bar{T}$ and let D_S be an instance of $\bar{S}$. Let $(t, Pr(t))$ be a by-tuple answer with respect to D_S and $\overline{pM}$. Let $\bar{T}(t)$ be the subset of $\mathbf{T}(D_S)$ such that for each $D \in \bar{T}(t)$, $t \in Q^{table}(D)$. The following two conditions hold:

1. $\bar{T}(t) \neq \emptyset$;
2. $Pr(t) = 1 - \Pi_{D \in \bar{T}(t), (t,p) \in Q^{table}(D)}(1 - p)$. □

Proof We first prove (1). Let T be the relation in Q that is the target of a p-mapping and let pM be the p-mapping. Let *seq* be the mapping sequence for pM with respect to which t is a by-tuple answer. Because Q is a non-p-join query, there is no self join over T. So there must exist a target tuple, denoted by t_t, that is involved in generating t. Assume this target tuple is consistent with the i-th source tuple and a possible mapping $m \in pM$. We now consider the i-th tuple database D_i in $\mathbf{T}(D_S)$. There is a target database that is consistent with D_i and m, and the database also contains the tuple t_t. Thus, t is a by-table answer with respect to D_i and m, so $D_i \in \bar{T}(t)$ and $\bar{T}(t) \neq \emptyset$.

We next prove (2). We denote by $\bar{m}(D_i)$ the set of mappings in $\mathbf{m}$, such that for each $m \in \bar{m}(D_i)$, t is a certain answer with respect to D_i and m. For the by-table answer (t, p_i) with respect to D_i, obviously $p_i = \sum_{m \in \bar{m}(D_i)} Pr(m)$.

Let d be the number of tuples in D_S. Now consider a sequence $seq =< m^1, \ldots, m^d >$. As long as there exists $i \in [1, d]$, such that $m^i \in \bar{m}(D_i)$, t is a certain answer with respect to D_S and *seq*. The probability of all sequences that satisfy the above condition is $1 - \Pi_{i=1}^{d}(1 - \sum_{m \in \bar{m}(D_i)} Pr(m)) = 1 - \Pi_{D \in \bar{T}(t), (t,p) \in Q^{table}(D)}(1 - p)$. Thus, $Pr(t) = 1 - \Pi_{D \in \bar{T}(t), (t,p) \in Q^{table}(D)}(1 - p)$. □

Theorem 5. Let $\overline{pM}$ be a schema p-mapping and let Q be a non-p-join query with respect to $\overline{pM}$. Answering Q with respect to $\overline{pM}$ in by-tuple semantics is in PTIME in the size of the data and the mapping. □

Proof We first prove Algorithm NonPJoin generates all by-tuple answers. According to Lemma 2, we should first answer Q on each tuple database, and then compute the probabilities for each answer tuple. In Algorithm NonPJoin, since we introduce the id attribute and return its values, Step 2 indeed generates by-tuple answers for all tuple databases. Finally, Step 3 computes the probability according to (2) in the lemma.

We next prove Algorithm NonPJoin takes polynomial time in the size of the data and the size of the mapping. Step 1 goes through each possible mapping to add one more correspondence and thus takes linear time in the size of the mapping. In addition, the size of the revised mapping is linear in the size of the original mapping. Since Algorithm ByTable takes polynomial time in the size of the data and the mapping, so does Step 2 in Algorithm NonPJoin; in addition, the size of the result is polynomial in the size of the data and the mapping. Step 3 of the algorithm goes over each result tuple generated from Step 2, doing the projection and computing the probabilities according to the formula, so takes linear time in the size of the result generated from Step 2, and so takes also polynomial time in the size of the data and the mapping. □

Lemma 3. Let $\overline{pM}$ be a schema p-mapping. Let Q be a projected p-join query with respect to $\overline{pM}$ and let $\bar{J}$ be a maximal p-join partitioning of Q. Let $Q_{J1}, \ldots, Q_{Jn}$ be the p-join components of Q with respect to $\bar{J}$.

For any instance D_S of the source schema of $\overline{pM}$ and result tuple $t \in Q^{tuple}(D_S)$, the following two conditions hold:

1. For each $i \in [1, n]$, there exists a single tuple $t_i \in Q_{Ji}^{tuple}(D_S)$, such that $t_1, \ldots, t_n$ generate t when joined together.
2. Let $t_1, \ldots, t_n$ be the above tuples. Then $Pr(t) = \Pi_{i=1}^{n} Pr(t_i)$. □

Proof We first prove (1). The existence of the tuple is obvious. We now prove there exists a *single* such tuple for each $i \in [1, n]$. A join component returns all attributes that occur in Q and the join attributes that join partitions. The definition of maximal p-join partitioning guarantees that for every two partitions, they are joined only on attributes that belong to relations involved in p-mappings. A projected-p-join query returns all such join attributes, so all attributes returned by the join component are also returned by Q. Thus, every two different tuples in the result of the join component lead to different query results.

We now prove (2). Since a partition in a join component contains at most one subgoal that is the target of a p-mapping in $\overline{pM}$, each p-join component is a non-p-join query. For each $i \in [1, n]$, let $\overline{seq}_i$ be the mapping sequences with respect to which t_i is a by-tuple answer. Obviously, $Pr(t_i) = \sum_{seq \in \overline{seq}_i} Pr(seq)$.

