Deduction with Contradictions in Datalog *

Serge Abiteboul
INRIA Saclay & ENS Cachan
serge.abiteboul@inria.fr

Daniel Deutch
Tel Aviv University
danielde@post.tau.ac.il

Victor Vianu
UC San Diego & INRIA Saclay
vianu@cs.ucsd.edu

ABSTRACT

We study deduction in the presence of inconsistencies. Following previous works, we capture deduction via datalog programs and inconsistencies through violations of functional dependencies (FDs). We study and compare two semantics for datalog with FDs: the first, of a logical nature, is based on inferring facts one at a time, while never violating the FDs; the second, of an operational nature, consists in a fixpoint computation in which maximal sets of facts consistent with the FDs are inferred at each stage.

Both semantics are nondeterministic, yielding sets of possible worlds. We introduce a PTIME (in the size of the extensional data) algorithm, that given a datalog program, a set of FDs and an input instance, produces a c-table representation of the set of possible worlds. Then, we propose to quantify nondeterminism with probabilities, by means of a probabilistic semantics. We consider the problem of capturing possible worlds along with their probabilities via probabilistic c-tables.

We then study classical computational problems in this novel context. We consider the problems of computing the probabilities of answers, of identifying most likely supports for answers, and of determining the extensional facts that are most influential for deriving a particular fact. We show that the interplay of recursion and FDs leads to novel technical challenges in the context of these problems.

Categories and Subject Descriptors

H.2.3 [**Database Management**]: Languages

General Terms

Theory,Languages,Algorithms

1. INTRODUCTION

The inference of conflicting information is often encountered in contexts such as social networks, where participants often disagree. In this paper, we study a model that is based on datalog, *for deduction*, together with functional dependencies, *whose violation captures conflicting information*. To settle conflicts, we choose nondeterministically between conflicting inferred facts, thereby generating sets of possible worlds. We quantitatively capture the resulting uncertainty using probabilities. As we will see, the interplay of recursion and FDs leads to new challenges for incomplete and probabilistic databases.

We consider *datalog*^{fd}, a language consisting of datalog with FDs imposed on intensional relations, with two previously introduced nondeterministic semantics [1]. These serve as the foundation for our investigation.

The first semantics is of a proof-theoretic nature, based on inferring one fact at a time, while never violating the FDs. The intuition is that a derived fact is rejected if its addition to the database would result in an FD violation. Nondeterminism results from the choice of *one fact* to infer at each stage. A proof theory for demonstrating possible and certain facts was presented in [1], as well as a natural stable model semantics using datalog programs with negation [28].

The second semantics is operational, in the spirit of previous proposals based on fixpoint logic with a witness operator [8], choice in logic programs [30] or repairs [10]; see Related Work section. At each stage, we add a maximal set of immediate consequences that are consistent with the facts that have been inferred so far. Nondeterminism results from the choice of *one consistent set of facts* at each stage. Relating to languages in [8], we show that *datalog*^{fd} with this semantics captures exactly the known class NDB-PTIME of nondeterministic queries computable in polynomial time. We also show that this second semantics is strictly more expressive than the first. We study the complexity of computing possible or certain answers in both cases.

We address the problem of compactly representing the *possible worlds* generated by the two semantics. Such compact representations have been shown to be useful for various practical reasons. We consider here the classical model for representing possible worlds in databases, namely conditional tables (c-tables) [34]. We present a PTIME (data complexity) algorithm for computing a c-table that captures the result of a *datalog*^{fd} program for each of the two semantics, even when the input is also represented as a c-table.

A natural way of quantifying the uncertainty arising from settling contradictions is based on probabilities. For instance, assume that 15 of Bob's friends tell him that Alice is in Paris and 5 claim that she is in London. In the absence of additional information (such as the reliability of each friend), Bob may choose to believe she is in Paris with a probability of 75% and in London with 25%. This is the semantics our model will specify in this simple case. Furthermore, the model provides a general yet simple and natural proba-

*This work has been supported in part by the Advanced European Research Council grant Webdam on Foundations of Web Data Management.

10.5441/002/icdt.2014.17

bilistic semantics for answering queries expressed with *datalog*^{fd} programs, in the presence of contradictions and recursion.

The semantics we use is the probabilistic counterpart of the nondeterministic semantics previously mentioned. In the presence of several options for a next derivation step, we consider them as equiprobable. If a *datalog*^{fd} program is applied to a database instance, a probabilistic database is obtained. To represent probabilistic databases, we consider the model of pc-tables [32], i.e., c-tables with Boolean variables of independent probabilities; see also [48, 20].

This sets the stage for the core of the paper, concerned with reasoning probabilistically in the presence of contradictions. Formally, this corresponds to *the evaluation of* datalog^{fd} *programs over pc-tables.*

First, we consider extending the construction capturing possible worlds with polynomial size c-tables, to the probabilistic setting (using pc-tables). We show that it is possible to represent the result with a polynomial-size pc-table for a fragment of the language: with no recursion and with only one FD per relation. The general case remains open.

We then consider the problem of computing the probability of deriving some particular fact. We show that the introduction of FDs leads to intractability of exact computation even for cases that are otherwise known to be tractable, namely tuple-independent databases [20] and nonrecursive safe queries. We nevertheless show a PSPACE upper bound, as well as a PTIME absolute approximation algorithm via sampling. We show that relative PTIME approximation cannot be achieved unless P=NP.

In presence of contradictions, it may be complicated to understand why particular answers hold. The explanation of answers has already been considered in the context of probabilistic databases in previous works [45, 43, 35], but these did not consider recursion or FDs, that greatly complicate the issue. We study two facets of such explanations:

top-k support Given a derived fact, its *top-k supports* consist of the top-k instances that derive this fact and are the instances with highest probability to do so. We also define a variant of the problem for finding top-k "minimal" subsets of facts (intuitively, each fact is useful for the proof). The problem of computing top-k (minimal) supports is in PTIME in the absence of recursion and FDs, but becomes computationally hard, and even hard to approximate, in the presence of *either*.

influence Given an intensional fact, the *influence* of an extensional fact is measured as the effect of the presence of the extensional fact on the derivation or not of the intensional one. We consider the qualitative measures (in absence of probabilities) of necessity and relevance of facts, and show that the introduction of recursion leads to hardness of deciding them. Then we consider a quantitative measure of influence in presence of probabilities. Surprisingly, while exact computation is already hard without recursion and FDs, we provide an approximation algorithm for the general case.

Novelty and significance. The focus of the paper is on the interplay, in the context of knowledge management, of *recursive* deduction, *nondeterminism* and *uncertainty*. Such a combination arises naturally in distributed systems such as social networks. The peers share and derive opinions in intricate ways, which motivates considering recursive deduction. Nondeterminism and uncertainty arise from the the presence of conflicting opinions, and from the different possible ways of settling conflicts.

Many previous works have addressed some of these three facets, see Section 5. A unified framework was briefly introduced in [1]. We are the first, to our knowledge, to propose an in-depth study combining all three facets. We start in Section 2 by relating the language and semantics of [1] to previous proposals. Although the results are new (besides Proposition 1 that is included for completeness), the main contribution of the section is to set the stage for the remainder of the paper.

Our main contributions are then the following:

- We present in Section 3 the construction of a compact (polynomial size) *representation system* for answers in *recursive* query language with *nondeterministic constructs* (based on functional dependencies). We also prove a completeness result showing that all sets of possible worlds defined by c-tables can be captured by such queries on databases. These results are nontrivial and novel.

- We consider in Section 4 the extension of this model with probabilities, in order to measure the uncertainty, and we investigate computational issues in this setting. We believe that our novel results in this respect are important besides the mere technical difficulty of combining recursion, contradictions and probabilities: probabilities arise naturally from the presence of contradictions. We study the complexity of computing the probability of answers. We propose an approach to explaining answers in such a setting based on top-k support and influence. Providing such explanations is clearly important in order for users to accept systems based on these ideas.

These contributions are essential for understanding reasoning in presence of conflicting opinions, which is central to social networks.

Paper Organization. The paper is organized as follows. In Section 2, we introduce the two semantics of *datalog*^{fd}. The compact representation of possible worlds by c-tables is presented in Section 3. In Section 4, we study the model with probabilities. Related work is described in Section 5. Proofs are omitted due to lack of space.

2. DATALOG WITH FDS

In this section, we present the model used in the paper to capture deduction in presence of contradictions. Deduction is performed using datalog rules, and the notion of contradictions is modeled by violations of functional dependencies over the intensional relations. While the syntax and semantics were already presented in [1], we provide here new expressiveness results. In particular, we compare the expressiveness of the language to previously proposed formalisms [29, 8]. Finally, we also close a gap in the complexity analysis of [1], by considering also the complexity of the non-recursive fragment of the language (the recursive case was addressed in [1]).

