Deduction in the Presence of Distribution, Contradictions and Uncertainty*

Serge Abiteboul
INRIA Saclay and ENS Cachan
fname.lname@inria.fr

Meghyn Bienvenu
CNRS and Université Paris-Sud
meghyn@lri.fr

Daniel Deutch
Ben Gurion University of the Negev; INRIA Saclay and ENS Cachan
deutchd@cs.bgu.ac.il

ABSTRACT

We study deduction, captured by *datalog*-style rules, in the presence of contradictions, captured by *functional dependencies* (FDs). We start with a simple semantics for datalog in the presence of functional dependencies that is based on inferring facts one at a time, never violating the FDs, until no further facts can be added. This is a non-deterministic semantics, that may lead to several possible worlds. We present a proof theory for this semantics and compare it to previous work on datalog with negation. We also discuss a set-at-a-time semantics, where at each iteration, all facts that can be inferred are added to the database, and then choices are made between contradicting facts. We then proceed to our main goal of defining a semantics for the *distributed setting*. Note that contradictions naturally arise in a distributed setting since different peers may have conflicting information, opinions or recommendations. In the distributed case, we propose and study a concrete semantics for (an important fragment of) a previously proposed distributed datalog idiom, namely *Webdamlog*, that we enrich to account for FDs. Here again, we compare the semantics with previously studied semantics and in particular Webdamlog with negation in the absence of FDs. Finally, we note that in a distributed environment, it is natural to settle contradictions by introducing *probabilities*. We consider a simple adaptation of the distributed semantics to a probabilistic setting and show that it captures an intuitive way of resolving contradictions. We propose a sampling algorithm for evaluating queries under this semantics.

1. INTRODUCTION

Our goal is to study how peers in a network exchange information and reason together to achieve some goals. A difficulty is that the peers may state or infer conflicting facts (conflicts are captured here by violations of functional dependencies). Peers can settle conflicts by choosing between contradicting base or inferred facts. Their choices add to the uncertainty already inherent in an asynchronous environment. This uncertainty can be naturally modeled by introducing *probabilities*. We study the semantics of datalog-like languages and query evaluation in this setting.

We use as the basis for our work some semantics for *datalog* in the presence of *functional dependencies* (FDs for short). We extend this semantics first to a distributed datalog language, namely Webdamlog, then to its natural probabilistic extension. We show connections with a number of prior works notably on datalog with negation, c-tables, probabilistic databases, and voting to resolve contradictions.

Datalog with FDs. As a starting point, we consider the centralized case. Here we study datalog in presence of FDs, for which we consider a nondeterministic semantics. The intuition behind this semantics is that a derived fact is rejected if its introduction into the database results in an FD violation (i.e., if it is in conflict with another existing fact). So, in a forward chaining manner, we keep adding inferable facts, one at a time, until any further addition would result in a dependency violation. Nondeterminism results from the order in which inferable facts are added. We consequently refer to this semantics as non-deterministic fact-at-a-time (*nfat*) semantics.

We give an interpretation of the *nfat* semantics in terms of datalog with negation.More precisely, we propose a translation of datalog programs with FDs to datalog programs with negation (but no FDs) and show that the possible worlds of a datalog program with FDs with respect to the *nfat* semantics correspond to the stable models of the translation of the program into datalog with negation. We also provide a corresponding proof theory for *nfat*. We show that proving that a fact necessarily or possibly holds is hard.

While the fact-at-a-time semantics is simple and intuitive, for practical reasons, it may be preferable to derive sets of facts at a time. We consider a nondeterministic set-at-a-time semantics (*nsat* for short). We compare it to *nfat* and show some tight connections with inflationary datalog$^\neg$.

Note that semantics for datalog with functional dependencies have been considered before, from a somewhat different perspective (see discussion in Sections 2 and 6). We revisit the topic and present new results, such as the proof theory,

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

mainly as a foundation for our study of the management of contradictions in a distributed setting.

Distribution and Webdamlog. Contradictions are very common in a distributed setting. As a simple example, consider peers exchanging information about the location of other peers. This can be captured using a binary relation $IsIn$, where $IsIn(p, c)$ means that a peer p is currently in city c. Each peer knows a partial copy of this relation that reflects her personal knowledge. We will also include the FD $IsIn : 1 \rightarrow 2$ since a peer cannot be in two distinct cities at the same time. Observe however that even if this FD is respected locally (i.e., in the knowledge of a peer), different peers may associate different locations to the same peer. Now, assume that facts are propagated in the network. Clearly, a peer Bob may receive contradicting facts about the location of Alice, if Bob has two friends who have different opinions about her whereabouts. We thus need a semantics that handles such contradictions.

For capturing the exchange of information and knowledge inference in a distributed setting, we use a recently proposed distributed datalog idiom called Webdamlog [**?**]. We "marry" the *nsat* semantics for datalog with FDs and the semantics of a positive fragment of Webdamlog. We study the resulting semantics and in particular we show how it can be supported by Webdamlog with negation.

