DOI:10.1145/3624717

Deploying possible world semantics and the challenge of computing the certain answers to queries.

BY BENNY KIMELFELD AND PHOKION G. KOLAITIS

A Unifying Framework for Incompleteness, Inconsistency, and Uncertainty in Databases

DATABASES ARE OFTEN assumed to have definite content. The reality, though, is the database at hand may be deficient due to missing, invalid, or uncertain information. As a simple illustration, the primary address of a person may be missing, or it may conflict with another primary address, or it may be improbable given the presence of nearby businesses. A common practice to address this challenge is to rectify the database by fixing the gaps, as done in data imputation, entity resolution, and data cleaning. The process of rectifying the database, however, may involve arbitrary choices due to computational limitations, such as errors in statistical or machine-learning models, or mere lack of information that even humans cannot cope with in full confidence. In turn, answers to queries over the deficient database may depend on the choices made to rectify it; thus, the answers to queries may vary from one choice to choice, even though both choices may be equally legitimate.

In the pursuit of principled solutions, there has been a continuous research effort to develop fundamental approaches for handling database deficiency with no (or with less) arbitrariness. The purpose of this review article is to highlight some of the ways in which the *possible world* semantics has been deployed as a principled approach to overcome database deficiency in different contexts. In this approach, we acknowledge that we need to rectify the deficiency: fill in missing information, delete wrong records (hereafter *tuples* or *facts*), correct erroneous values, and so on. Yet, since many rectifications may exist and since we do not know which is the correct one, we do not commit to a specific one. Instead, we view our deficient database as a representation of the re-

» key insights

- **Possible world semantics give meaning to queries over incomplete, inconsistent, or uncertain databases. Such databases are viewed as compact representations of all their possible rectifications; the "certain answers" are the query answers that hold true in every possible rectification.**
- **Queries that are computationally tractable under standard database semantics may become intractable under possible world semantics. In many cases, the tools of computational complexity can be applied to delineate the boundary between tractability and intractability.**
- **Election databases form a new setting for the framework of possible worlds. This setting brings social choice theory and relational databases together and supports reasoning about elections with partial voter preferences and with contextual information about the candidates.**

sults of *all* conceivable rectifications, each such rectification giving rise to a legitimate candidate of a valid database that we call a *possible world*.

Since the possible worlds differ from each other, a *query* may produce different collections of answers (which are also tuples) when applied to different possible worlds. Therefore, query answering requires the use of an aggregation method to combine the query results over the possible worlds. The *intersection* of all possible results is the prototypical example of such an aggregation method. In this context, the tuples in the intersection are called the "certain answers" because they are obtained in *every* possible world, regardless of what rectification has led to it. Intersection is the most studied aggregation method because of its intuitive meaning: it extracts the answers for which validity is not affected by the database anomaly. For example, if we have uncertainty regarding the apartment number of a person x, we can still be certain about x living in Paris in the same apartment as y. Intersection is the main, but not the only, aggregation method discussed in this survey. The *union* is a different type of aggregation, where we are interested in the "possible answers," that is, the answers that are obtained in *at least one* possible world. Probabilistic variants of the possible world semantics have also been studied in depth; here, each rectification has a probability (that can be determined from the database content), and then the aggregation is done by *marginal inference*: to each tuple, we associate the probability that it is an answer when the query is applied to a random possible world.

It is often the case that the original (deficient) database constitutes a compact representation of the set of possible worlds, and that this set is infeasible to materialize or store; for example, the number of possible worlds can be exponential in the size of the representation or even infinite. Consequently, finding the certain answers to queries poses a significant computational challenge because an efficient aggregation of the answers over all possible worlds has to be performed without actually accessing each possible world, but by directly processing the compact deficient database. Such processing might be attainable for some queries but unattainable for others, and in such cases we would like to distinguish between queries with tractable and intractable certain answers. The distinction is often achieved via dichotomy theorems that classify the complexity of the certain answers of *every* query in some large family of queries.

We will illustrate the possible-world semantics in four different settings. While not exhaustive, these four settings represent substantially different mechanisms for forming possible worlds. In the setting of *data exchange*, logical inference rules are used to form the possible worlds: we have a database under one schema that has to be transformed into a database under a different schema, and the possible worlds are the candidate solutions according to rule-based constraints specifying the relationship between the two schemas. In the setting of *inconsistent databases*, minimal intervention is used to form the possible worlds: we have a database that violates integrity constraints, and the possible worlds

IMAGE BY ANDRIJ BORYS ASSOCIATES. USING SHUTTERSTOCK

are the different ways to repair the inconsistent database. In the setting of *probabilistic databases*, we have a representation of a probability distribution over the collection of all databases. Here we focus on the most studied model, the *tuple-independent databases*: each record has a confidence value that is interpreted as a probability, and different records are probabilistically independent. Finally, in the setting of *election databases*, completions of partial orders are used to form possible worlds: we have ordinary databases that describe information about voters, candidates, and winners in elections, but only partial knowledge about voter preferences is known, thus there is uncertainty about who the winning candidates are. Note that the framework of election databases augments computational social choice with relational database context. In particular, it generalizes the *necessary winners* and the *possible winners* problems studied by researchers in computational social choice, because we can now ask for the certainty or possibility of *properties* of the winners (for example, their partisan affiliation or position on some issue) and not just their mere identity (for example, whether candidate *a* or candidate *b* is a winner).

In each setting considered here, we use the lens of computational complexity to highlight the algorithmic aspects of aggregating the answers to conjunctive queries, an extensively studied class of queries that form the core of most relational database queries.

An online appendix (available at https://dl.acm.org/doi/10.1145/3624717) contains additional discussion about topics not covered in this article and an expanded list of references.

Basic Notions

Relational databases and queries. We start with some terminology from relational databases. A *schema* **S** is a collection of *relation symbols R*, each with an associated arity (number of columns). A database over the schema **S**, also called an *instance of* **S** (or just an *instance* if **S** is clear from the context), associates with each relation symbol R a finite relation r with the same arity as R. We identify an instance I with its finite set of *facts* $R(c_1, \cdots, c_k)$, where each such fact asserts that the tuple $(c_1, \cdots, c_k)$ belongs to the relation r of I associated symbol R. In this case, we say that $R(c_1, \cdots, c_k)$ is an R-fact of I. We write $J \subseteq I$ to denote that for every relation symbol R, every R-fact of J is also an R-fact of I. We also view a database as a finite relational structure in the sense of mathematical logic (for example, see Abiteboul et al.[1]).

A k-ary *query* q over a schema **S** maps every instance I of **S** into a k-relation $q(I)$. Every tuple of $q(I)$ is referred to as an *answer*. As a special case, a query of arity zero is viewed as a *Boolean query* that is either *true* (has the empty tuple as an answer) or *false* (has no answers) on the given instance I. In logical terms, a Boolean query is simply a Boolean statement (*true* or *false*) about the database. When analyzing a query, we typically ignore the database schema and, without loss of generality, assume that it consists of the relation symbols that are mentioned in the query formulation.