We assume the reader is familiar with the standard definitions of relational schema and instance, standard query languages including first-order logic (FO) and datalog, and functional dependencies (FDs for short), see [3]. We assume the existence of infinite, disjoint alphabets *dom* of data values (sometimes called *constants*), *var* of data variables, and *bool* of Boolean variables. A database instance that includes data variables is called an *incomplete instance*. Otherwise, it is a *(complete) instance*. For a datalog program P, the set of relations, extensional relations, and intensional relations in P are respectively denoted $sch(P)$, $ext(P)$, and $int(P)$. An

instance over $ext(P)$ is an input instance for P. We study datalog in the context of FDs:

Definition 1 *A* datalog^{fd} *program is a pair* (P, F) *where* P *is a datalog program and* F *a set of FDs over* $int(P)$.

Example 1 *Consider a schema consisting of ternary relations IsIn*$_0$ *and IsIn and a binary relation Follower. Intuitively IsIn*$_0(x, y, z)$ *(extensional) and IsIn*(x, y, z) *(intensional) both mean "Person* z *thinks that person* x *is in city* y*". We impose the constraint that IsIn satisfies*[1] $1, 3 \rightarrow 2$*. Now consider the following datalog program:*

$$\textit{IsIn}(x, y, z) \text{ :- } \textit{Follower}(z, z'), \textit{IsIn}(x, y, z')$$
$$\textit{IsIn}(carol, y, z) \text{ :- } \textit{IsIn}(alice, y, z)$$
$$\textit{IsIn}(x, y, z) \text{ :- } \textit{IsIn}_0(x, y, z)$$

with an asymmetric relation Follower (in the style of Twitter). The first rule states that each person believes the people he follows about the whereabouts of people, whereas the second states that it is general knowledge that Carol is in the same city as Alice.

Consider the above example. Intuitively, the FD $1, 3 \rightarrow 2$ on *IsIn* places a constraint stating that it is inconsistent for someone to believe that a person is in two different cities. However, it may be the case that someone may infer that the same person is in two places, e.g. Ben that Alice is both in Paris and London. As usual, rather than declaring failure, one would like, while reasoning, to limit the damage and salvage as much information as possible. Clearly, there may be several means of resolving the inconsistencies. Indeed, is is possible to define different semantics for datalog^{fd}, a situation similar to that of languages for non-monotonic reasoning (e.g. datalog with negation) where various semantics can typically be proposed.

Semantics. In [1], two semantics for datalog^{fd} programs are proposed and basic properties were studied. The first, called *fact-at-a-time* semantics, is logically founded, with equivalent *model-theoretic* and proof-theoretic semantics. It is closely related to semantics previously studied in the context of datalog with *choice* (see [29] for a survey) and datalog$^{\neg}$ with stable semantics (see [3] for an introduction). The second semantics, called *set-at-a-time*, is *procedural* in nature. It is in the spirit of the inflationary fixpoint semantics for datalog$^{\neg}$ (see [3]). One motivation for the set-at-a-time semantics is that it naturally leads to more efficient set-at-a-time processing, as supported by relational database engines. Moreover, we will see that this semantics has appealing properties with regard to expressiveness, and is well-suited to our probabilistic framework.

We next present the two semantics and results on their expressiveness and complexity.

Fact-at-a-time semantics. The fact-at-a-time semantics for datalog in presence of FDs is based on *forward chaining* with instantiated rules applied one at a time. Each rule application generates a new candidate fact. This fact is added to the database instance unless its addition violates an FD. Note that this introduces nondeterminism since the result of the process possibly depends on the order of rule activation. We consequently refer to this semantics as *nfat*, standing for *non-deterministic fact-at-a-time* semantics.

We now formally define this semantics, by modifying the immediate consequence and the consequence operators to account for FDs. For simplicity, we assume that there is no FD on the extensional relations. As we will see, the database instances that will be generated will always be, by construction, consistent with the FDs on the intensional relations.

[1]If so desired, we could also require that *IsIn*$_0$ *satisfy* $1, 3 \rightarrow 2$.

Definition 2 *Let* (P, F) *be a* datalog^{fd} *program, and* I *an instance over* $sch(P)$ *satisfying* F*. The* immediate consequence operator *denoted* $\rightarrow_{nfat}$ *(with* (P, F) *understood) is defined by:* $I \rightarrow_{nfat} I \bigcup \{A\}$ *if there exists an instantiation* A *:-* $A_1, ..., A_n$ *of a rule in* P *such that* $\{A_1, ..., A_n\} \subseteq I$ *and* $I \bigcup \{A\} \not\models F$*. The* consequence operator $\rightarrow^*_{nfat}$ *is the transitive closure of the immediate consequence operator.*

The *possible worlds* then correspond to the different ways of settling contradictions.

Definition 3 *Let* (P, F) *be a* datalog^{fd} *program and* I *an instance over* $ext(P)$*. An instance* J *over* $sch(P)$ *is a* possible world *for nfat and* (I, P, F) *if* J *is a maximal instance satisfying* $I \rightarrow^*_{nfat} J$*. And*

$$P_F^{nfat}(I) = \{\Pi_{int(P)}(J) \mid J \textit{ is a possible world for nfat}\}$$

where $\Pi_{int(P)}(J)$ *is the restriction of* J *to int*(P).

Observe that a program (P, F) thereby defines a nondeterministic mapping P_F^{nfat} from instances over the extensional relations to instances over the intensional ones. In particular, a program can be viewed as defining a nondeterministic query by designating one specific intensional relation as the answer. We say that two programs (with the same extensional schema and designated answer relations) are *query equivalent* if they define the same query.

Example 2 *Reconsider the program in Example 1, and assume that the initial database is as follows:*

$$\textit{IsIn}_0(alice, paris, peter), \textit{IsIn}_0(carol, london, tom),$$
$$\textit{Follower}(ben, tom), \textit{Follower}(ben, peter)$$

There are two possible worlds containing:

1) $\textit{IsIn}(alice, london, ben), \textit{IsIn}(carol, london, ben)$, *but not* $\textit{IsIn}(carol, paris, ben)$.
2) $\textit{IsIn}(alice, paris, ben), \textit{IsIn}(carol, paris, ben)$ *but not* $\textit{IsIn}(carol, london, ben)$.

Remark 1 *(datalog rules as soft constraints) Observe a subtlety of the semantics in our use of datalog rules. In the style of, e.g., [8, 37], we interpret rules as constraints that are soft in the following sense: if the body holds, then the head holds as well* unless, together with the FDs, the head contradicts some other fact. *Recall that, in absence of contradictions, a model of a datalog program satisfies each rule of the program seen as a universally quantified first-order formula. In the presence of contradictions, it is no longer possible to impose such a strong requirement.*

As usual, a fact is said to be *certain* if it appears in all possible worlds, and *possible* if it belongs to at least one possible world. In [1], a proof theory is introduced for proving possibility and certainty of facts under the *nfat* semantics, and shown to be sound and complete. The complexity of checking possibility and certainty of a given tuple is discussed at the end of the section.

With the *nfat* semantics, the existence of some tuples in the database forbids the existence of other tuples. This leads to connections to datalog with negation. Indeed, it is shown in [1] that, given a datalog program P and a set F of FDs, one can construct a datalog$^{\neg}$ program $NegProg(P, F)$ with the following property. For every

datalog^{fd} program (P, F) and instance I over $ext(P)$, J is a possible world for (I, P, F) if and only if there is a *stable model* [27] of NegProg(P, F) with respect to input I that coincides with J on $int(P)$. This is reminiscent of the stable model semantics for datalog$^{\neg}$ with a nondeterministic choice construct [47]. An inverse translation from datalog$^{\neg}$ to datalog^{fd} does not follow from the construction in [1], and the question of whether such translation is possible remains open.

Set-at-a-time semantics. The fact-at-a-time semantics may lead to very inefficient inference because tuples are processed one at a time. One may instead adopt *set-at-a-time* inference as follows. At each stage, one infers a maximal subset of the derivable facts that is consistent (with respect to the FDs) with the facts of the original database and the facts inferred so far. We next consider this semantics, denoted *nsat*, and compare it to the earlier *nfat* semantics.

The *nsat* semantics is formalized as follows.