Introducing probabilities. Uncertainty is inherent in a distributed setting, and a probabilistic modeling of it is very natural. For instance, assume that 15 of Bob's friends tell him that Alice is in Paris and 5 say that she is in London. In the absence of additional information (such as a statement by Alice herself or her boyfriend), Bob may choose to believe she is in Paris with a probability of 75% and in London with 25%. (Or he may prefer to just go with the majority and believe that she is in Paris. But again his decision is based on computing probabilities.) It turns out that the probabilistic modeling is an easy adaptation of the *nsat* semantics: we assume a uniform distribution on the ordering of peer activation and on the possible ways for settling contradictions. Each such choice dictates a possible world. Then the probability of a fact is exactly the sum of the probabilities of the possible worlds which contain this fact.

This provides *a simple and natural probabilistic semantics for query answering in a distributed setting in the presence of contradictions.* We show with an example how this semantics naturally allows capturing peer policies based on voting. Then we investigate query evaluation for the probabilistic semantics. First we consider exact computation of probabilities based on conditional table representations of the possible worlds (equipped with probabilities) and of query answers. After highlighting the inherent difficulties of such a representation, we propose a more realistic alternative for the distributed setting, which is a sampling-based approximation technique.

Paper Organization. The rest of this paper is organized as follows. In Section 2, we study the *nfat* semantics for datalog in presence of FDs. In Section 3, we study the *nsat* semantics. In Section 4, we adapt the approach to a distributed setting and in Section 5, we further introduce probabilities. We provide an overview of related work in Section 6, and conclude in Section 7.

For space considerations, proofs of some results have been moved to the appendix.

2. DATALOG WITH FDs

In this section, we study the management of contradictions arising from extending datalog with functional dependencies. This provides foundations for our investigation of the analogous problem in a more general setting with distribution and probabilities. We begin by presenting a semantics based on fact inferences (forward chaining). We show how it may be interpreted in terms of datalog with negation. We then present a corresponding proof theory (backward chaining).

Datalog under functional dependencies has previously been studied in the context of nondeterministic extensions of datalog with choice operators [**?**, **?**]. The model semantics and the relationship with stable models we use are in the spirit of this line of work. The proof theory we present is new to our knowledge. In particular, for certain answers, it is nontrivial. The results in this section are primarily meant to pave the way for the study of distribution and probabilities in the next sections.

We assume the reader is familiar with the standard definitions of datalog and of functional dependencies (FDs for short), see [**?**]. We also consider in what follows datalog programs with negation. We recall that the body of a *datalog*$^\neg$ rule may contain negated atoms, subject to the restriction that any variable in the rule occurs in some positive atom in the body. The stable and well-founded semantics for datalog$^\neg$ programs are recalled in the appendix, and the inflationary semantics is recalled in Section 3.

Model semantics

We introduce a semantics for datalog in presence of FDs. Intuitively, this semantics is based on *forward chaining* with instantiated rules applied one at a time. (We will consider in the next section, set-at-a-time inference.) Each rule application generates a new candidate fact (tuple) that is added to the database unless its addition violates an FD. Note that this introduces non-determinism since the result of the process may rely on the order of rule activation. We consequently refer to this semantics as *nfat*, standing for *nondeterministic fact-at-a-time.*

To formally define the semantics, we first re-define the immediate consequence and the consequence operators in the presence of FDs. Note that we always assume that the database instance is consistent with the FDs.

DEFINITION 1. *Let S be a database schema. Let F and P be, respectively, a set of FDs and a datalog program over S. Let I be an instance over S satisfying F. The* immediate consequence operator $\rightarrow_{nfat}$ *is defined as follows: $I \rightarrow_{nfat} I \bigcup \{A\}$ if and only there exists an instantiation A :- $A_1, ..., A_n$ of a rule in P such that $\{A_1, ..., A_n\} \subseteq I$ and $I \bigcup \{A\}$ satisfies the FDs. The* consequence operator $\rightarrow^*_{nfat}$ *is the reflexive transitive closure of the immediate consequence operator.*

We now formally define possible worlds, which correspond to the different ways of settling contradictions.

DEFINITION 2. *Given a schema S, an instance I, a set F of FDs and a datalog program P over S, an instance J over*

S is a possible world *for* (I, P, F) *if* J *is a maximal instance satisfying* $I \rightarrow^*_{nfat} J$. *The set of possible worlds is denoted* $pw^{nfat}(I, P, F)$.