Consider choosing a set of mapping sequences $\bar{S} = \{seq_1, \ldots, seq_n\}$, where $seq_i \in \overline{seq}_i$ for each $i \in [1, n]$. Obviously, t is a certain answer with respect to $\bar{S}$. Because choosing different mapping sequences for different p-mappings are independent, the probability of $\bar{S}$ is $\Pi_{i=1}^{n} Pr(seq_i)$. Thus, we have

$$\begin{aligned} Pr(t) &= \sum_{seq_1 \in \overline{seq}_1, \ldots, seq_n \in \overline{seq}_n} \Pi_{i=1}^{n} Pr(seq_i) \\ &= \Pi_{i=1}^{n} \sum_{seq_i \in \overline{seq}_i} Pr(seq_i) \\ &= \Pi_{i=1}^{n} Pr(t_i) \end{aligned}$$

This proves the claim. □

Theorem 6. Let $\overline{pM}$ be a schema p-mapping and let Q be a projected-p-join query with respect to $\overline{pM}$. Answering Q with respect to $\overline{pM}$ in by-tuple semantics is in PTIME in the size of the data and the mapping. □

Proof We first prove Algorithm ProjectedPJoin generates all by-tuple answers for projected-p-join queries. First, it is trivial to verify that the partitioning generated by step 1 satisfies the two conditions of a p-join partitioning and is maximal. Then, step 2 and step 3 compute the probability for each by-tuple answer according to Lemma 3.

We next prove it takes polynomial time in the size of the mapping and in the size of the data. Step 1 takes time polynomial in the size of the query, and is independent of the size of the mapping and the data. The number of p-join components is linear in the size of the query and each is smaller than the original query. Since Algorithm NonPJoin takes polynomial time in the size of the data and the size of the mapping, Step 2 takes polynomial time in the size of the mapping and the size of the data too, and the size of each result is polynomial in size of the data and the mapping. Finally, joining the results from Step 2 takes polynomial time in the size of the results, and so also polynomial in the size of the data and the mapping. □

Theorem 8. Let $\overline{gpM}$ be a schema group p-mapping and let Q be an SPJ query. The mapping complexity of answering Q with respect to $\overline{gpM}$ in both by-table semantics and by-tuple semantics is in PTIME. □

Proof We first consider by-table semantics and then consider by-tuple semantics. For each semantics, we prove the theorem by first describing the query-answering algorithm, then proving the algorithm generates the correct answer, and next analyzing the complexity of the algorithm.

By-table semantics: I. First, we describe the algorithm that we answer query Q with respect to the group p-mapping $\overline{gpM}$. Assume Q's `FROM` clause contains relations $T_1, \ldots, T_l$. For each $i \in [1, l]$, assume T_i is involved in group p-mapping gpM_i, which contains g_i groups (if T_i is not involved in any group p-mapping, we assume it is involved in an identity p-mapping that corresponds each attribute with itself). The algorithm proceeds in five steps.

Step 1. We first partition all target attributes for $T_1, \ldots, T_l$ as follows. First, initialize each partition to contain attributes in one group (there are $\sum_{i=1}^{l} g_i$ groups). Then, for each pair of attributes a_1 and a_2 that occur in the same predicate in Q, we merge the two groups that t_1 and t_2 belong to. We call the result partitioning an *independence partitioning* with respect to Q and $\overline{gpM}$.

Step 2. For each partition p in an independence partitioning, if p contains attributes that occur in Q, we generate a sub-query of Q as follows. (1) The `SELECT` clause contains all variables in Q that are included in p, and an `id` column for each relation that is involved in p (we assume each tuple contains an identifier column `id`; in practice, we can use the key attribute of the tuple in place of `id`); (2) The `FROM` clause contains all relations that are involved in p; and (3) The `WHERE` clause contains only predicates that involve attributes in p. The query is called the *independence query* of p and is denoted by $Q(p)$.

Step 3. For each partition p, let $pM_1, \ldots, pM_n$ be the p-mappings for the group of attributes involved in p. For each $m^1 \in pM_1, \ldots, m^n \in pM_n$, rewrite $Q(p)$ w.r.t. $m^1, \ldots, m^n$ and answer the rewritten query on the source data. For each returned tuple, assign $\Pi_{i=1}^{n} m^i$ as the probability and add n columns $\mathsf{mapping}_1, \ldots, \mathsf{mapping}_n$, where the column $\mathsf{mapping}_i, i \in [1, n]$, has the identifier for m_i as the value. Union all result tuples.

Step 4. Join the results of the sub-queries on the id attributes. Assume the result tuple t is obtained by joining $t_1, \ldots, t_k$, then $Pr(t) = \Pi_{i=1}^{k} Pr(t_k)$.
Step 5. For tuples that have the same values, assuming to be tuple t, for attributes on Q's returned attributes but different values for the mapping attributes, sum up their probabilities as the probability for the result tuple t.

II. We now prove the algorithm returns the correct by-table answers. For each result answer tuple a, we should add up the probabilities of the possible mappings with respect to which a is generated. This is done in Step 5. So we only need to show that given a specific combination of mappings, the first four steps generate the same answer tuples as with normal p-mappings. The partitioning in Step 1 guarantees that different independence queries involve different p-mappings and so Step 2 and 3 generate the correct answer for each independence query. Step 4 joins results of the sub-queries on the id attributes; thus, for each source tuple, the first four steps generate the same answer tuple as with normal p-mappings. This proves the claim.

III. We next analyze the time complexity of the algorithm. The first two steps take polynomial time in the size of the mapping and the number of sub-queries generated by Step 2 is polynomial in the size of the mapping. Step 3 answers each sub-query in polynomial time in the size of the mapping and the result is polynomial in the size of the mapping. Step 4 joins a set of results from Step 3, where the number of the results and the size of each result is polynomial in the size of the mapping, so it takes polynomial time in the size of the mapping too and the size of the generated result is also polynomial in the size of the mapping. Finally, Step 5 takes polynomial time in the size of the result generated in Step 4 and so takes polynomial time in the size of the mapping. This proves the claim.
By-tuple semantics: First, we describe the algorithm that we answer query Q with respect to the group p-mapping $\overline{gpM}$. The algorithm proceeds in five steps and the first two steps are the same as in by-table semantics.
Step 3. For each partition p, let $pM_1, \ldots, pM_n$ be the p-mappings for the group of attributes involved in p. For each mapping sequence seq over $pM_1, \ldots, pM_n$, answer $Q(p)$ with respect to seq in by-tuple semantics. For each returned tuple, assign $Pr(seq)$ as the probability and add a column seq with an identifier of seq as the value.
Step 4. Join the results of the sub-queries on the id attributes. Assume the result tuple t is obtained by joining $t_1, \ldots, t_k$, then $Pr(t) = \Pi_{i=1}^{k} Pr(t_k)$.
Step 5. Let $t_1, \ldots, t_n$ be the tuples that have the same values, tuple t, for attributes on Q's returned attributes but different values for the seq attributes, sum up their probabilities as the probability for the result tuple t.