We will focus on the class of *conjunctive queries*, also known as *select-project-join* queries. They are at the core of the most frequently asked queries on relational databases, and are directly supported by SQL. For example, consider the schema with the binary relations Country(*id, name*) and Borders (*id1, id2*), and consider also the following query that finds the neighboring countries of France:

```
SELECT B.id2
FROM Country C, Borders B
WHERE C.id = B.id1 and
      C.name = "France"
```

This query is expressed as a conjunctive query using the formula

$$\exists y[\mathsf{Country}(y, \text{'France'}) \wedge \mathsf{Borders}(y, x)] \quad (1)$$

More generally, a conjunctive query is a logical formula of the form

$$\exists \mathbf{y}\left[\phi_1(\mathbf{x},\mathbf{y}) \wedge \cdots \wedge \phi_k(\mathbf{x},\mathbf{y})\right]$$

where $\mathbf{x}$ is a sequence of free variables, disjoint from $\mathbf{y}$ (the sequence of existentially quantified variables), and each $\phi_i(\mathbf{x},\mathbf{y})$ is an *relational atom*, that is, an expression of the form $R(\tau_1, \cdots, \tau_k)$ where R is a k-ary relation symbol and each τ_j is either a constant or a variable from $\mathbf{x}$ or $\mathbf{y}$. We refer to each $\phi_i(\mathbf{x},\mathbf{y})$ as an *atom* of the query. Semantically, an *answer* is a tuple $\mathbf{a}$ that makes the formula true when assigned to $\mathbf{x}$. We also assume that every variable in $\mathbf{x}$ occurs in at least one atom—this is a standard safety assumption. A Boolean conjunctive query has no free variables, hence it has the form $\exists \mathbf{y}\left[\phi_1(\mathbf{y}) \wedge \cdots \wedge \phi_k(\mathbf{y})\right]$. A conjunctive query has a *self-join* if two different atoms use the same relation symbol R. For example, the query of Equation (1) has no self-joins, but it would if the atom $\mathsf{Country}(x, x')$ is added for retrieving the id's and the names of the returned countries. Some of the results will require the assumption that the considered queries have no self-joins.

Possible worlds and certain answers. The standard query evaluation problem involves a query q and an instance I, and the problem is to compute the answer $q(I)$ of q on I. We explore settings where query evaluation involves a query and a collection of databases, instead of a single database. These databases can be thought of as *possible worlds* on which a query can be evaluated. What is the semantics of a query posed on different worlds? Clearly, we need a notion that provides a method for aggregating the answers obtained on each possible world. The *certain* answers is the most extensively studied such notion, where intersection is used as the aggregation method.

Definition 2.1.
Let q be a k-ary query and let $\mathscr{W}$ be a collection of databases. The *certain answers* of q on $\mathscr{W}$ is the set

$$\text{CERTAIN}(q, \mathscr{W}) = \bigcap_{J \in \mathscr{W}} q(J) = \{\mathbf{a} : \mathbf{a} \in q(J), \text{ for every } J \text{ in } \mathscr{W}\}.$$

We will discuss the cost of computing the certain answers. On the face of their definition, computing the certain answers requires evaluating the query on each possible world. This may be a formidable task because the number of possible worlds can be very large or even infinite. In several different settings, however, the collection $\mathscr{W}$ of the possible worlds admits a compact representation $\mathscr{R}$. This suggests a different approach: compute the certain answers of a query on $\mathscr{W}$ by working with the compact representation $\mathscr{R}$, instead of examining each possible world in $\mathscr{W}$.

We assume familiarity with basic notions in complexity theory, such as

the complexity classes P, NP, coNP, and #P, and the meaning of hardness and completeness for complexity classes.[18] We follow the convention of analyzing database problems in terms of their *data complexity*, which means that the input consists of the instance I while everything else, such as the schema and the query, are fixed.[35] In particular, when we analyze computational problems associated with queries, each query gives rise to a separate computational problem, and this problem might be tractable for one query and intractable for another query.

Nulls and Data Exchange

In many settings, the possible worlds under consideration involve the use of *null values* that, intuitively, are placeholders for missing values. Database systems supported null values from the very beginning. Moreover, the pioneering work of Imielinski and Lipski[20] gave semantics to null values by means of possible worlds. Their framework introduced the concept of *marked nulls*, later called *labeled nulls*, that behave like variables: two occurrences of the same labeled null represent the *same* missing value, while distinct labeled nulls represent missing values that may be different or equal.

We will use the symbols $N_1, N_2, \ldots$ to denote labeled nulls. An *incomplete* database I is a collection of relations $R_1^I, \ldots, R_k^I$ such that each tuple $(a_1, \ldots, a_n)$ in each relation R_i^I of I consists of constants and labeled nulls, that is, each a_j is a constant or a labeled null. Let $\mathcal{W}(I)$ be the collection of all databases obtained from I by replacing each labeled null by some constant. Therefore, the incomplete database I is a compact representation of the possible worlds in $\mathcal{W}(I)$. In this scenario, it is not hard to prove that the certain answers of a conjunctive query on $\mathcal{W}(I)$ can be computed by evaluating the query directly on the compact representation I, where each labeled null in I is treated as a distinct actual value (constant). More precisely, $\text{CERTAIN}(q, \mathcal{W}(I)) = q(I)_{\downarrow}$, where $q(I)_{\downarrow}$ is the set of of all null-free tuples of $q(I)$. For more details, Abiteboul et al.[1]

In the SQL standard, nulls are not labeled, that is, there is a single null each occurrence of which should be viewed as a distinct labeled null. It should be noted that SQL queries do not adopt the certain-answer semantics, but rather the semantics of three-valued logic, where every comparison to a null value results in "unknown" rather than "true" or "false." There has been a large body of work, such as Libkin's,[28] in which the SQL semantics are compared and contrasted with the certain-answer semantics.

The preceding uses of nulls have been extensively studied, and we will not focus on these here. Instead, we will focus on the use of labeled nulls in *data exchange*, which is the problem of transforming data structured under a *source* schema $\mathbf{S}$ into data structured under a different *target* schema $\mathbf{T}$. To be able to generate meaningful solutions to the data exchange problem, we may need to invent new values that are not in the source database; this is achieved by using labeled nulls that mark entries as invented values, instead of actual known values. Still, there might be many different solutions, and we view each solution as a possible world. In particular and unlike the possible worlds in the framework of Imielinski and Lipski, each possible world may include labeled nulls (that are treated as ordinary values by database queries).

Data exchange is formalized using *schema mappings*, that is, triples of the form $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$, where Σ is a set of constraints that specify the data transformation from $\mathbf{S}$ to $\mathbf{T}$.[14] If I is a source instance, then a *solution* for I with respect to $\mathcal{M}$ is a target instance J such that the pair (I,J) satisfies the constraints in Σ. The active domain of I consists entirely of constants, while the active domain of J may contain both constants and labeled nulls. If the schema mapping $\mathcal{M}$ is understood from the context, we simply talk about a solution for I, instead of a solution for I with respect to $\mathcal{M}$.

If $\mathcal{M}$ is a schema mapping and I is a source instance, we write $\mathcal{W}(\mathcal{M},I)$ to denote the set of all solutions for I. In general, the constraints in Σ may over-specify or under-specify the data transformation task, hence $\mathcal{W}(\mathcal{M},I)$ may be the empty set or a non-empty finite set or an infinite set. If q is a query over the target schema $\mathbf{T}$, we can consider the certain answers of q on $\mathcal{W}(\mathcal{M},I)$, that is, the set

$$\text{CERTAIN}(q, \mathcal{W}(\mathcal{M},I)) = \bigcap_{J \in \mathcal{W}(\mathcal{M},I)} q(J).$$

The pair $(\mathcal{M},I)$ is a compact representation of the space $\mathcal{W}(\mathcal{M},I)$ of all solutions for I. Thus, the question arises: can this compact representation be used to compute the certain answers of target queries, and if so, how? As shown in,[14] there are many natural settings in which the pair $(\mathcal{M},I)$ is used to compute a *universal solution* for I, a special type of solution for I with the property that the certain answers of every conjunctive query can be obtained by directly evaluating the query on the universal solution.