Definition 4 *Let (P, F) be a datalog^{fd} program and I an instance over $sch(P)$. Let A be the set of tuples that can be obtained by applying P on the facts in I (i.e. using in the body of the rules only facts from I, and not newly derived tuples). For every maximal subset A' of A such that $I \bigcup A'$ satisfies F, we say that $I \rightarrow_{nsat} I \bigcup A'$, and denote by $\rightarrow^*_{nsat}$ the transitive closure of this operator.*

Let (P, F) be a datalog^{fd} program and I an instance over $ext(P)$ satisfying F. An instance J over $sch(P)$ is a possible world *for nsat and (I, P, F) if J is a maximal instance satisfying $I \rightarrow^*_{nsat} J$. Also,*

$$P^{nsat}_F(I) = \{\Pi_{int(P)}(J) \mid J \text{ is a possible world for nsat}\}$$

Example 3 *Reconsider Examples 1 and 2, and let us now evaluate the program using nsat semantics. At each iteration, we add a maximal set of derivable facts that does not violate the FDs. Both IsIn(Alice,Paris,Ben) and IsIn(Carol, London, Ben) are derived at the first iteration; but then at the next iteration, IsIn(Carol,Paris,Ben) can no longer be derived since it contradicts IsIn(Carol, London, Ben). In particular, in this example, there is only one possible world under nsat semantics.*

Remark 2 *The basic step in the nsat semantics is to produce, for a given relation, all maximal subsets that satisfy a given set F of FDs. One may wonder if the global maximization can be replaced with sequential maximizations with respect to each FD in F. It turns out, as also observed in [29], that the answer is negative. For instance, consider a binary relation R consisting of the cross product $\{0, 1\} \times \{0, 1\}$, and $F = \{1 \rightarrow 2, 2 \rightarrow 1\}$. There are two maximal subsets of R satisfying F: $\{\langle 0, 0\rangle, \langle 1, 1\rangle\}$ and $\{\langle 0, 1\rangle, \langle 1, 0\rangle\}$. On the other hand, the maximal subsets satisfying $1 \rightarrow 2$ are $\{\langle 0, 0\rangle, \langle 1, 0\rangle\}$, $\{\langle 0, 0\rangle, \langle 1, 1\rangle\}$, $\{\langle 0, 1\rangle, \langle 1, 0\rangle\}$, $\{\langle 0, 1\rangle, \langle 1, 1\rangle\}$. Finally, the maximal subsets of these relations, satisfying $2 \rightarrow 1$, are $\{\langle 0, 0\rangle\}, \{\langle 1, 0\rangle\}, \{\langle 0, 0\rangle, \langle 1, 1\rangle\}, \{\langle 0, 1\rangle, \langle 1, 0\rangle\}$, $\{\langle 0, 1\rangle\}, \{\langle 1, 1\rangle\}$. While all of the relations satisfy F, not all are* maximal *with this property.*

Remark 3 *The* datalog^{fd} *with nsat semantics may be viewed as a fixpoint computation in which repairs are performed after each stage, with the difference with standard repairs that tuples, once asserted, cannot be retracted. One may wonder whether the same effect could be achieved by first applying the datalog program up to saturation and then applying repairs to the result, choosing e.g. a maximal consistent subset. Ths answer is negative: this latter semantics is different from nsat. Moreover, it is counterintuitive in our context, as it involves the "retraction" of facts that may have been used in the derivation of other facts, possibly leading to the presence of "unjustified" facts. This could be avoided by recursively deleting such facts, but would render the semantics harder to understand. The nfat and nsat semantics are by design inflationary, which precludes this problem.*

Comparison of nfat and nsat semantics. We have introduced two semantics for deduction in presence of FDs. We next compare them. We start by noting the connection shown in [1] between the possible and certain facts of a program under the two semantics:

Proposition 1 *[1] For each* datalog^{fd} *program, a possible fact under the nsat semantics is also possible under the nfat semantics. A certain fact for nfat is certain for nsat. The converse statements do not hold.*

We next study in more depth the connection. In particular, we show that one can simulate the *nfat* semantics with *nsat*, i.e. obtain the same set of possible worlds; but the converse does not hold. Note that this *does not follow* from the previous result (which only dealt with possible and certain facts).

We start with the former:

Theorem 1 *For each* datalog^{fd} *program (P, F), there exists a datalog program (P', F') such that $ext(P) = ext(P')$, $int(P) \subseteq int(P')$, and for each input instance I of (P, F), an instance J is a possible world of (P, F) for I under the nfat semantics iff $J = \Pi_{int(P)}(J')$ for some possible world J' of (P', F') of I under the nsat semantics.*

The idea of the simulation is to use an additional intensional relation to ensure that each stage in the *nsat* evaluation adds at most one tuple to the instance, thus simulating the *nfat* semantics.

We next show that the converse simulation is not possible. To do so, we introduce a notion of containment that is adapted to nondeterministic computations. Recall from [3] that a nondeterministic query is a computable and generic (i.e. that it commutes with isomorphisms) mapping that associates to each input instance a set of possible answer instances over the same active domain.

Definition 5 *A nondeterministic query q is* monotone *if for each $I, I', I \subseteq I'$, each fact possible in $q(I)$ is also possible in $q(I')$.*

We can now show the following result.

Theorem 2 *For each* datalog^{fd} *program (P, F) with nfat semantics, the nondeterministic query associating an input I to $P^{nfat}_F(I)$ is monotone. This is not generally the case with nsat semantics.*

Expressiveness of nsat. The language datalog^{fd} is closely related to nondeterministic query languages using the *witness* operator in FO and fixpoint logic. (See the Related Work section also for relations to datalog with *choice*.) We next examine in more detail these connections.

We briefly review the *witness* operator [6, 8]. For an FO formula $\varphi(\bar{u}, \bar{v})$ where $\bar{u}, \bar{v}$ are vectors of distinct variables, $W\bar{v}\varphi(\bar{u}, \bar{v})$ defines all maximal subsets of the set of answers to $\varphi(\bar{u}, \bar{v})$ satisfying the FD $\bar{u} \rightarrow \bar{v}$. The language obtained by augmenting FO with W is denoted FO+W. For technical reasons, it is convenient to

consider an extension of the language consisting of a sequence of assignments of the form $R{:=}\varphi$, where each φ is an FO+W formula using input or previously defined relations. Intuitively, this allows naming and reusing intermediate results obtained nondeterministically, which is not possible in FO+W. The extended language is denoted FO$^+$+W. (Observe that FO$^+$+W is strictly more expressive that FO+W.)

We can show the following:

Theorem 3 *Nonrecursive datalog^{fd} with a single FD per relation and nsat semantics expresses precisely the nondeterministic queries definable in FO$^+$+W.*

In the previous theorem, the language FO$^+$+W suggested the restriction to no more than one FD per intensional relation. It is open whether nonrecursive datalog^{fd} (with more than one FDs per relation) allows expressing strictly more queries than FO$^+$+W. We note that, in the recursive case, several FDs do not yield extra expressive power (see Remark 4 further).

We next consider recursive datalog^{fd}. Recall that for each complexity class C, the set of nondeterministic queries computable by a Turing Machine of complexity C is denoted by NDB-C. In particular, NDB-PTIME denotes the nondeterministic queries computable by a nondeterministic Turing Machine for which every computation is in PTIME. (This is *not* to be confused with the class of NP queries that is a class of deterministic Boolean queries.)

Now, we have:

Theorem 4 datalog^{fd} *with the nsat semantics captures NDB-PTIME. Moreover, this holds even for datalog^{fd} with at most one FD per relation.*

The inclusion in NDB-PTIME is immediate. The proof that each NDB-PTIME query can be expressed by a datalog^{fd} program is similar to the proof used in [29] that datalog with dynamic choice can express NDB-PTIME. However note that a direct translation of the proof of [29] to our settings would involve the use of multiple FDs. Avoiding this, so as to show that the proposition holds even with at most one FD per relation, requires some extra effort.

Remark 4 *Proposition 4 provides a normal form for datalog^{fd} programs with nsat semantics: each datalog^{fd} program is query equivalent to a program using at most one FD per relation.*

Complexity of possibility and certainty. To conclude this section, we establish the complexity of testing possibility and certainty of answers for the *nfat* and *nsat* semantics. The recursive case was already stated in [1], and here we complete the picture by studying the *nonrecursive case*. For the *nfat* semantics, the fact that certainty is coNP-complete while possibility is in PTIME is unusual.

Theorem 5 *(i) For nonrecursive* datalog^{fd} *with nfat semantics, deciding whether a fact in an intensional relation is possible is in PTIME. (ii) For nonrecursive* datalog^{fd} *with nfat semantics, deciding whether a fact in an intensional relation is certain is coNP-hard. (iii) For nonrecursive* datalog^{fd} *with nsat semantics, deciding whether a fact in an intensional relation is possible is NP-complete; deciding whether it is certain is coNP-complete.*

Remark 5 *We note that two equivalent datalog queries may yield different sets of possible worlds, when FDs are incorporated. This is consistent with our view of the FDs as part of the query, and with the intended applications.*

3. REPRESENTATION SYSTEM

We next study the problem of representing the set of possible worlds defined by a datalog^{fd} program on a given instance, for the *nfat* and *nsat* semantics. We will show how to construct in polynomial time a representation using "concrete c-tables" and thus obtain a compact representation of the possible worlds.

Conditional tables. Compact representations have proven to be effective for many applications, leading in practice to better performance even if the theoretical complexity remains the same. They have been considered in a number of contexts such as *incomplete databases* [34], *incremental maintenance* [33], *explaining computations* [14] or *probabilistic databases* [20]. We use here very classically the *conditional tables* of [34], *c-tables* for short. (Note that c-tables are sometimes equipped with global conditions, e.g., in [3].) Let *var* and *dom* be domains of variables and values respectively.

Definition 6 *A* c-table *over schema S is a pair (T, φ) where T is a finite incomplete instance over S (with entries in* var$\cup$dom*), and φ maps each tuple in T to a Boolean combination of equalities of the form $[x = y]$ or $[x = a]$, for $x, y \in$ var and $a \in$ dom.*

The set of possible worlds represented by (T, φ) is defined as follows. Let V be the set of variables occurring in (T, φ). Each valuation ν from V to *dom* generates a possible world T_ν consisting of all tuples $\nu(t)$ where $t \in T$ and $\nu \models \varphi(t)$ (ν is extended to tuples componentwise, with the identity on *dom*). The set of possible worlds defined by (T, φ) is denoted by *rep*(T, φ).