We can verify the correctness of the algorithm and analyze the time complexity in the same way as in by-table semantics. □

Proposition 1. For each $n \geq 1$, $\mathcal{M}_{ST}^{n+1} \subset \mathcal{M}_{ST}^{n}$. □

Proof We first prove for each $n \geq 1$, $\mathcal{M}_{ST}^{n+1} \subseteq \mathcal{M}_{ST}^{n}$, and then prove there exists an instance in $\mathcal{M}_{ST}^{n}$ that does not have an equivalent instance in $\mathcal{M}_{ST}^{n+1}$.

(1) We prove $\mathcal{M}_{ST}^{n+1} \subseteq \mathcal{M}_{ST}^{n}$ by showing for each $(n+1)$-group p-mapping we can find a n-group p-mapping equivalent to it. Consider an instance $gpM = (S, T, \overline{pM}) \in \mathcal{M}_{ST}^{n+1}$, where $\overline{pM} = \{pM_1, \ldots, pM_{n+1}\}$. We show how we can construct an instance $gpM' \in \mathcal{M}_{ST}^{n}$ that is equivalent to gpM. Consider merging $pM_1 = (S_1, T_1, \mathbf{m}_1)$ and $pM_2 = (S_2, T_2, \mathbf{m}_2)$ and generating a probabilistic mapping $pM_{1-2} = (S_1 \cup S_2, T_1 \cup T_2, \mathbf{m}_{1-2})$, where $\mathbf{m}_{1-2}$ includes the Cartesian product of the mappings in $\mathbf{m}_1$ and $\mathbf{m}_2$. Consider the n-group p-mapping $gpM' = (S, T, \overline{pM'})$, where $\overline{pM'} = \{pM_{1-2}, pM_3, \ldots, pM_{n+1}\}$. Then, gpM and gpM' describe the same mapping.

(2) We now show how we can construct an instance in $\mathcal{M}_{ST}^{n}$ that does not have an equivalent instance in $\mathcal{M}_{ST}^{n+1}$. If S and T contain less than n attributes, $\mathcal{M}_{ST}^{n} = \emptyset$ and the claim holds. Otherwise, we partition attributes in S and T into $\{\{s_1\}, \ldots, \{s_{n-1}\}, \{s_n, \ldots, s_m\}\}$ and $\{\{t_1\}, \ldots, \{t_{n-1}\}, \{t_n, \ldots, t_l\}\}$. Without losing generality, we assume $m \leq l$. For each $i \in [1, n-1]$, we define

$$\mathbf{m}_i = \{(\{(s_i, t_i)\}, 0.8), (\emptyset, 0.2)\}.$$

In addition, we define

$$\mathbf{m}_n = \{(\{(s_n, t_n)\}, \frac{1}{(m-n+1)}), \ldots, (\{(s_m, t_n)\}, \frac{1}{(m-n+1)})\}.$$

We cannot further partition S into $n+1$ subsets such that attributes in different subsets correspond to different attributes in T. Thus, we cannot find a $(n+1)$-group p-mapping equivalent to it. □

Proposition 2. Given a p-mapping $pM = (S, T, \mathbf{m})$, we can find in polynomial time in the size of pM the maximal n and an n-group p-mapping gpM, such that gpM is equivalent to pM. □

Proof We prove the theorem by first presenting an algorithm that finds the maximal n and the equivalent n-group p-mapping gpM, then proving the correctness of the algorithm, and finally, analyzing its time complexity.

I. We first present the algorithm that takes a p-mapping $pM = (S, T, \mathbf{m})$, finds the maximal n and the n-group p-mapping that is equivalent to pM.
Step 1. First, partition attributes in S and T. Initialize the partitions such that each contains a single attribute in S or T. Then for each attribute correspondence (s, t) occurring in a possible mapping, if s and t are in different partitions, merge the two partitions. Let $\mathcal{P} = \{p_1, \ldots, p_n\}$ be the result partitioning.
Step 2. For each partition $p_i, i \in [1, n]$, and each $m \in \mathbf{m}$, select the correspondences in m that involve only

attributes in p_i, use them to construct a sub-mapping, and assign $Pr(m)$ to the sub-mapping. We compute the marginal probability of each sub-mapping.
Step 3. For each partition $p_i, i \in [1, n]$, examine if its possible mappings are independent of the possible mappings for the rest of the partitions. Specifically, for each partition $p_j, j > i$, if there exists a possible mapping m for p_i and a possible mapping m' for p_j, such that $Pr(m|m') \neq Pr(m)$, merge p_i into p_j. For the new partition p_j, update its possible sub-mappings and their marginal probabilities. Step 3 generates a set of partitions, each with a set of sub-mappings and their probabilities.
Step 4. Each partition generated in Step 3 is associated with a p-mapping. The set of all p-mappings forms the group p-mapping gpM that is equivalent to pM.

II. We now prove the correctness of the algorithm. It is easy to prove gpM is equivalent to pM. Assume gpM is an n-group p-mapping. We next prove n is maximal. Consider another group p-mapping gpM'. We now prove for each p-mapping in gpM', it either contains all attributes in a partition generated in Step 3 or contains none of them. According to the definition of group p-mapping, each p-mapping in gpM' must contain either all attributes or none of the attributes in a partition in $\mathcal{P}$. In addition, every two partitions in $\mathcal{P}$ that are merged in Step 3 are not independent and have to be in the same p-mapping in gpM' too. This proves the claim.