By a definition, a *universal* solution for I is a solution J for I such that for every solution J' for J, there is a homomorphism h from J to J' that is the identity on the constants in J. Intuitively, a universal solution for I is a "most general" solution for I because it has no more and no less information than what is encapsulated in the pair $(\mathcal{M},I)$. Universal solutions can be used to obtain the certain answers

Figure 1. Illustration of the certain answers.

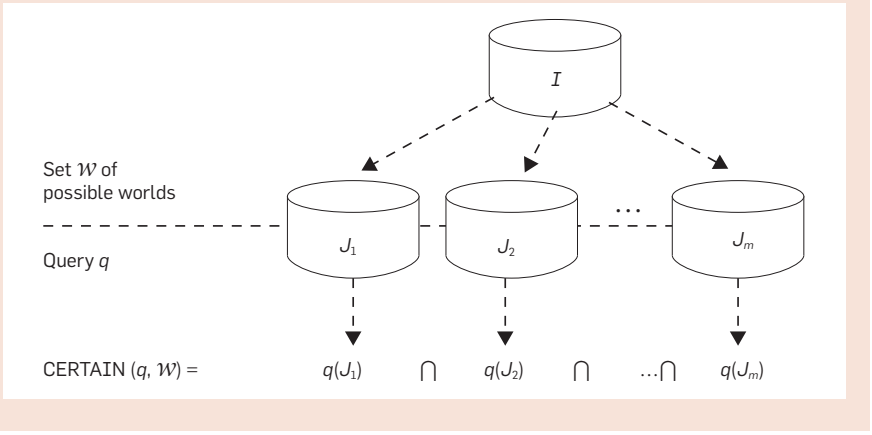
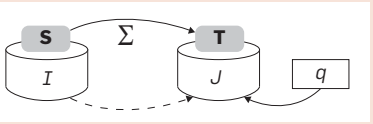


Figure 2. Illustration of a schema mapping.



of conjunctive target queries. More precisely, let $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ be a schema mapping, I a source instance, and q a target conjunctive query. If J is a universal solution for I, then $\text{CERTAIN}(q, \mathcal{W}(\mathcal{M}, I)) = q(J)_{\downarrow}$, where $q(J)_{\downarrow}$ is the set of of all null-free tuples of $q(J)$. The proof follows from the definitions and the fact that conjunctive queries are preserved under homomorphisms.

When do universal solutions exist? To answer this question, we also need to address a different question: what are good schema-mapping specification languages, that is, languages for expressing the constraints used in schema mappings? Ideally, a good schema-mapping specification language should strike a balance between high expressive power and tame computational behavior. From top down, if arbitrary first-order formulas are allowed to specify schema mappings, then it is easy to show that there are simple conjunctive queries for which computing the certain answers is an undecidable problem. From bottom up, let us consider some basic data transformation tasks that every schema-mapping specification language ought to support. In each case, we first describe the transformation and then give a first-order formula expressing an example of it.

- Project a column from a source relation and copy the result to a target relation; in symbols, $\forall \mathbf{x} \forall y(P(\mathbf{x}, y) \rightarrow R(\mathbf{x}))$.
- Decompose a source relation into two target relations; in symbols, $\forall x, y, z\,(P(x, y, z) \rightarrow R(x, y) \wedge Q(y, z))$.
- Add a column to a source relation and copy the result to a target relation; in symbols, $\forall \mathbf{x}(P(\mathbf{x}) \rightarrow \exists z R(\mathbf{x}, z))$.
- Join two source relations and copy the result to a target relation; in symbols, $\forall x, y, z(E(x, y) \wedge F(y, z) \rightarrow H(x, y, z))$.

The formulas expressing the preceding data transformations are first-order formulas of the form $\forall \mathbf{x}(\phi(\mathbf{x}) \rightarrow \exists \mathbf{y} \psi(\mathbf{x}, \mathbf{y}))$, where $\phi(\mathbf{x})$ is a conjunction of atoms with variables from $\mathbf{x}$ and $\psi(\mathbf{x}, \mathbf{y})$ is a conjunction of atoms with variables from $\mathbf{x}$ and $\mathbf{y}$. These formulas are known as *tuple-generating dependencies* (tgds) and constitute one of the broadest classes of integrity constraints in databases.[15] In fact, the formulas that we used to express these data transformations belong to a special class of tgds called *source-to-target tgds* (s-t tgds), because the atoms in the antecedent $\phi(\mathbf{x})$ of each tgd are over the source schema, while the atoms in the consequent $\psi(\mathbf{x}, \mathbf{y})$ are over the target schema.

In what follows, we will consider schema mappings $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ where Σ always contains s-t tgds. In addition, Σ may include *target tgds* and *target equality-generating dependencies* (target egds). Target tgds are tgds of the form $\forall \mathbf{x}(\phi(\mathbf{x}) \rightarrow \exists \mathbf{y} \psi(\mathbf{x}, \mathbf{y}))$ in which both $\phi(\mathbf{x})$ and $\psi(\mathbf{x}, \mathbf{y})$ are conjunctions of atoms over the target schema; target egds are first-order formulas of the form $\forall \mathbf{x}\left(\phi(\mathbf{x}) \rightarrow x_i = x_j\right)$, where $\phi(\mathbf{x})$ is a conjunction of atoms over the target schema and x_i, x_j are two distinct variables from $\mathbf{x}$. Target tgds and target egds are the most extensively studied class of integrity constraints over a database schema.[15] As special cases, they contain the functional dependencies and the *full* tgds, which are tgds with no existential quantifiers in the consequent.

Let $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ be a schema mapping and let q be a Boolean target conjunctive query.

- The *existence-of-universal-solutions* problem $\text{EoU}(\mathcal{M})$ asks: given a source instance I, is there a universal solution J?
- The *certain answers* problem $\text{CERTAINTY}(q, \mathcal{M})$ asks: given a source instance I, is $\text{CERTAIN}(q, \mathcal{W}(\mathcal{M}, I))$ true?

The complexity of these two decision problems exhibits diverse behavior, which depends on the constraints allowed in the schema mappings. Figure 3 summarizes the precise results in three parts. Next, we discuss the main ideas behind the proofs of each part.

The undecidability results in the first part are obtained via a reduction from the *embedding problem* for finite semigroups: given a finite partial semigroup, can it be extended to a finite semigroup? (see Kolaitis et al.[24] for the details).

The main tool for the tractability results in the second part is the *chase procedure*, a versatile algorithm originally designed to study the *implication problem* for database dependencies.[2,7,29] Intuitively, when an instance is chased with a set of tgds and egds, we consider all instantiations of variables that satisfy the antecedents of the constraints and then generate new facts as needed to satisfy the consequents of the constraints. For example, if the instance consisting of $E(1, 2)$ is chased with the tgd $\forall x, y(E(x, y) \rightarrow \exists z(H(x, z) \wedge H(z, y)))$, then the chase procedure will generate a labeled null N_i and the facts $H(1, N_i)$ and $H(N_i, 2)$. With egds, the chase will attempt to equate the values of the two variables in the consequent of the egd by replacing a labeled null by a constant or one labeled null by another, but it will *fail* if it has to equate two different constants.