Concrete c-tables. In the contexts of e.g. probabilistic data and explaining query results, a particular class of conditional tables is often used, that we refer to as *concrete c-tables* (see e.g. [20, 33]). These are variants of c-tables in which all tuples are ground (have no variables) and their associated conditions use only Boolean variables (As we shall do in Section 4, in probabilistic settings, independent probabilities are assigned to these Boolean variables.),

Given an instance I and a $datalog^{fd}$ program P, we will show that the set of possible worlds defined by P on I, under both *nfat* and *nsat* semantics, can be described by a small (polynomial in I) concrete c-table. Concrete tables are formally defined as follows.

Definition 7 *A* concrete c-table *is a pair (I, β) where I is an instance with values in* dom *and β maps each tuple in I to a Boolean expression.*

An example of concrete c-table:

			β
Alice	London	Ben	x
Carole	London	Ben	x
Alice	Paris	Ben	$\neg x$
Carole	Paris	Ben	$\neg x$

The set of possible worlds represented by a concrete c-table (I, β) is defined as follows. Let V be the set of variables occurring in (I, β). Each truth assignment ν for V generates a possible world I_ν consisting of all tuples t in I for which $\nu \models \beta(t)$. The set of possible worlds defined by (I, β) is denoted by *rep*(I, β).

Note that a c-table possibly represents an *infinite* set of possible worlds, whereas a concrete c-table always represents a *finite* set of worlds, since each is a sub-instance of the complete instance of the concrete c-table. Let us call c-tables that represent only a finite set of worlds *finitary*. As shown next, concrete c-tables essentially

"capture" finitary c-tables. It is easy to see that each finitary c-table can be represented by a concrete one. The only difficulty is to show that the transformation to a concrete c-table can be achieved with only a polynomial blowup. This will be useful in the proof of Theorem 6.

Proposition 2 *Let S be a fixed schema and (T, φ) a finitary c-table over S. One can construct in polynomial time a concrete c-table (I, β), such that $rep(I, \beta) = rep(T, \varphi) \cup \{\emptyset\}$.*

We next present the two main results of the section, namely that one can represent the result of applying a $datalog^{fd}$ program on a given instance by a polynomial size concrete c-table, with both the *nfat* and *nsat* semantics. (The *size* of a concrete c-table (I, β) is the size of I plus the sum of the sizes of the conditions $\beta(t)$ for t in I.)

Compact representation. Consider first *nfat* semantics.

Theorem 6 *Let (P, F) be a $datalog^{fd}$ program. Given a database instance J over $ext(P)$, one can compute in polynomial time with respect to J a concrete c-table (I, β) over $int(P)$ such that $rep(I, \beta) = P_F^{nfat}(J) \cup \{\emptyset\}$.*

To obtain the desired concrete c-table, it suffices to show how to construct in polynomial time a finitary (non-concrete) c-table (T, φ) over $int(P)$ such that $rep(T, \varphi) = P_F^{nfat}(J) \cup \{\emptyset\}$. (For then we can use Proposition 2.) Intuitively, the incomplete instance T consists of N tuples $t_1, ..., t_N$ whose components are all distinct variables. The conditions φ enforce that: (i) each tuple in T is derived using previously derived tuples or tuples in J; (ii) no FD is violated; (iii) no rule can be applied to the resulting instance in order to generate additional tuples.

An analogous result also holds for the *nsat* semantics.

Theorem 7 *Let (P, F) be a $datalog^{fd}$ program. Given a database instance J over $ext(P)$, one can compute in polynomial time with respect to J a concrete c-table (I, β) such that $rep(I, \beta) = P_F^{nsat}(I) \cup \{\emptyset\}$.*

The proof is similar to that of Theorem 6. A difficulty is the simulation of the set-at-a-time semantics. So, instead of having N tuples, we have N^2 tuples, using N of them for each iteration, knowing that there are at most N iterations.

Theorems 6-7 show that the possible results of a datalog^{fd} program applied to a database instance can be represented by concrete c-tables computable in polynomial time, for both *nfat* and *nsat* semantics. One may wonder whether concrete c-tables form a strong representation system for datalog^{fd}, in the classical sense of [34]. The answer is positive. For this, we extend the semantics of programs to sets of instances in the straightforward way: $P_F^{nfat}(\mathcal{J}) = \cup\{P_F^{nfat}(J) \mid J \in \mathcal{J}\}$ for each set $\mathcal{J}$ of instances. Now, it can be shown that for each concrete c-table (I, β) and datalog^{fd} program (P, F), one can construct in polynomial time a concrete c-table (I', β') such that

$$rep(I', \beta') = P^{nfat}(rep(I, \beta)) \cup \{\emptyset\}.$$

The same holds for *nsat* semantics. The proofs are similar to those of Theorems 6 and 7.

We conclude this section by presenting two results; the first is on the synthesis of table and the second on the restriction to positive conditions.

Synthesizing concrete c-tables. One can view a concrete c-table or a datalog^{fd} program over a complete database as alternative mechanisms for specifying sets of possible worlds. We have seen that the set of possible worlds resulting from applying a datalog^{fd} program to a complete database can be represented as a concrete c-table. Interestingly, the converse also holds, so the two formalisms are essentially equivalent. Indeed, we have the following.

Proposition 3 *For each concrete c-table (I, β) over schema S there exists a nonrecursive datalog^{fd} program (P, F) with $S \subseteq int(P)$, and an instance J over $ext(P)$ such that*

$$rep(I, \beta) = \Pi_S(P_F^{nsat}(J)) = \Pi_S(P_F^{nfat}(J)).$$

Also, J and (P, F) are both linear in the size of (I, β).

Restricting to positive concrete c-tables. Because of applications in probabilistic databases (for which probability computation for positive formulas may be more efficient) and in semiring-based provenance [31] (for which it is necessary), it would be desirable if conditions in the concrete c-tables representing results of datalog^{fd} programs could be restricted to be positive (no negation occurs). Unfortunately, this is not possible, since the use of positive conditions does not allow to capture maximal "repairs" to key violations as required by the semantics.

Proposition 4 *There exists a nonrecursive FD-datalog program (P, F) and a database instance J over $ext(P)$ such that there is no positive concrete c-table (I, β) for which $rep(I, \beta) = P_F^{nfat}(J) \cup \{\emptyset\}$, and similarly for nsat.*

4. PROBABILISTIC SEMANTICS

We consider in this section a semantics based on "measuring" the non-determinism resulting from contradictions using probabilities. We formally define the semantics. We study the problem of constructing a representation system that correctly captures probabilistic results according to that semantics. We also study three essential problems in this context (base facts are facts occurring in the input database; intensional facts are facts derived by a query): How to compute the probability of an intensional fact? How to find the most probable supports of that fact? How to find the most influential base facts for deriving that fact?

4.1 Probabilistic datatog with FDs

We recall that a *probabilistic database* [5] over a schema S is a pair $(\mathcal{I}, \mathcal{P})$, where $\mathcal{I}$ is a set of instances over S and $\mathcal{P} : \mathcal{I} \to [0, 1]$ defines a probability distribution over the instances, i.e., $\Sigma_{I \in \mathcal{I}} \mathcal{P}(I) = 1$.

We can extend the semantics *nfat* and *nsat* to datalog^{fd} programs over probabilistic databases. Doing it for both would be somewhat of a tedious exercise. We therefore focus primarily on *nsat* and its probabilistic extension, that we call *psat* (for probabilistic set-at-a-time). This is the most challenging of the two cases, and as we saw, also the most powerful. We briefly discuss the probabilistic extension *pfat* of *nfat* towards the end of the section.

Starting from some instance I, consider the *nsat* derivation for a particular program P. Suppose that, at some step, we have derived an instance J with some probability p. A number of rule instantiations may allow adding maximal sets of facts to J to obtain instances $J_1, ..., J_n$. Recall that in the *nsat* semantics, each J_i may be obtained from J by multiple *sets* of rule instantiations. In

the absence of other information, we consider all such sets of instantiations equally likely. For example, if J_1 is obtained from J by 3 sets of instantiations and J_2 by 2 sets of instantiations, then the derivation of J_1 is considered more likely. Note that we could have adopted an alternative semantics that would make the derivations of all J_i equiprobable. However, this would be contrary to the intuition that multiple derivations increase confidence in the outcome.

From the previous discussion, the derivation process is equipped with some probability distribution. Thereby, starting from a probabilistic database, instances are derived with certain probabilities. Finally, at the end of the process. we obtain a resulting probabilistic database. The semantics *psat* is formally defined as follows.