III. We next consider the time complexity of the algorithm. Let m be the number of mappings in pM, and a be the minimum number of attributes in R and in S. Step 1 considers each attribute correspondence in each possible mapping. A mapping contains no more than a attribute correspondences, so Step 1 takes time $O(ma)$. Step 2 considers each possible mapping for each partition to generate sub-mappings. The number of partitions cannot exceed a, so Step 2 also takes time $O(ma)$. Step 3 considers each pair of partitions, and takes time $O(ma^2)$. Finally, Step 4 outputs the results and takes time $O(ma)$. Overall, the algorithm takes time $O(ma^2)$, which is polynomial in the size of the full-distribution instance. □

Theorem 9. Let $\overline{pC}$ be a schema p-correspondence, and Q be an SPJ query. Then, Q is p-mapping independent with respect to $\overline{pC}$ if and only if for each $pC \subseteq \overline{pC}$, Q is a single-attribute query with respect to pC. □

Proof We prove for the case when there is a single p-correspondence in $\overline{pC}$ and it is easy to generalize our proof to the case when there are multiple p-correspondences in $\overline{pC}$.

If: Let pM_1 and pM_2 be two p-mappings over S and T where $pC(pM_1) = pC(pM_2)$. Let D_S be a database of schema S. Consider a query Q over T. Let t_j be the only attribute involved in query Q. We prove $Q(D_S)$ is the same with respect to pM_1 and pM_2 in both by-table and by-tuple semantics.

We first consider by-table semantics. Assume S has n attributes $s_1, \ldots, s_n$. We partition all possible mappings in pM_1 into $\bar{m}_0, \ldots, \bar{m}_n$, such that for any $m \in \bar{m}_i, i \in [1, n]$, m maps attribute s_i to t_j, and for any $m \in \bar{m}_0$, m does not map any attribute in S to t_j. Thus, for each $i \in [1, n], Pr(\bar{m}_i) = Pr(c_{ij})$.

Consider a tuple t. Assume t is an answer tuple with respect to a subset of possible mappings $\bar{m} \subseteq \mathbf{m}$. Because Q contains only attribute t_j, for each $i \in [0, n]$, either $\bar{m}_i \subseteq \bar{m}$ or $\bar{m}_i \cap \bar{m} = \emptyset$. Let $\bar{m}_{k_1}, \ldots, \bar{m}_{k_l}, k_1, \ldots, k_l \in [0, n]$, be the subsets of $\bar{m}$ such that $\bar{m}_{k_j} \subseteq \bar{m}$ for any $j \in [1, l]$. We have

$$Pr(t) = \sum_{i=1}^{l} Pr(\bar{m}_{k_i}) = \sum_{i=1}^{l} Pr(c_{k_i j}).$$

Now consider pM_2. We partition its possible mappings in the same way and obtain $\bar{m}'_0, \ldots, \bar{m}'_n$. Since Q contains only attribute t_j, for each $i \in [0, n]$, the result of Q with respect to $m' \in \bar{m}'_i$ is the same as the result with respect to $m \in \bar{m}_i$. Therefore, the probability of t with respect to pM_2 is

$$Pr(t)' = \sum_{i=1}^{l} Pr(\bar{m}'_{k_i}) = \sum_{i=1}^{l} Pr(c_{k_i j}).$$

Thus, $Pr(t) = Pr(t)'$ and this proves the claim.

We can prove the claim for by-tuple semantics in a similar way where we partition mapping sequences. We omit the proof here.

Only if: We prove by showing that for every query Q that contains more than one attribute in a relation being involved in a p-correspondence, there exist p-mappings pM_1 and pM_2 and source instance D_S, such that $Q(D_S)$ obtains different results with respect to pM_1 and pM_2.

Assume query Q contains attributes a' and b' of T. Consider two p-mappings pM_1 and pM_2, where

$$pM_1 = \{(\{(a, a'), (b, b')\}, .5), (\{(a, a')\}, .3), (\{(b, b')\}, .2)\}$$
$$pM_2 = \{(\{(a, a'), (b, b')\}, .6), (\{(a, a')\}, .2), (\{(b, b')\}, .1), (\emptyset, .1)\}$$

One can verify that $pC(pM_1) = pC(pM_2)$.

Consider a database D_S, such that for each tuple of the source relation in pM_1 and pM_2, the values for attributes a and b satisfy the predicates in Q. Since only when the possible mapping $\{(a, a'), (b, b')\}$ is applied can we generate valid answer tuples, but the possible mapping $\{(a, a'), (b, b')\}$ has different probabilities in pM_1 and pM_2, $Q(D_S)$ obtains different results with respect to pM_1 and pM_2 in both semantics. □

Corollary 1. Let $\overline{pC}$ be a schema p-correspondence, and Q be a p-mapping independent SPJ query with respect to $\overline{pC}$. The mapping complexity of answering Q with respect to $\overline{pC}$ in both by-table semantics and by-tuple semantics is in PTIME. □

Proof **By-table:** We revise algorithm By-Table, which takes polynomial time in the size of the schema p-mapping, to compute answers with respect to schema p-correspondences. At the place where we consider a possible mapping in the algorithm, we revise to consider a possible attribute correspondence. Obviously the revised algorithm generates the correct by-table answers and takes polynomial time in the size of the mapping.

By-tuple: We revise the algorithm in the proof of Theorem 3, which takes polynomial time in the size of the schema p-mapping, to compute answers with respect to schema p-correspondences. Everywhere we consider a possible mapping in the algorithm, we revise to consider a possible attribute correspondence. Obviously the revised algorithm generates the correct by-tuple answers and takes polynomial time in the size of the mapping. □

Theorem 10. Let $pM = (S, T, \mathbf{m})$ be a p-mapping and D_S be an instance of S.