In general, the chase procedure may not terminate. There are, however, broad sufficient conditions guaranteeing the termination of the chase. In particular, if the set of the target tgds in the schema mapping considered obeys certain structural conditions, such as *weak acyclicity*, then the chase procedure terminates within time bounded by a polynomial in the size of the input structure and produces a universal solution precisely when a solution ex-

Figure 3. Complexity of the existence of universal solutions (EoU) and the certain answers (CE) problems for different types of schema mappings.

Theorem 3.1. The following statements are true.

(1) There is a schema mapping $\mathcal{M} = (S, T, \Sigma)$ and a target conjunctive query q such that Σ consists of s-t tgds, target tgds and target egds, and $\text{EoU}(\mathcal{M})$ and $\text{CERTAINTY}(q, \mathcal{M})$ are undecidable.

(2) If $\mathcal{M} = (S, T, \Sigma)$ is a schema mapping such that Σ consists of s-t tgds, full target tgds, and target egds, then $\text{EoU}(\mathcal{M})$ is solvable in polynomial time. Also, for every target conjunctive query q, we have that $\text{CERTAINTY}(q, \mathcal{M})$ is solvable in polynomial time.

(3) If $\mathcal{M} = (S, T, \Sigma)$ is a schema mapping such that Σ consists of s-t tgds, then EoU(M) is trivial, i.e., every source instance has a universal solution. Also, for every target conjunctive query q, we have that $\text{CERTAINTY}(q, \mathcal{M})$ is SQL-rewritable; in fact, there is a union q' of source conjunctive queries such that $q'(I) = \text{CERTAIN}(q, \mathcal{W}(\mathcal{M}, I))$ for every I.

ists.[14] Every set of full tgds is weakly acyclic, while the set consisting of the target tgd $\forall x,y(H(x,y) \rightarrow \exists z H(y,z))$ is not. Actually, if the instance consisting of the fact $H(1,2)$ is chased with this tgd, then the chase procedure will not terminate as it will generate an infinite sequence $H(2,N_1),H(N_1,N_2),H(N_2,N_3),\ldots$ of facts.

As regards the rewritability results in the third part, if a schema mapping $\mathcal{M} = (\mathbf{S},\mathbf{T},\Sigma)$ is specified using s-t tgds only, then, given a source instance I, the chase procedure always terminates and produces a universal solution J for I in time bounded by a polynomial in the size of I. Therefore, by the properties of universal solutions. J can be used to compute the certain answers $\text{CERTAIN}(q,\mathcal{W}(\mathcal{M},I))$, where q is a target conjunctive query, in time bounded by a polynomial in the size of I. In this case, however, the certain answers can also be obtained by rewriting q to a union q' of conjunctive queries over the source schema and then directly evaluating q' on I. This is achieved by "decomposing" Σ into a set of *local-as-view* (LAV) constraints and a set of *global-as-view* (GAV) constraints, where a LAV constraint is a tgd whose antecedent consists of a single atom and a GAV constraint is a tgd whose consequent consists of a single atom. The rewriting of q into q' can then be achieved by combining the MiniCon algorithm for LAV constraints[31] with an *unfolding* technique for GAV constraints.

For further reading about data exchange, see the monograph[5] and the collection.[24]

Inconsistent Databases

In designing databases, one specifies a schema **S** and a set Σ of integrity constraints on **S**. An *inconsistent database* is a database I that does not satisfy Σ. Inconsistent databases arise in a variety of contexts and for different reasons, including the following two. First, a database system may not support the enforcement of a particular type of integrity constraints. Second, data at different sources may be transferred to a central repository and the resulting database may be inconsistent, even though each source database is consistent.

The framework of database *repairs*, introduced by Arenas, Bertossi, and Chomicki,[5] is a principled approach to coping with inconsistent databases and without "cleaning" dirty data first. Database repairs constitute a mechanism for representing possible worlds through *interventions* in the database. Specifically, given an inconsistent database, the aim is to achieve consistency through some minimal intervention in the database. There may exist, however, many different such minimal interventions, and it may not be known which one represents the ground truth. Hence, the result of each such minimal intervention is viewed as a possible world.

Let Σ be a set of integrity constraints and let I be an inconsistent database. Intuitively, a database J is a *repair of I w.r.t. Σ* if J is a consistent database (that is, $J \models \Sigma$), and J differs from I in a "minimal" way. Several different types of repairs have been considered, including *set-based repairs* (*subset, superset, $\oplus$-repairs*), *cardinality-based repairs*, *attribute-based repairs*, and *preferred repairs*.

Definition 4.1.
Let Σ be a set of integrity constraints and let I be an inconsistent database. A database J is a subset-repair of I w.r.t. Σ if J is a maximal consistent sub-database of I, that is, the following conditions hold:
(a) $J \subset I$;
(b) $J \models \Sigma$;
(c) there is no database J' such that $J' \models \Sigma$ and $J \subset J' \subset I$, where $\subset$ denotes proper containment.

In what follows, we will focus on subset repairs and so from now on we will use the term *repair*, instead of the term *subset repair*.

Example 4.2.
Let Σ be the set consisting of the key constraint
$\forall x\forall y\forall z((R(x,y) \wedge R(x,z) \rightarrow y = z)$.
The inconsistent database
$I = \{R(a_1,b_1),R(a_1,b_2),R(a_2,b_1),R(a_2,b_2)\}$ has four repairs:
- $J_1 = \{R(a_1,b_1),R(a_2,b_1)\}$;
- $J_2 = \{R(a_1,b_1),R(a_2,b_2)\}$;
- $J_3 = \{R(a_1,b_2),R(a_2,b_1)\}$;
- $J_4 = \{R(a_1,b_2),R(a_2,b_2)\}$.

A straightforward generalization of this example shows that an inconsistent database may have *exponentially* many repairs.

Definition 4.3.[5]
Let Σ be a set of integrity constraints, let q be a query, and let I be a database. The *consistent answers of q on I w.r.t. Σ* is the set

$$\text{CONS}(q,I,\Sigma) = \bigcap\{q(J) : J \text{ is a repair of } I \text{ w.r.t. } \Sigma\}.$$

The consistent answers *are* certain answers, where the *possible worlds* are the *repairs*. Indeed, if Σ is a set of constraints and I is a database, let $\mathcal{W}(I)$ be the set of all repairs of I w.r.t. Σ. Then $\text{CONS}(q,I,\Sigma) = \text{CERTAIN}(q,\mathcal{W}(I))$.

Let us revisit the preceding Example 4.2, where we have:

$$\Sigma = \{\forall x\forall y\forall z((R(x,y) \wedge R(x,z) \rightarrow y = z)\}$$
$$I = \{R(a\,1,b\,1),R(a\,1,b\,2),R(a\,2,b\,1),R(a\,2,b\,2)\}$$

- If $q(x)$ is the query $\exists y R(x,y)$, then $\text{CONS}(q,I,\Sigma) = \{a_1,a_2\}$.
- If $q(x)$ is the query $\exists z R(z,x)$, then $\text{CONS}(q,I,\Sigma) = \emptyset$.

Let Σ be a set of integrity constraints and let q be a Boolean query. CERTAINTY(q,Σ) is the following decision problem: Given a database I, is $\text{CONS}(q,I,\Sigma)$ true? (In other words, is q true on every repair of I?) We will examine the complexity of CERTAINTY(q,Σ) in a basic setting, namely, when q is a Boolean conjunctive query and Σ is a set of key constraints with one key per relation.