Given a datalog^{fd} program (P, F) and a probabilistic database $(\mathcal{I},\mathcal{P})$ over $ext(P)$, we construct inductively a labeled tree. The root of the tree is labeled $(\mathcal{I},\mathcal{P})$. The non-root nodes are labeled by pairs (I, p) with I an instance and p a value from $[0, 1]$. To start, the children of the root node are the set of pairs $(I, \mathcal{P}(I))$ for $I \in \mathcal{I}$. Now suppose that at some point in the construction, there is a leaf node with label (J, p) such that $J \rightarrow_{nsat} J'$ for some J'. Let $J_1, ..., J_n$ be the set of instances that can be derived from J. We call *instantiation set* for J and J_i for some i, a set θ of instantiations of rules in P such that (i) θ generates exactly $J_i - J$, and (ii) two rule instantiations in θ generate different facts. The crux of the construction is the set of instantiation sets:

$$\Theta(J) = \{\theta \mid \theta \text{ instantiation set for } J \text{ and } J_i \text{ for some } i\}$$

The children of (J, p) are $(J_1, p_1) \ldots, (J_n, p_n)$ where for each i, p_i is the product of p by the number of instantiation sets for J and J_i divided by $|\Theta(J)|$. Observe that p is equal to $\Sigma_i p_i$ (and that the "weight" of each J_i is proportional to the number of its instantiation sets).

We continue until all leaf nodes have labels (J', p') such that there exists no J'' with $J' \rightarrow_{nsat} J''$.

Observe that at each step in the construction of the tree, the set of leaves defines a probabilistic database (and in particular the obtained distribution is indeed a probability distribution). We can therefore define the result of applying (P, F) to $(\mathcal{I},\mathcal{P})$, denoted $P_F^{psat}(\mathcal{I}, \mathcal{P})$, as the probabilistic database $(\mathcal{I}', \mathcal{P}')$ where $\mathcal{I}'$ contains all instances J which appear in the label of some leaf node in the completed tree, and $\mathcal{P}'(J)$ sums up the values p appearing in the leaf labels (J, p). (Note that if k leaves have label (J, p), then p is counted k times).

Example 4 *Consider again the datalog^{fd} program from Example 1. Clearly, it may be the case that Alice follows both Bob and Carol who have conflicting opinions on the whereabouts of a particular person, leading to an FD violation in the course of inference. In this case, the semantics dictates a probabilistic choice of a single fact out of each contradicting set of facts (i.e. a single $IsIn(x, y, z)$ fact for each pair (x, z)), where the probability of a fact to be chosen corresponding for a particular "believer" z is the proportion between the number of rule instantiations yielding it, and the overall possible number of rule instantiations. This is exactly the relative support for the fact, among those followed by z.*

There are clearly many possible alternative semantics one could have chosen for this setting. A main motivation for our choice was to ensure that the semantics coincides with voting in a case such as the previous example.

Remark 6 *A natural extension to our framework would be to allow probabilities to be attached also to the rules, with the semantics that a rule is available for use with some probability. This extension can be easily simulated in our framework. Suppose that we would like a rule r to be considered with probability p. To do that, we create a new fact $rule(r)$ with probability p and add this fact to the body of the rule r. Then this rule is active with probability p.*

4.2 Representation system

We recall the definition of pc-tables [32] that extend concrete c-tables by assigning probabilities to Boolean variables.

Definition 8 *A* pc-table *is a triple (I, β, γ) where (I, β) is a concrete c-table, and γ assigns a probability (a value in the range [0,1]) to each Boolean variable in (I, β).*

Each pc-table (I, β, γ) defines a probabilistic database. A possible world K is an instance of $rep(I, \beta)$. Its probability $\mathcal{P}(K)$ is the sum of probabilities of valuations that yield it:

$$\sum_{\{\nu \text{ generates } K\}} \prod_{\{x|\nu(x)\}} \gamma(x) \times \prod_{\{x|\neg\nu(x)\}} (1 - \gamma(x))$$

The resulting probabilistic database is denoted $rep(I, \beta, \gamma)$.

Computing a representation system. We next consider the computation of a pc-table capturing the result of evaluating a datalog^{fd} program. We will refer invariably to a probabilistic database and to a pc-table representing it.

Recall that we showed that c-table could describe compactly the result of evaluating a datalog^{fd} program. Unfortunately, it is not possible to use the c-table construction of the previous section because it fails to yield correct probabilities. Intuitively, although the variables in the resulting table suffice for capturing all the alternatives, they do not encode enough information regarding the derivation to be assigned correct probabilities.

In general, given a datalog^{fd} program and a pc-table, one can clearly find a pc-table representing the result, that is of exponential size in the input pc-table. It is open whether one of polynomial size exists. Already in a restricted case where the program is nonrecursive and there is only one FD per relation, it is nontrivial to obtain a compact pc-table representation. We do this next. In the next result $rep(T_0)$ refers to the input probabilistic database.

Theorem 8 *For each nonrecursive datalog^{fd} program (P, F) where F includes at most one FD per relation, and for each pc-table T_0, there exists a pc-table T such that*

$$rep(T) = P_F^{psat}(rep(T_0))$$

whose size is polynomial in the size of T_0.

The construction proceeds as follows. We translate a nonrecursive query into a "normal form" where (1) the intensional relations are of two kinds, some relations R that are constrained by a single FD and for each R, a relation R_u that is not constrained, (2) The only rules for constrained relations are of the form $R(u)$:- $R_u(u)$ where u is a vector of distinct variables and (3) each unconstrained relation is defined using an algebraic operation (e.g., selection or union) of already defined relations. This normal form allows inductively defining the conditions based on tables corresponding to already defined relations. The variables used in the conditions include enough information at each stage to be assigned correct probabilities.

Synthesizing pc-tables. We next obtain an analog of Proposition 3, showing how pc-tables can be synthesized using nonrecursive datalog^{fd} programs. For this, only a restricted class of

pc-tables will be necessary as input. An important sub-class of pc-tables that we will focus on are the so-called *tuple-independent databases* [19], in which the function β assigns a distinct variable x_A to each fact A in I. A probabilistic database is *tuple-independent* if it is equal to $rep(T)$ for some tuple-independent pc-table T. The following theorem holds:

Theorem 9 *(i) For each pc-table T over schema S, there exists a tuple-independent pc-table T' of linear size in T over a schema including S and a nonrecursive datalog^{fd} program (P, F) such that $rep(T)$ is the projection on S of $P_F^{psat}(rep(T'))$.*

(ii) For each pc-table T over schema S such that the probabilities of the variables in T are all rational, there exists an instance I over a schema including S and a nonrecursive datalog^{fd} program (P, F) such that $rep(T)$ is the projection on S of $P_F^{psat}(I)$.

The proof of (i) builds on Proposition 3, that of (ii) relies on generating probabilities with arbitrary rational numbers using datalog^{fd} programs on complete instances.

We next consider the problem of computing the probability that a fact is derived.

4.3 Probability computation

Denote by TUPLE−PROB the problem of computing, given a $datalog^{fd}$ query (P, F), a pc-table T_0 and an output tuple t, the probability of t occurring in a possible world of $P_F^{psat}(rep(T_0))$. We start by noting that the problem is hard even for a very restricted case. In [20], the authors identify a class of (nonrecursive) queries for which computation of probabilities over a tuple-independent database is tractable (incurs polynomial data complexity), namely that of *safe queries*. We then note that with the use of a single FD per relation, we can simulate BID tables [49], and that there exist safe queries that are hard w.r.t. BID tables. It follows that TUPLE−PROB is $\sharp P$-hard (data complexity) even when restricted to nonrecursive safe queries, tuple-independent databases, and a single FD per relation.

As upper bound for exact probability computation, we can show a PSPACE algorithm:

Proposition 5 TUPLE−PROB *is in PSPACE.*

Probability approximation. The high complexity of exact query evaluation calls for approximation algorithms. It is standard to distinguish between two kinds of approximation algorithms depending on whether they compute *relative* and *absolute* approximations. An approximation algorithm A to a counting problem C takes as input an instance I of C and a parameter ϵ. Suppose $C(I)$ is the correct answer for input I. We say that A is an absolute approximation if $\mid A(I) - C(I) \mid \leq \epsilon$ for each input instance I. We say that A is a relative approximation if $\mid A(I) - C(I) \mid \leq \epsilon \cdot \mid C(I) \mid$ for each input instance I.

The NP-hardness of possibility (i.e. the hardness of deciding whether the probability of a tuple is greater than 0) immediately implies the nonexistence of a PTIME *relative* approximation algorithm unless P=NP.

On the other hand, we introduce a tractable absolute approximation algorithm. Let Q be the fact whose probability of being derived we wish to compute. We propose an algorithm based on the standard sampling approach, which operates as follows: (1) choose randomly a truth assignment φ for the Boolean variables of the pc-table, (2) generate the instance I corresponding to φ, (3) evaluate the datalog^{fd} rules making probabilistic choices according to the *psat* semantics, until a fixpoint is reached, and (4) check for the existence of Q in the resulting possible world, (5) repeat 1-4 counting the number of worlds where the fact Q is present. The output of the algorithm is the count of such worlds, divided by the number of samples that were considered. This is an estimation for the probability of Q to be derived with respect to the given pc-table. We call this algorithm SAMPLE, and can show the following:

Theorem 10 *Given a pc-table T, a datalog^{fd} program (P, F), and an intensional fact Q:*

(i) The probability computed by SAMPLE converges (as the number of samples grows) to the probability of Q in $P_F^{psat}(rep(T))$.

(ii) The number of samples required for obtaining the correct probability up to an additive error of ϵ, with probability at least $1 - \delta$, is $O(\frac{ln(\frac{1}{\delta})}{\epsilon^2})$.