1. There is a unique by-table core universal solution and a unique by-tuple core universal solution up to isomorphism for D_S with respect to pM.
2. Let Q be a conjunctive query over T. We denote by $Q(pD)$ the results of answering Q on pD and discarding all answer tuples containing null values. Then,

   $Q^{table}(D_S) = Q(pD_T^{table})$.

   Similarly, let pD_T^{tuple} be the by-tuple core universal solution for D_S under pM. Then,

   $Q^{tuple}(D_S) = Q(pD_T^{tuple})$. □

Proof We first consider by-table semantics and then consider by-tuple semantics. For each semantics, we first present an algorithm that generates the core universal solution, then prove the generated solution (1) is a core universal solution and (2) is unique, and last prove answering Q on the core universal solution obtains the same results as answering Q on the source data with respect to pM.

By-table semantics: I. First, we describe the algorithm that generates a by-table core universal solution for D_S with respect to pM. The algorithm proceeds in two steps.

Step 1. For each mapping $m \in \mathbf{m}$, generate the core universal solution for D_S with respect to m, denoted by $(D_T, Pr(D_T))$, as follows.

1. For each tuple $t \in D_S$, apply m to obtain t' as follows: (1) for each attribute $a_t \in T$ such that there exists an attribute correspondence $(a_s, a_t) \in m$, the value of a_j in t' is the same constant as the value of a_s in t; (2) for the rest of the attributes $a \in T$, the value of a in t' is a fresh labeled null.
2. If there does not exist a tuple in D_T that has the same constant values as t', insert t' to D_T.
3. Set $Pr(D_T)$ to $Pr(m)$.

Step 2. Let pD_T be a p-database with all possible databases generated as described. Examine each pair of possible databases $(D_T, Pr(D_T))$ and $(D'_T, Pr(D'_T))$. If D_T and D'_T are isomorphic, replace them with a single possible database $(D_T, Pr(D_T) + Pr(D'_T))$.

II. We now prove the result pD_T is a by-table core universal solution. First, the way we generate the p-database guarantees that it is a by-table *solution*.

Second, we show for every solution pD'_T for D_S with respect to pM, we can construct a homomorphism mapping from pD_T to pD'_T. Consider $(D_T, Pr(D_T)) \in pD_T$. Let $\bar{m}(D_T) \subseteq \mathbf{m}$ be the mappings that are involved in generating D_T. Each possible mapping in $\bar{m}(D_T)$ must correspond to a possible database $D'_T \in pD'_T$ and different mappings can correspond to the same possible database. Let $\bar{D}'_T(\bar{m}(D_T)) \subseteq pD'_T$ be the set of possible databases that together correspond to all mappings in $\bar{m}(D_T)$. We define the homomorphism mapping as $D_T \to \bar{p}D'_T(\bar{m}(D_T))$. (1) Because for each $m \in \bar{m}(t)$, D_T is a core universal solution, for every $D'_T \in \bar{D}'_T(\bar{m}(D_T))$, there is a homomorphism from D_T to D'_T. (2) $Pr(D_T) = \sum_{m \in \bar{m}(D_T)} Pr(m) = \sum_{D \in \bar{D}'_T(\bar{m}(D_T))} Pr(D)$. (3) For any $D_T^1, D_T^2 \in pD_T, D_T^1 \neq D_T^2$, $h(D_T^1)$ and $h(D_T^2)$ do not overlap because otherwise, there are two possible mappings that correspond to different possible databases in pD_T but the same possible database in pD'_T, so D_T^1 and D_T^2 should be isomorphic and should be merged in Step 2. All possible databases in pD'_T together cover all mappings, and so we can partition $h(pD_1), \dots, h(pD_n)$. Thus, pD_T is a *universal* solution.

Finally, the way we generate pD_T guarantees that in each possible database, any two tuples are not homomorphic. So pD_T is a *core* universal solution.

III. Next, we prove pD_T is unique. Assume there is another p-database pD'_T that is also a core universal solution. We now prove there exists an isomorphism between pD_T and pD'_T. Because pD_T is a universal solution, there is a homomorphism h from pD_T to pD'_T. Similarly, there is a homomorphism h' from pD'_T to pD_T. Thus, the number of possible databases in pD_T and pD'_T must be the same and both h and h' are one-to-one mappings. Now we prove for every $D \in pD_T$, $h'(h(D)) = D \in pD_T$ and so D and $h(D)$ are isomorphic. Assume in contrast, this statement does not hold. Then, because the numbers of databases in pD_T and pD'_T are finite, there must be a database $D \in pD_T$ for which there exist $k \geq 1$ databases in pD_T such that $h'(h(D)) = D_1, h'(h(D_1)) = D_2, \dots, h'(h(D_{k-1})) = D_k$ and $h'(h(D_k)) = D$. For each $i \in [1, k]$, D is homomorphic to D_i and D_i is homomorphic to D. Thus, $D, D_1, \dots, D_k$ are all isomorphic. Now consider a p-database pD_0 that contains all possible databases in pD_T except $D_1, \dots, D_k$. This database is also a by-table solution of D_S. However, as pD_0 contains less databases, there does not exist a homomorphism from pD_T to pD_0, contradicting the fact that pD_T is a universal solution. Thus, pD_T and pD'_T are isomorphic.