We first observe that in this setting the *repair-checking* problem w.r.t. Σ is in P, where this problem asks: given two databases I and J, is J a repair of I w.r.t. Σ? From this observation and the relevant definitions, it follows that if Σ is a set of key constraints with one key constraint per relation and q is a Boolean conjunctive query, then CERTAINTY(q,Σ) is in coNP.

Let Σ be the following set of constraints asserting that the binary relations R and S have their first attribute as key:

$$\Sigma = \{R(u,v) \wedge R(u,w) \rightarrow v = w,\ S(u,v) \wedge S(u,w) \rightarrow v = w\}.$$

Consider the following two Boolean queries:

- q_1 is the *path* query $\exists x,y,z(R(x,y) \wedge S(y,z))$.
- q_2 is the *sink* query $\exists x,y,z(R(x,y) \wedge S(z,y))$.

The question now arises: what can we say about the computational complexity of CERTAINTY(q_i, Σ), $i = 1, 2$? Fuxman and Miller[17] showed that CERTAINTY(q_1, Σ) is in P, whereas CERTAINTY(q_2, Σ) is coNP-complete.

Let's explain why CERTAINTY(q_1, Σ) is in P. In fact, a stronger result holds: CERTAINTY(q_1, Σ) is *FO-rewritable*, which means that there is a first-order definable query q^* such that for every database I, we have CONS(q_1, I, Σ) is true if and only if $q^*(I)$ is true.

Concretely, it is easy to verify that q^* is the query

$$q^* \equiv \exists x, y, z(R(x,y) \wedge S(y,z) \wedge \forall y'(R(x,y') \rightarrow \exists z' S(y',z'))).$$

From a complexity standpoint, FO-rewritability is the most desirable outcome: we can compute the consistent answers by directly evaluating a FO-query on the inconsistent database. Furthermore, FO-rewritability implies that CERTAINTY(q_1, Σ) is in P.

We show CERTAINTY(q_2, Σ) is coNP-complete via a polynomial-time reduction from the (complement of) MONOTONE SAT, which is the restriction of SAT to *monotone* formulas, that is, to Boolean formulas in conjunctive normal form such that each clause (conjunct) of the formulas is either a disjunction of variables (*positive* clause) or a disjunction of negated variables (*negative* clause).

Given a monotone formula ϕ, construct a database I with relations R and S: if c is a positive clause and the variable v occurs in c, then put (c, v) in R; if c is a negative clause and v occurs in c, then put (c, v) in S. It is now easy to verify that ϕ is satisfiable if and only if CONS(q_2, I, Σ) is false (that is, there is a repair J of I such that $q(J)$ is false). Thus, CERTAINTY(q_2, Σ) is coNP-complete.

Next, consider the following Boolean query:

- q_3 is the *cycle* query $\exists x, y(R(x,y) \wedge S(y,x))$.

Wijsen[36] showed that the problem CERTAINTY(q_3, Σ) is in P, but it is *not* FO-rewritable. Thus, FO-rewritability is *not* the only way in which the consistent answers of a query can be tractable.

In summary, we have seen that the following statements are true.

- CERTAINTY(q_1, Σ) is in P; in fact, it is FO-rewritable.
- CERTAINTY(q_3, Σ) is in P, but it is *not* FO-rewritable.
- CERTAINTY(q_2, Σ) is coNP-complete.

The trifurcation in the complexity of CERTAINTY(q_i, Σ), for $i = 1, 2, 3$, is not an isolated phenomenon, but an instance of a deeper result, depicted in Figure 4, about conjunctive queries without self-joins. In that result, the boundaries of the trichotomy are delineated with the help of a graph, called the *attack* graph, which is associated with Σ and q. This graph has two types of edges, *regular* edges and *strong* edges. If the attack graph is acyclic, then CERTAINTY(q, Σ) is FO-rewritable. If the attack graph is cyclic, but no cycle contains a strong edge, then CERTAINTY(q, Σ) is in P but it is not FO-rewritable. Finally, if the attack graph has a cycle containing at least one strong edge, then CERTAINTY (q, Σ) is coNP-complete.

The Koutris-Wijsen Dichotomy Theorem vastly generalizes several different earlier dichotomy results, including the one in[25] that classifies the queries that consist of two atoms over two distinct relations. Classifying of the complexity of CERTAINTY(q, Σ) for arbitrary Boolean conjunctive queries (including those with self-joins) remains an open problem.

For further reading about inconsistent databases and consistent answers, see the monograph.[8]

Probabilistic Databases

A *probabilistic database* is a probability space over possible worlds (that is, ordinary databases). Restricting to the discrete case, which is the most commonly studied, we formally define a probabilistic database over a schema $\mathbf{S}$ as a pair $(\mathscr{P}, p)$ where $\mathscr{P}$ is a countable set of databases over $\mathbf{S}$, and is, $\sum_{J \in \mathscr{P}} p(J) = 1$. The conventional semantics of query evaluation over a probabilistic database asks for the marginal probability of every answer. For a Boolean query, this amounts to computing the probability that the query is true on a random possible world. Formally, the task is to compute the probability $\Pr[q]$ that $J \models q$ when J is drawn randomly from $(\mathscr{P}, p)$:

$$\Pr[q] := \sum_{J \in \mathscr{P}, J \models q} p(J)$$

As conventional in probabilistic modeling, we seek compact representations of probabilistic databases, where a finite object represents many possible worlds (for example, exponential number w.r.t. the size of the representation itself). Such representations rely on assumptions of probabilistic independence that, in turn, allow for factorization. The most studied representation is the *Tuple-Independent Database* (TID)[11] (with roots in prior work, for example, Grädel et al.[19]). A TID is represented as an ordinary database where each fact is associated with a probability of existence. Then, a possible world is a subset of the database and it is randomly drawn by independently considering each tuple and deciding whether or not it is valid. More formally, a TID over a schema $\mathbf{S}$ is represented as a pair (I, π) where I is a database over $\mathbf{S}$ and $\pi : I \rightarrow [0, 1]$ assigns a probability to each fact. The pair (I, π) represents the probabilistic database $(\mathscr{P}, p)$ where $\mathscr{P}$ consists of all the subsets of I and

$$p(J) := \prod_{f \in J} \pi(f) \times \prod_{f \in I \setminus J} (1 - \pi(f)).$$

The computational complexity of query evaluation over TIDs has been scrutinized by the database community over the past two decades. As we shall see next, the probability of a query can be #*P*-hard even for very simple join queries. Past results have established thorough classifications of query classes into tractable and intractable, in either the exact or approximate sense. In this part, we explain the first dichotomy found for TIDs[11] that considers the class of conjunctive queries without self-joins. We refer to it as the *Little Dichotomy Theorem* as it was later generalized to the broader class of all *union of* conjunctive queries (or, equivalently, the class of queries that can be phrased using the positive operators of the relational algebra).[12]

Before we give the dichotomy, let us illustrate it. For the positive side, let q_1 be the query $\exists x \exists y(R(x) \wedge S(x,y))$. Given the input TID (I, π), we compute $\Pr[q]$ as follows. First, write q_1 as the expression $\exists x(R(x) \wedge \exists y S(x,y))$. Then, using tuple-independence, the computation can be done as in Figure 5.

The last expression has size $O(n^2)$, where $n = |\mathrm{dom}(I)|$.