(iii) Each sample can be produced in polynomial time in $|T|$.

Proof. (sketch) It is easy to observe that we obtain independent samples, since we fully restart the computation whenever we reach convergence. Then (i) is by the law of large numbers and (ii) follows from the Chernoff bound (see e.g. [50]). For (iii), observe that an applicable rule can be identified and applied in PTIME and that only polynomially many inferences have to be performed. ❑

Consequently, there exists a polynomial-time absolute approximation algorithm for TUPLE−PROB.

We next address two additional questions that rise in the context of the probabilistic semantics: how can we *explain* a computed result (using the notion of "most likely support"), and how do we *influence* (i.e. increase or decrease) the probability of a given result.

4.4 Most likely supports

Many ways of explaining why a fact has been derived may be considered. We next introduce and study the notion of "support", that is a set of base facts that explain a result with high confidence in the context of datalog^{fd}. We then analyze the complexity of ranking these supports.

Example 5 *Consider again the datalog program of Example 1. Suppose the fact IsIn$(Carol, London, Bob)$ (standing for: Bob believes Carol is in London) is derived with a certain probability p. There may be many derivations of this fact, using different sets of extensional facts. Perhaps one derivation ("proof") uses the extensional fact that Alice is in London according to Tom. Another derivation may be based on an extensional fact that Carol is in London according to Peter. An analyst of the network (or Bob himself) may be interested in tracking back the origin of Bob's beliefs, finding what are the extensional facts that caused Bob's belief. But this analyst may find an improbable possible origin, a set of base facts with tiny probability, of little interest. The analyst may want to understand what caused Bob's belief with high probability.*

A natural answer in a classical setting would be a set of extensional facts that together allow proving the derived fact. (Such sets are sometimes called the "support" of the proof, which motivates our terminology.) One could consider such a set of facts with maximum probability. However, the situation is more complex because of the FDs. Facts may affect derivations in intricate ways as illustrated next.

Example 6 *Suppose that an intensional relation $Liar$ satisfies the FD $1 \rightarrow 2$. Suppose that $Liar(Bob, yes)$ can be derived using a base fact $Cretan(Bob)$; and $Liar(Bob, no)$ using another*

base fact $Trusty(Bob)$. Now suppose that $Liar(Bob, no)$ allows deriving $IsIn(Alice, London)$ and other facts allow deriving $IsIn(Alice, Paris)$. Note that $Cretan(Bob)$ has some effect on the probability of $IsIn(Alice, Paris)$ even though it is not used in any derivation of that fact.

It should be clear from the examples that the notion of support is intricate in our setting. To define it, we will use the following two auxiliary notions.

Definition 9 *Let (P, F) a datalog^{fd} program, and Q an intentional fact. For a complete database I, the probability of deriving Q from I is denoted $prob(Q \mid I)$. For a probabilistic database $(\mathcal{I}, \mathcal{P})$, the* weight *of an extensional possible world I for Q, denoted $weight_{\mathcal{P},Q}(I)$, is defined as the probability of the extensional possible world I when Q is derived, i.e.,*

$$weight_{\mathcal{P},Q}(I) = \mathcal{P}(I) \cdot prob(Q \mid I).$$

(In both cases, (P, F) is understood.)

Intuitively, the weight of I denotes how likely it is to have the possible extensional world I *as well as* Q derived.

Observe that it may be the case that some fact in a possible extensional instance of top weight is totally irrelevant for deriving Q; it does not participate in a derivation nor in blocking a derivation of Q. Such irrelevant facts should be excluded from an explanation of Q (For instance, an analyst may prefer to see only the facts that really matter). This motivates us to quantify the likelihood of observing *subsets* of instances.

Definition 10 *Let $(\mathcal{I}, \mathcal{P})$ be a probabilistic database, (P, F) a datalog^{fd} program, and Q an intentional fact. The* core-weight *of a set K of base facts for Q is defined by:*

$$\text{core-weight}_{\mathcal{P},Q}(K) = \Sigma_{\{I \in \mathcal{I} \mid K \subseteq I\}} weight_{\mathcal{P},Q}(I).$$

Intuitively, the core-weight of a set of facts is the probability of observing it together with Q.

One may be interested in the set of base facts of highest core-weights. However, not all sets that have high core-weights may be of interest. It may still be the case that such a set includes a very probable fact that does not contribute to deriving Q . We can exclude such sets as follows. We say that a set of base facts K is a *support* of Q if (i) $prob(Q \mid K) > 0$, and (ii) there is no subset $K' \subset K$ with $prob(Q \mid K') \geq prob(Q \mid K)$. As a sanity check, observe that in absence of FDs, $prob(Q \mid K)$ is either 1 (when there is a proof of Q using only facts in K) or 0 otherwise. So in this case, the supports are exactly the sets of base facts that are the leaves of some minimal proof of Q.

We are then interested in the *supports having highest core-weight*. We call `TOP-K-SUPPORTS` the problem of identifying the top k ones. First, we may show a PSPACE algorithm for the general case.

Proposition 6 `TOP-K-SUPPORTS` *can be solved for pc-tables in PSPACE complexity with respect to the size of the pc-table.*

We first observe that in absence of recursion and FDs, and when the pc-table is tuple-independent, the problem is tractable.

Proposition 7 *For tuple-independent pc-tables and non-recursive datalog programs with no FDs,* `TOP-K-SUPPORTS` *can be solved in PTIME (data complexity).*

We observe that for this case the top-k-supports correspond exactly to the most probable disjunct in the DNF lineage of the fact. We note also that the tuple-independence assumption is essential here: for general tables, hardness can be shown.

We then explore separately the introduction of recursion and of FDs. Interestingly, each of them separately already leads to hardness, even for tuple-independent pc-tables and even under further restrictive assumptions.

Theorem 11 *Given a tuple-independent pc-table T, a threshold p, $0 \leq p < 1$, a datalog^{fd} program (P,F), and an intentional fact Q, deciding whether there exists a support for Q with weight greater than p is NP-complete in $|T|$. This holds even if:*

1. *$F = \emptyset$ (no FDs) and P a linear datalog program, or*
2. *P is nonrecursive and F includes at most one FD per relation.*

Note that hardness can be shown in each case by reduction from Set Cover, whose optimization problem is known to be hard to approximate, unless $P = NP$. This means that even a polynomial-time approximation algorithm for top-k supports is unlikely to exist in these cases.

While recursion leads to intractability even under such restricted assumptions, we note that the natural and important program for computing Transitive Closure does allow a natural PTIME solution for the two problems.

Example 7 *Consider the program P*

$$TC(x, y) \text{ :- } E(x, z), TC(z, y)$$
$$TC(x, y) \text{ :- } E(x, y)$$

Consider the graph defined by the relation E and assign the edge from a to b the weight $-log(p)$, where p is the probability assigned to the tuple $E(a, b)$. Let $TC(s, t)$ be an output tuple and observe that the top-k supports are exactly the top-k simple shortest paths *from s to t. The latter problem is known to be in PTIME (see e.g. [25]).*

Identifying the precise fragment of the language that allows for PTIME solutions is an intriguing open question.

4.5 Influencing the derivation

An important related problem is that of understanding *how to modify the extensional data to best influence the probability of a particular answer to a query,* e.g., increase or decrease it. This has strong connections to causality [43], as discussed further. Our quantification of influence follows that of [45, 35] while accounting for the first time for recursion and FDs (both absent in these works).

No FDs, no probability. To set the stage, we start by examining the simplest setting, in which we have a (possibly recursive) datalog program P (without FDs) and an instance I (without probabilities). Given an intensional fact Q such that $P(I) \models Q$, we can define:

• the *necessary* facts for deriving Q: if such a fact is removed, then Q no longer holds (these are the counterfactual tuples of [43]);

• the *relevant* facts for deriving Q: these facts are necessary for deriving Q for some subset of I (tuples belonging to the contingency in [43]).

The nonrecursive case has been considered in [43]. We are interested in investigating the complexity of these problems in the

context of *recursive queries*. For identifying necessary facts, the presence of recursion does not change the complexity since we can still use a straightforward PTIME algorithm of removing the fact and checking whether Q is still entailed. Deciding the relevance of a fact is more difficult. While it is in PTIME for the nonrecursive case, it turns out to be NP-complete in the presence of recursion.

Proposition 8 *Let P be a datalog program. Given an instance I over $ext(P)$, an extensional fact A and an intensional fact Q, the problem of deciding whether A is relevant to the derivation of Q relative to I and P, is NP-complete in data complexity.*

Membership in NP is straightforward, and NP-hardness is by reduction from SAT.

The general case. Classifying facts becomes more challenging when FDs and probabilities are considered, as the next example illustrates:

Example 8 *Consider the following program:*

Q :- R(1), R(2)	*Q'(1) :- S(1)*	*Q'(0) :- S'(x)*
S(1) :- R(1)	*S'(0) :- R(1)*	*S'(1) :- R(1)*

with R for only extensional relation.

Consider Q. Suppose $R(1)$ has probability 0.2 and $R(2)$, 0.8. Thus Q has probability 0.16. If we add 0.1 to $R(1)$ we increase the probability of Q by 0.08, i.e., more than if we increase $R(2)$ by 0.1 that increases it only by 0.02.