IV. Finally, we prove that $Q^{table}(D_S) = Q(pD_T)$ by showing that for every tuple t, the probability of t in $Q^{table}(D_S)$ is the same as in $Q(pD_T)$ (the probability can be 0). We denote by $\bar{m}(t)$ the set of mappings with respect to which t is a certain answer ($\bar{m}(t)$ can be empty), and by $\bar{D}(\bar{m}(t))$ the set of possible databases related to mappings in $\bar{m}(t)$. Obviously, $Pr(\bar{m}(t)) = Pr(\bar{D}(\bar{m}(t)))$. So we only need to prove that (1) for each $D \in \bar{D}(\bar{m}(t))$, $t \in Q(D)$, and (2) for each $D \notin \bar{D}(\bar{m}(t))$, $t \notin Q(D)$. First, for each $m \in \bar{m}(t)$, there exists at least a source tuple $t_s \in D_S$ on which answering Q obtains t. Then, according to the way we generate $D(m)$, answering Q on t_s's corresponding tuple in $D(m)$ must also obtain t. Second, consider a database $D \notin \bar{m}(t)$. Let m' be the possible mapping with respect to which D is consistent with D_S. The way we construct pD guarantees that $m' \notin \bar{m}(t)$. Assume in contrast, answering Q on D generates t. Thus, there must exist a tuple $t_t \in D$ on which answering Q obtains t. Accordingly, there must exist a source tuple $t_s \in D$ on which answering Q can generate t as a certain answer with respect to m', contradicting the fact that t is not a certain answer with respect to m'. This proves the claim.

By-tuple semantics: Here we generate the by-tuple core universal solution in a similar way except that we consider each mapping sequence with the same length as the number of tuples in D_S, rather than each possible mapping. The rest of the proof is similar to the by-table semantics. □

Lemma 4. Let $pM = (S, T, \mathbf{m})$ be a p-mapping and D_S be an instance of S.

The by-tuple core universal solution for D_S under pM can be represented as a disjunctive p-database. □

Proof We describe how we construct such disjunctive p-database, denoted by $pD_T^\vee$, and show it is equivalent to the p-database we constructed in the proof of Theorem 10.

The disjunctive p-database $pD^\vee$ has attributes in T and a key column that is the key of the relation. For the i-th tuple t_s in S and each $m \in \mathbf{m}$, generate a target tuple t_t, such that (1) for each attribute correspondence $(a_s, a_t) \in m$, the value of a_t is the same as the value of a_s in t_s; (2) for each attribute a_t in T that is not involved in any attribute correspondence in m, the value of a_t is a fresh labeled null; and (3) the value of the key attribute is i. The probability of the tuple is $Pr(m)$. Let n be the number of tuples in D_S and l be the number of mappings in pM. Generating the target instance takes time $O(l \cdot n)$, polynomial in the size of the data and the mapping.

We now show the equivalence of $pD_T^\vee$ and pD_T, the p-database constructed as described in the proof of Theorem 10. The disjunctive p-database $pD^\vee$ is equivalent to a p-database pD' that contains n^l possible worlds, in each of which the possible database corresponds to a mapping sequence of length n, which is isomorphic to the p-database we generated in the first step towards generating pD_T. This proves the claim. □

Theorem 11. Let $pM = (S, T, \mathbf{m})$ be a p-mapping and D_S be an instance of S.

Generating the by-table or by-tuple core universal solution for D_S under pM takes polynomial time in the size of the data and the mapping. □

Proof Let n be the number of source tuples and l be the number of possible mappings. We first examine the time complexity of generating the by-table core universal solution. In the algorithm described in the proof of Theorem 10, the first step takes time $O(n \cdot l)$. In the second step, we basically compare the constant values of tuples so it takes time $O(n^2 l^2)$. Thus, the algorithm takes time $O(n^2 l^2)$, which is polynomial in the size of the data and the size of the mapping.

We now examine the time complexity of generating the by-tuple disjunctive p-database solution. In the algorithm described in the proof of Lemma 4, for each source tuple and each mapping, we generate a target tuple. So the algorithm takes time $O(n \cdot l)$, which is linear in the size of the data and the size of the mapping. □

Theorem 13. Let pGM be a GLAV p-mapping between a source schema $\bar{S}$ and a target schema $\bar{T}$. Let D_S be an instance of $\bar{S}$.

Generating the by-table core universal solution for D_S under pM takes polynomial time in the size of the data and the mapping. □

Proof For each possible GLAV mapping $m \in pGM$, generating the core universal solution takes polynomial time [11] in the size of the data and the size of m. The number of core universal solutions we need to generate is the same as the number of possible mappings in pGM. Thus, generating the by-table core universal solution for D_S under pM takes polynomial time in the size of the data and the size of the p-mapping. □

Theorem 14. Let $pM_1 = (R, S, \mathbf{m_1})$ and $pM_2 = (S, T, \mathbf{m_2})$ be two p-mappings. Between R and T there exists a unique p-mapping, pM, that is the composition of pM_1 and pM_2 in both by-table and by-tuple semantics and we can generate pM in polynomial time. □

Proof We prove the theorem by first describing an algorithm that generates the composition of two mappings and analyzing the complexity of the algorithm, then proving it is both the by-table composition and the by-tuple composition, and finally showing it is unique.

I. We generate the composition mapping pM in two steps.

Step 1. For each $m_1 \in \mathbf{m_1}$ and $m_2 \in \mathbf{m_2}$, generate the composition of m_1 and m_2 as follows. For each correspondence $(r, s) \in m_1$ and each correspondence $(s, t) \in m_2$, add (r, t) to $m_1 \circ m_2$. The probability of $m_1 \circ m_2$ is set to $Pr(m_1) \cdot Pr(m_2)$.

Step 2. Merge equivalent mappings generated in the previous step and take the sum of their probabilities as the probability of the merged mapping.

Let m be the number of mappings in pM_1 and n be the number of mappings in pM_2. The first step of the algorithm takes time $O(m \cdot n)$ and the second step of the algorithm takes time $O(m^2n^2)$. Thus, the algorithm takes time $O(m^2n^2)$, polynomial in the size of the input.