An example of an intractable query is $\exists x \exists y(R(x) \wedge S(x,y) \wedge T(y))$, which we denote by q_2. We will show that $\Pr[q_2]$ is #*P*-complete through a reduction from a well known #*P*-complete problem.

A *positive partitioned 2DNF* formula (PP2DNF) is a DNF-formula of the form $(x_{i_1} \wedge y_{j_1}) \vee \cdots \vee (x_{i_k} \wedge y_{j_k})$, where the x_i's and the y_j's form disjoint sets of variables. Counting the satisfying assignments of a PP2DNF formula known to be #*P*-complete.[32] We can visualize a PP2DNF as a bipartite graph, where the left and right sides consist of the x and y variables, respectively, and the edges correspond to the clauses. As an example, the graph of Figure 6a corresponds to the formula $(x_1 \wedge y_1) \vee (x_1 \wedge y_2) \vee (x_2 \wedge y_1)$. If we view a truth assignment to the variables as stating whether a vertex is present or not, then a satisfying assignment corresponds to a non-independent set, that is, a set of vertices with at least one edge. Hence, the number of satisfying assignments is $2^n - M$ where n is the number of variables and M is the number of independent sets. Also, every bipartite graph represents some PP2DNF formula in this way. Thus, counting the satisfying assignments of a PP2DNF formula is the same as counting the independent sets of a bipartite graph.

Dalvi and Suciu[11] showed that this problem can be simulated as the evaluation of q_2 over a TID, as we illustrate in Figure 6a. Let (I, π) be the TID shown in Figure 6b. There is a one-to-one correspondence between truth assignments for ϕ and possible worlds for (I, π). It is easy to verify $\#\phi = 2^n \Pr[q_2]$ where n is the number of variables of ϕ. Hence, we reduce #PP2DNF to $\Pr[q_2]$.

The queries q_1 and q_2 are examples of *hierarchical* and *non-hierarchical* queries, respectively. As it turns out, the property of being hierarchical characterizes tractability. A conjunctive query q without self-joins is *hierarchical* if for every two variables x and y of q, either $\text{at}(x) \subseteq \text{at}(y)$, or $\text{at}(y) \subseteq \text{at}(x)$, or $\text{at}(x) \cap \text{at}(y) = \emptyset$, where $\text{at}(z)$ is the set of all atoms of q where z appears.

For example, our query q_1, namely $\exists x \exists y(R(x) \wedge S(x,y))$, is hierarchical, since $\text{at}(y) \subseteq \text{at}(x)$. Query q_2, namely $\exists x \exists y(R(x) \wedge S(x,y) \wedge T(y))$, is *not* hierarchical since $at(x) \subseteq at(y)$, $at(y) \subseteq at(x)$, and yet, $\text{at}(x)$ and $\text{at}(y)$ have a common member, namely $S(x,y)$.

Figure 4. Trichotomy in the complexity of consistent query answering over conjunctives queries without self-joins.[27]

Theorem 4.4. If Σ is a set of key constraints with one key per relation and q is a Boolean self-join free conjunctive query, then one of the following statements holds:

- Certainty(q, Σ) *is FO-rewritable* (*hence, it is in* P).
- Certainty(q, Σ) *is in* P, *but it is not FO-rewritable.*
- Certainty(q, Σ) *is* coNP-*complete.*

Moreover, this trichotomy is effective: there is an algorithm for deciding, given Σ and q, which of the three cases holds.

Figure 5. Computation example.

$$\begin{aligned}
\Pr[q_1] &= 1 - \prod_{a \in \text{dom}(I)} (1 - \Pr[R(a) \wedge \exists y S(a,y)]) \\
&= 1 - \prod_{a \in \text{dom}(I)} (1 - \Pr[R(a)] \cdot \Pr[\exists y S(a,y)]) \\
&= 1 - \prod_{a \in \text{dom}(I)} \left(1 - \pi(R(a)) \cdot \left(1 - \prod_{b \in \text{dom}(I)} (1 - \pi(S(a,b)))\right)\right)
\end{aligned}$$

Figure 7 depicts the formal theorem that we refer to as the "little dichotomy theorem," stating that being hierarchical characterizes being tractable (under standard complexity assumptions) as far as conjunctive queries without self-joins are concerned.

For further reading about probabilistic databases, we refer the reader to the monograph.[34]

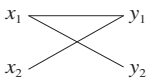
Election Databases

Social choice theory is concerned with the aggregation of individual preferences into a collective outcome. Election databases form a framework that infuses social choice theory with relational database context. An election database may contain information about individual candidates (for example, age, education, wealth), voters (for example, age, occupation, affiliation), positions of candidates on various issues, and preferences of voters. This framework supports the formulation of sophisticated queries that go well beyond asking who the winners in some election are, which is the traditional focus of social choice theory. Ideally, each voter casts a complete preference, which is formalized as a total order of the candidates that the voter has to rank. It is often the case, however, that some voters can only cast incomplete preferences, which are formalized as partial orders of the candidates. Each such partial order represents the possible total rankings of the candidates by a voter, and different total rankings may produce different election outcomes. This constitutes a different mechanism for defining possible worlds.

To present a more rigorous treatment, we first recall the basic terminology of voting theory in computational social choice; for additional background, we refer the reader to the Handbook of Computational Social Choice.[10] Let $C = \{c_1, \ldots, c_m\}$ be a set of *candidates*, and let $V = \{v_1, \ldots, v_n\}$ be a set of voters. A *complete voting profile* is a tuple $\mathbf{T} = (T_1, \ldots, T_n)$, where each T_i is a total order over C, representing the ranking (preference) of voter v_i of the candidates in C. A *voting rule* r is a function that maps every complete profile $\mathbf{T}$ into a set $\mathcal{W}(r, \mathbf{T}) \subseteq C$ of *winners*. We say that a candidate c is a *winner* if $c \in \mathcal{W}(r, \mathbf{T})$.

We focus on the class of *positional scoring rules*, which is arguably the most extensively studied class of voting rules in computational social choice. A positional scoring rule r maps every number m of candidates to a *scoring vector* $(r(m,1), \cdots, r(m,m))$ of natural numbers where $r(m,1) \geq r(m,2) \ldots \geq r(m,m)$. Here, $r(m,j)$ is the score that a candidate is awarded when positioned jth by a voter. The score $s(T_i, c)$ of a candidate c on the order T_i of the voter v_i is $r(m,j)$, where j is the position

Figure 6. (a) Representation of $(x_1 \wedge y_1) \vee (x_1 \wedge y_2) \vee (x_2 \wedge y_1)$ as a bipartite graph. (b) The TID for the bipartite graph.



(a)

R	X	π
	x_1	0.5
	x_2	0.5

S	X	Y	π
	x_1	y_1	1
	x_1	y_2	1
	x_2	y_1	1

T	Y	π
	y_1	0.5
	y_2	0.5

(b)

of candidate c in T_i. When r is applied to $\mathbf{T} = (T_1,\ldots,T_n)$, it assigns to each candidate c the sum $\sum_{i=1}^{n} s(T_i,c)$ as the *score* of c. A candidate is a *winner* if her score is greater or equal to the score of every candidate. Popular positional scoring rules include the *plurality* rule $(1,0,\cdots,0)$ where the winners are the candidates most frequently ranked first, the *veto* rule $(1,\cdots,1,0)$ where the winners are the candidates least frequently ranked last, and the *Borda* rule $(m-1,m-2,\cdots,0)$, where the score of a candidate is the position itself minus 1 in reverse order.