Now suppose there is an FD $\emptyset \rightarrow 1$ on Q'. The impact of $R(1)$ on $Q'(1)$ is mixed: it can both be used positively to derive $Q'(1)$ via $S(1)$ and negatively to block the derivation of $Q'(1)$ with $Q'(0)$ via $S'(0)$ and $S'(1)$.

From this example, it is clear that the classical notions of necessary and relevant do not suffice to explain the effect of base facts on derived tuples in a setting with FDs and/or probabilities. To this end, we adapt a definition of [35] to datalog^{fd}. The intuition is to rank extensional facts based on how changes in their probabilities affect the probability of some derived fact of interest.

To simplify, we focus in the remainder of the section on tuple-independent databases and, as in the previous discussion, consider the influence of facts. This can be generalized to arbitrary pc-tables by considering instead the influence of the Boolean variables occurring in a table.

Definition of influence. The intuition is formalized as follows. Given a tuple-independent database (I, γ), $t \in I$, and some $\epsilon < \gamma(t)$, let γ' be defined by: $\gamma'(t) = \gamma(t) - \epsilon$ and $\gamma'(t') = \gamma(t')$ for $t \neq t'$. The *ϵ-influence of t on Q* (relative to I and a datalog^{fd} program P, F) is then defined as ϵ-infl$(t, Q) = \mathcal{P}_{I,\gamma}(Q) - \mathcal{P}_{I,\gamma'}(Q)$ where these are the probabilities of Q with respect to the original, and refined probability function as well as I, P, F (P and F are omitted from notation for brevity). Probability of derivation is with respect to the *psat* semantics.

We aim at ranking tuples based on their influence. We can show an analogous result to that shown in [35] for UCQs: the ranking of facts based on their ϵ-influence is insensitive to ϵ. We denote $\mathcal{P}_{I-\{t\},\gamma}(Q)$ the probability of Q with respect to an instance where t is omitted and all other facts stay intact with no change in their probability; we denote $\mathcal{P}_{I,\{t\},\gamma}(Q)$ the probability of Q with respect to an instance where the probability of t is 1 (and all other facts stay intact). We have:

Proposition 9 *For each $\epsilon > 0$,t, t', I, Q,*
ϵ-infl$(t, Q) < \epsilon$-infl(t', Q) if and only if
$\mathcal{P}_{I,\{t\},\gamma}(Q) - \mathcal{P}_{I-\{t\},\gamma}(Q) < \mathcal{P}_{I,\{t'\},\gamma}(Q) - \mathcal{P}_{I-\{t'\},\gamma}(Q)$.

From the previous result, we may simply define the influence of t on Q as $infl_Q(t) = \mathcal{P}_{I,\{t\},\gamma}(Q) - \mathcal{P}_{I-\{t\},\gamma}(Q)$. We next address the problem of finding the top-k most influential facts, which we call `TOP-K-INFL`.

Influence ranking. Even without recursion and FDs, deciding whether A has the highest influence on Q relative to P, F, I, γ is NP-hard. So we turn to approximate ranking based on sampling.

Definition 11 *We say that a ranking of elements in a set is an absolute ϵ-approximation of the ranking provided by a measure f if, for each elements t, t' in the set, t is ranked higher than t' implies that $f(t') \leq f(t) + \epsilon$.*

Intuitively, if an algorithm ranks t above t', it may be the case that t' has in fact greater influence than t, however the margin is at most ϵ. We can then show:

Proposition 10 *Let $\epsilon, \delta > 0$, (I, γ) be a tuple-independent database and (P, F) be a (*possibly recursive*) datalog^{fd} program computing some fact Q. There exists a probabilistic algorithm that outputs an absolute ϵ-approximation for the `TOP-K-INFL` ranking with probability at least $1 - \delta$. The time complexity of the algorithm is polynomial in $ln(\frac{1}{\delta})$, in $\frac{1}{\epsilon}$ and in $\mid (I, \gamma) \mid$.*

Remark 7 *(Probabilistic fact-at-a-time) We have focused in this section on a probabilistic extension of nsat. A probabilistic extension of nfat can be defined in an analogous way. Recall that we have shown in the previous section how to simulate nfat with nsat. This can be extended to a simulation of pfat with psat. Using this simulation, most results carry to pfat immediately. Other results, namely Theorem 8 and Theorem 11(2) concerned nonrecursive programs. Unfortunately, the simulation of pfat by psat requires recursion. The analogs for pfat of these results thus remain open.*

5. RELATED WORK

There is a large body of work on various aspects of deduction in the presence of inconsistencies, specifying possible worlds, and nondeterministic query languages. We next discuss some of this research and connections with our work.

Dealing with data that violates integrity constraints, including FDs, is studied in the work on *database repairs*. A comprehensive survey of the work on repairs is provided in [13]. Repairs of a given inconsistent database are consistent databases obtained from the original in some specified way (e.g. by removing a minimal set of tuples). The main question is that of *consistent query answering*, i.e. finding answers to queries that are true in all repairs. Datalog-like rules (see e.g. [17]) as well as FDs or key constraints (see e.g. [52]) are commonly used to capture *constraints* in this setting. A fundamental difference between with our work is that we use rules for *deduction*, and are concerned with inconsistencies that arise from the reasoning process rather than in the input database. This in particular means that, unlike the case of repairs, a possible world in our case will not necessarily be a model of the rules: it may be the case that the rules allow derivation of further facts, which are not derived due to the FDs.

While not motivated by inconsistencies, the work on *datalog with choice* provides a mechanism for defining possible worlds

from a given input that is closely related to datalog^{fd}. This is done using nondeterministic choice atoms in datalog rules (see [29]). Intuitively, choice atoms generate maximal instances that satisfy specified FDs. As discussed in Section 2, our *nfat* and *nsat* semantics are closely related to variants of datalog with choice. In particular, it is shown in [29] that datalog with dynamic choice expresses precisely the NDB-PTIME queries, a result that also applies to datalog^{fd} with *nsat* semantics. None of these works incorporate a probabilistic semantics.

Another form of nondeterminism related to choice is provided by the *witness* operator, used in conjunction with FO or fixpoint queries (see [3]). The connection to the FO variant is shown by Theorem 3. The inflationary fixpoint variant is known to express NDB-PTIME [7] so is equivalent in expressive power to datalog^{fd} with *nsat* semantics and to datalog with dynamic choice. A variant of the non-deterministic witness operator is the *repair-key* operator of [10, 44]. The authors propose a representation system (based on a notion of *world-set decomposition*) and study the complexity of query evaluation in the non-deterministic case.

Inference in the presence of FDs is a form of non-monotonic reasoning, since the presence of some tuples forbids the presence of other tuples. Not surprisingly, datalog^{fd} (like datalog with choice) is related to datalog with negation (datalog$^{\neg}$). Indeed, as discussed in Section 2, the set of possible worlds defined by datalog^{fd} with *nfat* semantics can equivalently be defined as the set of stable models of a corresponding datalog$^{\neg}$ program. On the other hand, datalog^{fd} with *nsat* semantics is similar in spirit to datalog$^{\neg}$ with inflationary fixpoint semantics (see [3]).

Data integration and exchange also raise the issue of handling contradictions, studied in different lines of work [24, 38, 33, 26]. This research emphasizes algorithms for data sharing or corroboration of opinions, rather than inference semantics in presence of FDs. Integrity constraints in data exchange and for description logics (see e.g. [22, 11, 46, 15, 16, 51]) are also studied mainly in the context of semantics and efficient query answering rather than representation systems and probabilities. The work of [39] studies the construction of representation systems for possible answers but does not show a PTIME construction for recursive queries with FDs, and also does not handle probabilities.

In the context of probabilistic and incomplete databases, typical works, e.g. [50, 34, 4, 32, 20], consider contradictions in the input database, for which (in the probabilistic case) a distribution on their possible solutions is given in advance. Recursion in a probabilistic setting is considered in [23], where probabilities are introduced only once, on ground facts. In the presence of FDs that may be violated in the course of query evaluation in some possible worlds (possible input instances), a common solution is to ignore these worlds and to adapt the distribution to account only for the worlds in which no such violation occurs. Our solution is different, as we propose semantics to *settle* the contradictions. This approach was taken in [36] in conjunction with *nonrecursive* queries (the probabilistic repair-key operator is used to enrich the positive relational algebra). The semantics is different, since it (1) only allows to probabilistically repair one FD violation at a time (which is different in the probabilistic setting) and (2) probabilities are not based on the number or order of derivations, but rather on an attribute with numeric values. Similar differences thus hold with respect to [21], enriching datalog with probabilistic repair of keys. We also mention in this context the work of [12], that models probabilistic XML through the use of Recursive Markov Chains, but does not account for contradictions. Last, we note that the computational problems of identifying top-k support and influence computation were not studied in the context of recursive queries.

Finally, we have mentioned connections between top-k supports and influence and the works of [45, 43, 35]. Also relevant is the work on querying parse trees of PCFGs [18]. One major technical difference is that for PCFGs, probabilities of derivation rule activation are usually assumed to be independent; in contrast, in our context the rules may depend on common probabilistic tuples and thus may be correlated.