II. We now prove pM is a composition of pM_1 and pM_2. We first consider the by-table semantics. It is easy to prove the "if" side in Definition 22 so we only prove the "only if" side.

Consider an instance D_R of R and an instance D_T of T where D_T is consistent with D_R with probability p. We now describe how we construct a set of instances $\bar{D}_S$ of S such that the three conditions in Definition 22 hold. Let $\bar{m}$ be the set of mappings in pM with respect to which D_T is consistent with D_R. For each $m \in \bar{m}$, according to the way we construct pM, there must be a list of mappings $\bar{m}_1$ and a list of mappings $\bar{m}_2$ with the same length, such that for the i-th mapping $m_1^i \in \bar{m}_1$ and the i-th mapping $m_2^i \in \bar{m}_2$, composing them obtains m. For each i, construct the core universal solution of D_R with respect to m_1^i and denote it by D_S^i. Obviously, D_S^i is consistent with D_R with probability $Pr(m_1^i)$. The way we construct pM also guarantees that D_T is consistent with D_S with probability $Pr(m_2^i)$. Finally, for an instance D_S of S that is not isomorphic to any database in $\bar{m}$, it cannot happen that D_S is consistent with D_R and D_T is consistent with D_S. Thus, $p = \sum_i Pr(m_1^i)Pr(m_2^i)$.

The proof for by-tuple semantics is similar, except that we consider each mapping sequence.

III. We prove for by-table semantics and the proof for by-tuple semantics is similar. Assume there exists another p-mapping pM' that is the composition of pM_1 and pM_2. Assume pM' contains a possible mapping m that does not occur in pM. Then, there must exist an instance D_T of T that is consistent with D_R with respect to m but not with respect to any mapping in pM. Thus, there must exist a set of instances of S that satisfy the three conditions in the definition, leading to the contradictory fact that D_T should also be consistent with D_R with respect to pM. This proves the claim. □

Theorem 15. *N-group ($n > 1$) p-mappings are not closed under mapping composition.* □

Proof We show a counter example where the composition of two 2-group p-mappings can not be represented as a 2-group p-mapping.

Let pM_1 be a 2-group p-mapping between $R(a, b, c)$ and $S(a', b', c')$, where attributes in R are partitioned into $\{a\}$ and $\{b, c\}$, and attributes in S are partitioned into $\{a'\}$ and $\{b', c'\}$:

$$\overline{pM_1} = \{pM_1, pM_1'\},$$
$$pM_1 = \{(\{(a, a')\}, 1)\},$$
$$pM_1' = \{(\{(b, b'), (c, c')\}, .5), (\{(b, c'), (c, b')\}, .5)\}.$$

Let pM_2 be a 2-group p-mapping between $S(a', b', c')$ and $T(a'', b'', c'')$, where attributes in S are partitioned into $\{a', b'\}$ and $\{c'\}$, and attributes in T are partitioned into $\{a'', b''\}$ and $\{c''\}$, and the two p-mappings are

$$\overline{pM_2} = \{pM_2, pM_2'\},$$
$$pM_2 = \{(\{(a', a''), (b', b'')\}, .5), (\{(a', b''), (b', a'')\}, .5)\},$$
$$pM_2' = \{(\{(c', c'')\}, 1)\}.$$

The composition of pM_{RS} and pM_{ST} contains four possible mappings, shown as follows:

$$\begin{aligned} pM_3 = \{&(\{(a, a''), (b, b''), (c, c'')\}, .25),\\ &(\{(a, b''), (b, a''), (c, c'')\}, .25),\\ &(\{(a, a''), (b, c''), (c, b'')\}, .25),\\ &(\{(a, b''), (b, c''), (c, a'')\}, .25)\}. \end{aligned}$$

In this mapping, R's attribute b can be mapped to any attribute in T, and thus there does not exist an equivalent 2-group p-mapping. □

Theorem 16. Let $pM = (S, T, \mathbf{m})$ be a p-mapping. Then, pM has an inverse p-mapping if and only if

- $\mathbf{m}$ contains a single possible mapping $(m, 1)$;
- each attribute in S is involved in an attribute correspondence in m. □

Proof If: Construct a p-mapping $pM' = (T, S, \mathbf{m}')$ where $\mathbf{m}'$ contains a single mapping m' and for each correspondence $(s, t) \in m$, there is a correspondence $(t, s) \in m'$.

If we compose pM and pM' in the way we described in the proof of Theorem 14, we obtain a p-mapping between S and S that contains a single possible mapping, where the mapping maps each attribute to itself. Thus, the result is an identical p-mapping and pM' is an inverse of pM.

Only if: We show that if any of the conditions does not hold, we cannot generate an inverse mapping of pM. First, assume $\mathbf{m}$ contains two possible mappings m_1 and m_2 and both of them have inverse mappings, denoted by m_1^{-1} and m_2^{-1}. Then, if there exists a p-mapping pM' that is the inverse of pM, it should contain both m_1^{-1} and m_2^{-1} as possible databases. However, a mapping has a unique inverse mapping, so composing m_1 with m_2^{-1} does not obtain the identical mapping. Thus, composing pM with pM' does not obtain the identical mapping.

Now consider a p-mapping which satisfies the first condition, but not the second. Let a be the source attribute that is not involved in any attribute correspondence in m. Then for any mapping m' from T to S, composing m with m' does not map m to any attribute so the result is not an identical mapping. This proves the claim. □

Theorem 17. Let $\overline{pCM}$ be a complex schema p-mapping between schemas $\bar{S}$ and $\bar{T}$. Let D_S be an instance of $\bar{S}$.