To avoid trivialities, we assume that every $r(\cdot,m)$ contains at least two different score values, that is, $r(m,1) > r(m,m)$. We also assume that the scores in $r(\cdot,m)$ are co-prime (that is, their biggest common divisor is 1), since multiplying all scores by the same number has no impact on the outcome. Rule r is *pure* if for every $m \geq 2$, the scoring vector $r(\cdot,m)$ is obtained from $r(\cdot,m-1)$ by inserting a score value at some position. For example, essentially all specific rules studied in the literature (for example, plurality, veto and Borda) are pure.

A *partial voting profile* is a tuple $\mathbf{P} = (P_1,\ldots,P_n)$, where each P_i is a partial order over the set C of candidates, representing a partially known preference of the voter v_j. A *completion* of $\mathbf{P} = (P_1,\ldots,P_n)$ is a complete voting profile $\mathbf{T} = (T_1,\ldots,T_n)$ such that each T_i is a *completion* of P_i, that is, a total order that completes the partial order P_i. A candidate c is a *necessary winner* if $c \in \mathscr{W}(r,\mathbf{T})$ for all completions $\mathbf{T}$.[26] (We later discuss the analogous notion of a *possible winner*.)

An *election database* consists of a partial voting profile, along with a database I that has contextual information about the candidates. Formally, it is a tuple $(I,C,\mathbf{P},r)$ that represents a set of possible worlds, where each possible world is a *completion* of I. In turn, a completion of I is obtained by taking a completion $\mathbf{T}$ of $\mathbf{P}$, and adding to I a new unary relation WINNER that has the fact WINNER(c) for each winner $c \in \mathscr{W}(r,\mathbf{T})$. In particular, the possible worlds are identical, except for the content of the WINNER relation that depends on the underlying completion of $\mathbf{P}$. A Boolean query q is *necessary* if $J \models q$ for all completions J of I. (Note that "necessary" is the adaptation of the term "certain" to the election setting.)

As an example, let c be a candidate and q be the query WINNER(c). Then q is necessary if and only if c is a necessary winner. As another example, let q' be the query $\exists x[\text{WINNER}(x) \wedge R(x)]$ where R can be thought of as representing a party of candidates. Necessity of q' means that there is necessarily a winner from R. Note that it may be the case that q is necessary even if no candidate in R is a necessary winner, and even if there are no necessary winners at all.

We now discuss the problem of deciding the necessity of a Boolean conjunctive query q under a positional scoring rule r. This problem has two parameters, q and r, and the input consists of a candidate set C, a partial profile $\mathbf{P}$, and a database I.

An immediate observation is that the necessity problem is in coNP. Let us consider several examples of tractability and hardness. The simplest example is the query $\exists x[\text{WINNER}(x)]$ asking whether there exists a winner. By definition, there is always one or more winners, so this query is necessary on every election database (and the problem is trivially solvable in constant time). More interesting examples are the conjunctive queries depicted in Figure 8, namely,

$$\exists x_1,\cdots,x_k[\text{WINNER}(x_1) \wedge \cdots \wedge \text{WINNER}(x_k) \wedge R(x_1,\cdots,x_k)]$$

for different $k \geq 1$, asking whether there are k winners who participate in the k-ary relation R. In the remainder of this section, we discuss the proofs of the complexity results in Figure 8.

We begin with $k = 1$. In the case of plurality and veto, the polynomial-time algorithm is based on a polygamous version of the perfect-matching problem, where each vertex of one side should be matched with a number of neighbors from the other side, and this number is from an interval associated to the vertex. Hardness for the remaining pure rules is via a reduction from the problem of the *possible unique winner*: given a partial profile $\mathbf{P}$ and a member c of the set C of candidates, is there any completion where c is the single winner? Indeed, the complexity of this problem exhibits a classification analogous to the theorem: it is solvable in polynomial time under plurality and veto, yet NP-hard for every other pure rule.[6]

Figure 7. The little dichotomy theorem for probabilistic query answering.

Theorem 5.1 (The Little Dichotomy Theorem). ***Let q be a Boolean self-join free conjunctive query.***

- *If q is hierarchical, then* Pr[q] *is in* P.
- *If q is not hierarchical, then* Pr[q] *is* #P*-complete.*

Figure 8. Complexity of the necessity for different combinations of queries and voting rules.[22]

Theorem 6.1. *For every natural number k, let q_k be the query* $\exists x_1,\ldots,x_k[\text{Winner}(x_1) \wedge \cdots \wedge \text{Winner}(x_k) \wedge R(x_1,\ldots,x_k)]$ *asking whether there is a sequence of co-winners who jointly participate in the relation R.*

- *For $k = 1$, the necessity of q_k is:*
  - *– solvable in polynomial time, if r is plurality or veto;*
  - *– coNP-complete, if r is any other pure rule.*
- *For $k > 1$, the possibility of q_k is coNP-complete for every positional scoring rule.*

more online

An online appendix is available at https://dl.acm.org/doi/10.1145/3624717

For $k = 2$, hardness for plurality and veto is proved via the *maximum independent set* problem, and hardness for any other pure rule is via a reduction from $k = 1$. Hardness for $k > 2$ can be easily established via a reduction from $k = 2$. Interestingly, Kimelfeld et al.[22] used a more complex reduction (from *DNF tautology*) that applies to *all* positional scoring rules, pure or not; hence, for $k > 2$ the necessity problem is coNP-complete for every such rule.

We conclude this section with several comments. First, in the case of the plurality rule, a dichotomy in the complexity has been establish for a large fragment of conjunctive queries.[21] Second, the necessity problem becomes tractable if we assume that the number of candidates is small. In particular, the problem is *fixed-parameter tractable* with respect to the parameter $|C|$ for every polynomial-time query.[22]

Conclusion

The possible world semantics is a natural and principled approach to coping with deficiencies in data. We discussed the versatility of this approach in such deficiencies as missing information (null values), under-specified transformations (data exchange), violation of integrity constraints (inconsistent databases), and quantified uncertainty (probabilistic databases). When possible world semantics is adopted, query evaluation entails the aggregation of the results of the query over the possible worlds. The intersection of the results (certain/consistent answers) amounts to logical inference—these are the answers that are derived from the deficient data. Summation in probabilistic databases amounts to probabilistic inference—each answer is assigned its marginal probability. Other aggregation mechanisms have also been investigated. We touched on the union (possible answers) in passing, but did not cover the *range* semantics for queries with numerical operators, such as COUNT and SUM.[5] There is also work on combinations of settings discussed here; relevant references are given in the online appendix.