6. CONCLUSION

We briefly describe some directions for future work.

This work was originally motivated by the management of knowledge in distributed settings [2]. Distributed variants of datalog have been investigated recently [2, 9, 40]. In a distributed setting, different peers may have opinions that may lead to the derivation of conflicting facts. Asynchronicity among peers is an additional source of nondeterminism that may also be quantified probabilistically; the need to address asynchronicity is relevant also for distributed systems processing big data, such as [42, 41]. It would be interesting to study how techniques and results from this paper extend to such a setting. In particular, one could consider adapting the sampling algorithm to approximate probabilities of answers in a distributed environment. Distribution also raises novel issues such as minimizing the communication cost to perform such sampling.

Finally, we plan to implement and optimize our algorithms and study experimentally their applicability to the analysis of interactions in networks. There is a wealth of potential applications, such as analyzing how rumors spread in a network, or improving estimates on correctness of facts and trust of peers.

Acknowledgments. We thank Meghyn Bienvenu for working with us on previous related research, and for her invaluable comments on this paper. This research has been supported in part by the Advanced European Research Council grant Webdam on Foundations of Web Data Management, by the Israeli Ministry of Science and by the Israeli Science Foundation.

7. REFERENCES

[1] S. Abiteboul, M. Bienvenu, and D. Deutch. Deduction in the presence of distribution and contradictions. In *WebDB Workshop*, pages 31–36, 2012.

[2] S. Abiteboul, M. Bienvenu, A. Galland, and E. Antoine. A rule-based language for web data management. In *PODS*, 2011.

[3] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[4] S. Abiteboul, P. C. Kanellakis, and G. Grahne. On the representation and querying of sets of possible worlds. *Theor. Comput. Sci.*, 78(1), 1991.

[5] S. Abiteboul, B. Kimelfeld, Y. Sagiv, and P. Senellart. On the expressiveness of probabilistic xml models. *VLDB J.*, 18(5), 2009.

[6] S. Abiteboul and V. Vianu. Fixpoint extensions of first-order logic and datalog-like languages. In *LICS*, pages 71–79, 1989.

[7] S. Abiteboul and V. Vianu. Datalog extensions for database queries and updates. *J. Comput. Syst. Sci.*, 43(1):62–124, 1991.

[8] S. Abiteboul and V. Vianu. Non-determinism in logic-based languages. *Ann. Math. Artif. Intell.*, 3(2-4):151–186, 1991.

[9] P. Alvaro, W. R. Marczak, N. Conway, J. M. Hellerstein, D. Maier, and R. Sears. Dedalus: datalog in time and space. In *Datalog*, pages 262–281, 2011.

[10] L. Antova, C. Koch, and D. Olteanu. $10^{(10^6)}$ worlds and beyond: efficient representation and processing of incomplete information. *VLDB J.*, 18(5):1021–1040, 2009.

[11] M. Arenas and L. Libkin. Xml data exchange: Consistency and query answering. *J. ACM*, 55(2), 2008.

[12] Michael Benedikt, Evgeny Kharlamov, Dan Olteanu, and Pierre Senellart. Probabilistic xml via markov chains. *PVLDB*, 3(1), 2010.

[13] L. E. Bertossi. *Database Repairing and Consistent Query Answering*. Morgan & Claypool Publishers, 2011.

[14] P. Buneman, S. Khanna, and W.C. Tan. Why and where: A characterization of data provenance. In *ICDT*, 2001.

[15] A. Calì, G. Gottlob, T. Lukasiewicz, and A. Pieris. Datalog+/-: A family of languages for ontology querying. In *Datalog*, pages 351–368, 2010.

[16] A. Calì, G. Gottlob, and A. Pieris. Towards more expressive ontology languages: The query answering problem. *Artif. Intell.*, 193:87–128, 2012.

[17] Andrea Cali, Domenico Lembo, and Riccardo Rosati. Query rewriting and answering under constraints in data integration systems. In *IJCAI*, 2003.

[18] S. Cohen and B. Kimelfeld. Querying parse trees of stochastic context-free grammars. In *ICDT*, pages 62–75, 2010.

[19] N. N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. In *VLDB*, pages 864–875, 2004.

[20] N. N. Dalvi and D. Suciu. Management of probabilistic data: foundations and challenges. In *PODS*, 2007.

[21] D. Deutch, C. Koch, and T. Milo. On probabilistic fixpoint and markov chain query languages. In *PODS*, 2010.

[22] R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data exchange: semantics and query answering. *Theor. Comput. Sci.*, 336(1), 2005.

[23] N. Fuhr. Probabilistic datalog:a logic for powerful retrieval methods. In *SIGIR*, 1995.

[24] A. Galland, S. Abiteboul, A. Marian, and P. Senellart. Corroborating information from disagreeing views. In *WSDM*, 2010.

[25] J. Gao, H. Qiu, X. Jiang, T. Wang, and D. Yang. Fast top-k simple shortest paths discovery in graphs. In *CIKM*, 2010.

[26] Wolfgang Gatterbauer and Dan Suciu. Data conflict resolution using trust mappings. In *SIGMOD*, 2010.

[27] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *Proc. of ICLP*, pages 1070–1080, 1988.

[28] F. Giannotti, S. Greco, D. Sacca, and C. Zaniolo. Programming with non-determinism in deductive databases. *Annals of Mathematics and Artificial Intelligence*, 19, 1997.

[29] F. Giannotti and D. Pedreschi. Datalog with non-deterministic choice computers ndb-ptime. *J. Log. Program.*, 35(1):79–101, 1998.

[30] S. Greco, D. Saccà, and C. Zaniolo. Datalog queries with stratified negation and choice: from p to d^p. In *ICDT*, pages 82–96, 1995.

[31] T. J. Green, G. Karvounarakis, and V. Tannen. Provenance semirings. In *Proc. of PODS*, 2007.

[32] T. J. Green and V. Tannen. Models for incomplete and probabilistic information. *IEEE D. Eng. Bull.*, 29(1), 2006.

[33] T.J. Green, G. Karvounarakis, Z. G. Ives, and V. Tannen. Update exchange with mappings and provenance. In *VLDB*, 2007.

[34] T. Imielinski and W. Lipski Jr. Incomplete information in relational databases. *J. ACM*, 31(4), 1984.

[35] B. Kanagal, J. Li, and A. Deshpande. Sensitivity analysis and explanations for robust query evaluation in probabilistic databases. In *SIGMOD Conference*, pages 841–852, 2011.

[36] C. Koch. Approximating predicates and expressive queries on probabilistic databases. In *PODS*, pages 99–108, 2008.

[37] C. Koch. A compositional query algebra for second-order logic and uncertain databases. In *ICDT*, 2009.

[38] L. Kot and C. Koch. Cooperative update exchange in the youtopia system. *PVLDB*, 2(1), 2009.

[39] L. Libkin. Data exchange and incomplete information. In *PODS*, pages 60–69. ACM, 2006.

[40] B. T. Loo, T. Condie, M. Garofalakis, D. Gay, J. M. Hellerstein, P. Maniatis, R. Ramakrishnan, T. Roscoe, and I. Stoica. Declarative networking: language, execution and optimization. In *SIGMOD*, 2006.

[41] Yucheng Low, Danny Bickson, Joseph Gonzalez, Carlos Guestrin, Aapo Kyrola, and Joseph M. Hellerstein. Distributed graphlab: A framework for machine learning and data mining in the cloud. *Proc. VLDB Endow.*, 5(8):716–727, April 2012.

[42] Grzegorz Malewicz, Matthew H. Austern, Aart J.C Bik, James C. Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. Pregel: A system for large-scale graph processing. In *SIGMOD*, 2010.

[43] A. Meliou, W. Gatterbauer, K. F. Moore, and D. Suciu. The complexity of causality and responsibility for query answers and non-answers. *PVLDB*, 4(1):34–45, 2010.

[44] D. Olteanu, C. Koch, and L. Antova. World-set decompositions: Expressiveness and efficient algorithms. *Theor. Comput. Sci.*, 403(2-3):265–284, 2008.

[45] C. Ré and D. Suciu. Approximate lineage for probabilistic databases. *Proc. VLDB Endow.*, 1(1):797–808, August 2008.

[46] R. Rosati. On the complexity of dealing with inconsistency in description logic ontologies. In *IJCAI*, pages 1057–1062, 2011.

[47] D. Saccà and C. Zaniolo. Stable models and non-determinism in logic programs with negation. In *PODS*, pages 205–217, 1990.

[48] P. Senellart and S. Abiteboul. On the complexity of managing probabilistic xml data. In *PODS*, pages 283–292, 2007.

[49] D. Suciu, D. Olteanu, C. Ré, and C. Koch. *Probabilistic Databases*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2011.

[50] D. Suciu, D. Olteanu, C. Ré, and C. Koch. *Probabilistic Databases*. Morgan & Claypool Publishers, 2011.

[51] B. ten Cate, G. Fontaine, and P. G. Kolaitis. On the data complexity of consistent query answering. In *ICDT*, pages 22–33, 2012.

[52] Jef Wijsen. Charting the tractability frontier of certain conjunctive query answering. In *PODS*, pages 189–200, 2013.