1. Let Q be an SPJ query over $\bar{T}$. The data complexity and mapping complexity of computing $Q^{table}(D_S)$ with respect to $\overline{pCM}$ are PTIME. The data complexity of computing $Q^{tuple}(D_S)$ with respect to $\overline{pCM}$ is #P-complete. The mapping complexity of computing $Q^{tuple}(D_S)$ with respect to $\overline{pCM}$ is in PTIME.
2. Generating the by-table or by-tuple core universal solution for D_S under $\overline{pCM}$ takes polynomial time in the size of the data and the mapping. □

Proof We prove the theorem by showing that we can construct a normal schema p-mapping from $\overline{pCM}$ and answer a query with respect to the normal p-mapping. For each $pCM \in \overline{pCM}$ between source $S(s_1, \ldots, s_m)$ and target $T(t_1, \ldots, t_n)$, we construct a normal p-mapping $pM = (S', T', m)$ as follows. The source S' contains all elements of the power set of $\{s_1, \ldots, s_m\}$ and the target T' contains all elements of the power set of $\{t_1, \ldots, t_n\}$. For each complex mapping $cm \in pCM$, we construct a mapping m such that for each set correspondence between S and T in cm, m contains an attribute correspondence between the corresponding set attributes in S' and T'. Because each attribute set occurs in one correspondence in cm, m is a one-to-one mapping. The result pM contains the same number of possible mappings and each mapping contains the same number of correspondences as pCM. We denote the result schema p-mapping by $\overline{pM}$. The complexity of data exchange carries over.

Now consider query answering. Since for each possible mapping $cm \in \overline{pCM}$, an attribute is involved in at most one correspondence, query answering with respect to $\overline{pCM}$ gets the same result as with respect to $\overline{pM}$ and so the complexity results for normal schema p-mappings carry over. □

Theorem 18. Let $\overline{pUM}$ be a union schema p-mapping between a source schema $\bar{S}$ and a target schema $\bar{T}$. Let D_S be an instance of $\bar{S}$.

1. Let Q be a conjunctive query over $\bar{T}$. The problem of computing $Q^{table}(D_S)$ with respect to $\overline{pUM}$ is in PTIME in the size of the data and the mapping; the problem of computing $Q^{tuple}(D_S)$ with respect to $\overline{pUM}$ is in PTIME in the size of the mapping and #P-complete in the size of the data.
2. Generating the by-table or by-tuple core universal solution for D_S under $\overline{pUM}$ takes polynomial time in the size of the data and the mapping. □

Proof Answering a query with respect to a union mapping can be performed by first answering the query on each element mapping, and then taking the union of the results. Thus, the complexity of answering a query with respect to a union mapping is the same as the complexity of answering a query with respect to a normal mapping. When we have union probabilistic mappings, in each step where we need to answer a query with respect to a possible union mapping, we first answer the query on each element mapping and then union the results. So the complexity results carry over.

Now consider data exchange for a union probabilistic schema mapping. In by-table semantics, we can generate the core universal solution in the same way as with respect to normal mappings, except that we consider each element mapping in the union mapping when we generate the target for a source tuple. Thus, we can generate the by-table core universal mapping in polynomial time.

In by-tuple semantics, we need a new representation of p-databases, called *union disjunctive p-databases*, which we define as follows. Let R be a relation schema where there exists a set of attributes that together form the *key* of the relation and an attribute group. Let $pUD_R^{\vee}$ be a set of tuples of R, where some of the tuples are attached with probabilities. We say that $pUD_R^{\vee}$ is a *union disjunctive p-database* if (1) for each key value that occurs in $pUD_R^{\vee}$, the probabilities of the tuples with this key value sum up to 1, (2) the value of group in each tuple with a probability is unique, and the value of group in each tuple without probability is the same as that of a tuple with probability and with the same key value. In a union disjunctive p-databases, we consider tuples with the same key value as disjoint, and tuples with the same group value as unioned. Specifically, let $key_1, \ldots, key_n$ be the set of all distinct key values in $pD_R^{\vee}$. For each $i \in [1, n]$, we denote by d_i the number of tuples whose key value is key_i and who has a probability. Then, $pD_R^{\vee}$ can define a set of $\Pi_{i=1}^n d_i$ possible databases, where each possible database $(D, Pr(D))$ contains n tuples $t_1, \ldots, t_n$ with probabilities and m tuples without probabilities, such that (1) for each $i \in [1, n]$, the key value of t_i is key_i; (2) a tuple without probability is in D if and only if it shares the same value of group with one of $t_1, \ldots, t_n$; and (3) $Pr(D) = \Pi_{i=1}^n Pr(t_i)$.

We generate the by-tuple core universal solution with respect to a union probabilistic mapping in the same way as for normal p-mappings, except that for each possible union mapping, we generate a target tuple with respect to each element mapping, assigning a unique value to their group attribute and assigning the probability of the union mapping to one and only one of the target tuples. Thus, we can generate the by-tuple core universal mapping in polynomial time as well. □

Theorem 19. Let $\overline{cpM}$ be a conditional schema p-mapping between $\bar{S}$ and $\bar{T}$. Let D_S be an instance of $\bar{S}$.

1. Let Q be an SPJ query over $\bar{T}$. The problem of computing $Q^{tuple}(D_S)$ with respect to $\overline{cpM}$ is in PTIME in the size of the mapping and #P-complete in the size of the data.
2. Generating the by-tuple core universal solution for D_S under $\overline{cpM}$ takes linear time in the size of the data and the mapping. □

Proof By-tuple query answering with respect to conditional schema p-mappings is essentially the same as that with respect to normal p-mappings, where for each source tuple, we first decide which condition it satisfies and then consider applying possible mappings associated with that condition. Thus, the complexity of by-tuple query-answering with respect to normal schema p-mappings carries over.

Constructing the core universal by-tuple solution is also essentially the same as that with respect to normal p-mappings, where for each source tuple s we first decide which condition it satisfies and then generate target tuples that are consistent with s and the possible mappings associated with that condition. Thus, the complexity of by-tuple data-exchange with respect to normal schema p-mappings carries over as well. □