The adoption of the possible world semantics increases the computational complexity since the number of possible worlds can be prohibitively large. Indeed, computational hardness already arises for frequently asked queries, such as conjunctive queries; we discussed several dichotomy theorems that strive to delineate the boundary between tractability and intractability for queries in a class. In practice, however, hard problems are often attacked by general-purpose tools, such as integer linear programming, SAT solvers, and answer-set programming, as done in consistent query answering,[13,24] variants of consistent query answering[9,30], and probabilistic databases.[16,33]

Beyond data management, the possible world semantics provides a natural approach to incorporating contextual data in the study of other settings that involve deficiencies. As a case in point, we illustrated this idea by discussing the framework of election databases. We believe the versatility of the possible world semantics can have significant applications in a plethora of other fields. C

References

1. Abiteboul, S., Hull, R. and Vianu, V. *Foundations of Databases*. Addison-Wesley, 1995; http://webdam.inria.fr/Alice/
2. Aho, A.V., Beeri, C. and Ullman, J.D. The theory of joins in relational databases. *ACM Trans. Database Syst. 4*, 3 (1979), 297–314; 10.1145/320083.320091
3. Arenas, M., Barceló, P., Libkin, L. and Murlak, F. *Foundations of Data Exchange*. Cambridge University Press, 2014. http://www.cambridge.org/9781107016163
4. Arenas, M., Bertossi, L.E. and Chomicki, J. Consistent query answers in inconsistent databases. In *Proceedings of the Eighteenth ACM SIGACTSIGMOD-SIGART Symp. on Principles of Database Systems, May 31 - June 2, 1999, Philadelphia, Pennsylvania, USA*. V. Vianu and C.H. Papadimitriou (eds). ACM Press, 1999, 68–79; 10.1145/303976.303983
5. Arenas, M. et al. Scalar aggregation in inconsistent databases. *Theor. Comput. Sci. 296*, 3 (2003), 405–434; 10.1016/S0304-3975(02)00737-5
6. Baumeister, D. and Rothe, J. Taking the final step to a full dichotomy of the possible winner problem in pure scoring rules. *Inf. Process. Lett. 112*, 5 (2012), 186–190.
7. Beeri, C. and Vardi, M.Y. A proof procedure for data dependencies. *J. ACM 31*, 4 (1984), 718–741; 10.1145/1634.1636
8. Bertossi, L.E. *Database Repairing and Consistent Query Answering*. Morgan & Claypool Publishers, 2011.
9. Bienvenu, M., Bourgaux, C. and Goasdoué, F. Computing and explaining query answers over inconsistent DL-lite knowledge bases. *J. Artif. Intell. Res. 64* (2019), 563–644.
10. *Handbook of Computational Social Choice*. F. Brandt et al (eds). Cambridge University Press, 2016; 10.1017/CBO9781107446984
11. Dalvi, N. and Suciu, D. Efficient query evaluation on probabilistic databases. *VLDB J. 16*, 4 (2007), 523–544.
12. Dalvi, N.N. and Suciu, D. The dichotomy of probabilistic inference for unions of conjunctive queries. *J. ACM 59*, 6 (2012), 30:1–30:87.
13. Dixit, A.A. and Kolaitis, P.G. Consistent answers of aggregatio queries via SAT. In *Proceedings of the 38th Intern. Conf. on Data Engineering (ICDE 2022)*. IEEE, 2022, 924–937.
14. Fagin, R., Kolaitis, P.G., J. Miller, R. and Popa, L. Data exchange: Semantics and query answering. *Theor. Comput. Sci. 336*, 1 (2005), 89–124; 10.1016/j.tcs.2004.10.033
15. Fagin, R. and Vardi, M.Y. The theory of data dependencies - A survey. In *Symposia in Applied Mathematics*, 1986, 19–71.
16. Fink, R., Han, L. and Olteanu, D. Aggregation in probabilistic databases via knowledge compilation. *Proc. VLDB Endow. 5*, 5 (2012), 490–501.
17. Fuxman, A. and Miller, R.J.. First-order query rewriting for inconsistent databases. *J. Comput. Syst. Sci. 73*, 4 (2007), 610–635; 10.1016/j.jcss.2006.10.013
18. Garey, M.R. and Johnson, D.S. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.
19. Grädel, E., Gurevich, Y. and Hirsch, C. The complexity of query reliability. In *PODS*. ACM Press, 1998, 227–234.
20. Imielinski, T. and Lipski, W. Jr. Incomplete information in relational databases. *J. ACM 31*, 4 (1984), 761–791; 10.1145/1634.1886
21. Kimelfeld, B., Kolaitis, P.G. and Stoyanovich, J. Computational social choice meets databases. In *IJCAI*. ijcai.org, 2018, 317–323.
22. Kimelfeld, B., Kolaitis, P.G. and Tibi, M. Query evaluation in election databases. In *PODS*. ACM, 2019, 32–46.
23. Dagstuhl follow-ups. *Data Exchange, Integration, and Streams 5*. P.G. Kolaitis, M. Lenzerini and N. Schweikardt (eds). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2013; http://www.dagstuhl.de/dagpub/978-3-939897-61-3
24. Kolaitis, P.G., Panttaja, J. and Tan, W.C. The complexity of data exchange. In *Proceedings of the Twenty-Fifth ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, June 26-28, 2006, Chicago, Illinois, USA*. S. Vansummeren (ed). ACM, 2006, 30–39; 10.1145/1142351.1142357
25. Kolaitis, P.G. and Pema, E. A dichotomy in the complexity of consistent query answering for queries with two atoms. *Inf. Process. Lett. 112*, 3 (2012), 77–85; 10.1016/j.ipl.2011.10.018
26. Konczak, K. and Lang, J. Voting procedures with incomplete preferences. In *Proc. IJCAI-05 Multidisciplinary Workshop on Advances in Preference Handling 20*, 2005.
27. Koutris, P. and Wijsen, J. Consistent query answering for self-join-free conjunctive queries under primary key constraints. *ACM Trans. Database Syst. 42*, 2 (2017), 9:1–9:45; 10.1145/3068334
28. Libkin, L. SQL's three-valued logic and certain answers. In *ICDT (LIPIcs, Vol. 31)*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2015, 94–109.
29. Maier, D., Mendelzon, A.O. and Sagiv, Y. Testing implications of data dependencies. *ACM Trans. Database Syst. 4*, 4 (1979), 455–469; 10.1145/320107.320115
30. Manna, M., Ricca, F. and Terracina, G. Taming primary key violations to query large inconsistent data via ASP. *Theory Pract. Log. Program 15*, 4-5 (2015), 696–710.
31. Pottinger, R. and Halevy, A.Y. MiniCon: A scalable algorithm for answering queries using views. *VLDB J. 10*, 2-3 (2001), 182–198; 10.1007/s007780100048
32. Provan, J.S. and Ball, M.O. The complexity of counting cuts and of computing the probability that a graph is connected. *SIAM J. Comput. 12*, 4 (1983), 777–788.
33. Senellart, P., Jachiet, L., Maniu, S. and Ramusat, Y. ProvSQL: Provenance and probability management in postgresql. In *Proc. VLDB Endow 11*, 12 (2018), 2034–2037.
34. Suciu, D., Olteanu, D., Ré, C. and Koch, C. *Probabilistic Databases*. Morgan & Claypool Publishers, 2011.
35. Vardi, M.Y. The complexity of relational query languages (extended abstract). In *Proceedings of the 14th Annual ACM Symp. on Theory of Computing, May 5-7, 1982, San Francisco, California, USA*. H.R. Lewis, B.B. Simons, W.A. Burkhard and L.H. Landweber (Eds). ACM, 1982, 137–146; 10.1145/800070.802186
36. Wijsen, J. A remark on the complexity of consistent conjunctive query answering under primary key violations. *Inf. Process. Lett. 110*, 21 (2010), 950–955; 10.1016/j.ipl.2010.07.021

Benny Kimelfeld is a professor in the Computer Science Faculty at Technion-Israel Institute of Technology, Haifa, Israel.

Phokion G. Kolaitis is Distinguished Research Professor in the Department of Computer Science and Engineering at the University of California, Santa Cruz, and Principal Research Staff Member at IBM Research, Almaden, CA, USA.

© 2024 Copyright held by the owner/author(s).