EXAMPLE 1. *Consider a database whose schema consists of a ternary relation IsIn and a binary relation Friend. Intuitively* $IsIn(\$X, \$Y, \$P)$ *means "The person* $\$P$ *thinks that the person* $\$X$ *is in the city* $\$Y$*", and so we impose the FD* $IsIn : 1, 3 \rightarrow 2$. *Now consider the following datalog program:*

$$IsIn(\$X, \$Y, \$P) \text{ :- } Friend(\$P, \$P_1), IsIn(\$X, \$Y, \$P_1)$$
$$IsIn(carol, \$Y, \$P) \text{ :- } IsIn(alice, \$Y, \$P)$$

The first rule states that each person believes his friends about the whereabouts of people, whereas the second states that it is general knowledge that Carol is in the same city as Alice. Now assume that the initial database is as follows:

$$IsIn(alice, paris, peter), IsIn(carol, london, tom),$$
$$Friend(ben, tom), Friend(ben, peter)$$

There are two possible worlds for this program. They respectively contain:

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

since the presence of a fact in each case blocks the derivation of another one.

Certainty and possibility of facts can be defined in the standard manner: a fact is said to be *certain* if it appears in all possible worlds, and it is called *possible* if it belongs to at least one possible world. The sets of all certain (resp. possible) facts according to the *nfat* semantics is denoted $cert^{nfat}(I, P, F)$ (resp. $poss^{nfat}(I, P, F)$).

Connection with negation

FDs and negation share similar characteristics, since the existence of some tuples in the database combined with the FDs forbids other tuples from appearing in the database. Indeed, we next examine the connection between our semantics for datalog with FDs and previously proposed semantics of datalog programs with negation. To this end, we introduce a natural translation of datalog programs with FDs to datalog programs with negation.

Given a datalog program P and a set F of FDs, we create a datalog$^\neg$ program $NegProg(P, F)$ as follows. Each rule $R(\$U_1, ..., \$U_n)$:- $Body$ in P is replaced by the rule

$$R(\$U_1, ..., \$U_n) \text{ :- } Body, \neg C_R(\$U_1, ..., \$U_n)$$

where C_R is a new relation of the same arity as R. Then, $NegProg(P, F)$ also contains the following rule:

$$C_R(\$U_1, ..., \$U_n) \text{ :- } R(\$V_1, ..., \$V_n), \$U_{j_1} = \$V_{j_1}, \ldots, \$U_{j_\ell} = \$V_{j_\ell}, \$U_k \neq \$V_k$$

for each R, each FD $R : X \rightarrow Y$ in F with $X = \{j_1, \ldots, j_\ell\}$ and each $k \in Y$.

The program $NegProg(P, F)$ can be interpreted according to any of the many different semantics proposed for datalog programs with negation. However, the following theorem shows that it is the stable model semantics that leads to our semantics for datalog with FDs:

THEOREM 1. *For each schema* S, *each input instance* I, *program* P, *and set* F *of FDs over* S, $J \in pw^{nfat}(I, P, F)$ *if and only if there is a stable model of* $NegProg(P, F)$ *with respect to input* I *that coincides with* J *on* S.

It should be observed that some general datalog programs with negation do not have any stable models. The previous characterization is possible only because FDs capture a limited form of negation.

Note also that if we were to use well-founded semantics in place of stable model semantics, fewer facts would be certain, as the following example demonstrates:

EXAMPLE 2. *Consider the following datalog program* P:

$$R(a, \$X) \text{ :- } T(\$X) \qquad B \text{ :- } R(a, \$X)$$

and suppose F *contains a single FD* $R : 1 \rightarrow 2$. *The translated program* $NegProg(P, F)$ *is:*

$$R(a, \$X) \text{ :- } T(\$X), \neg C_R(a, \$X) \qquad B \text{ :- } R(a, \$X)$$
$$C_R(\$X, \$Y) \text{ :- } R(\$W, \$Z), \$X = \$W, \$Y \neq \$Z$$

On the instance $I = \{T(a), T(b)\}$, *there are two stable models:* $\{T(a), T(b), R(a, a), C_R(a, b), B\}$ *and* $\{T(a), T(b), R(a, b), C_R(a, a), B\}$. *As the fact* B *appears in all stable models,* B *is certain w.r.t. stable model semantics (and hence also w.r.t. our possible worlds semantics). By contrast, well-founded semantics assigns unknown to* B.

Proof theory

We next equip the previous semantics with a proof theory. We believe that the proof theory further shows that the semantics naturally captures a correct intuition on managing contradictions. Furthermore, the proof theory can be used to effectively check whether some logical statements are certain or possible.

First, we define a notion of a proof tree which takes into account the FDs:

DEFINITION 3. *Given an instance* I, *a datalog program* P, *and a set* F *of FDs, a* proof tree *for a fact* A *w.r.t. (I,P,F) is a tree labelled with facts, such that the root is labelled* A *and each node satisfies one of the following conditions:*

1. *It is a leaf node and is labelled with a fact from* I.
2. *It is labelled* B *and its children have labels* $C_1, ..., C_n$, $n \geq 0$, *where* B :- $C_1, ..., C_n$ *is an instantiation of a rule in* P.

Additionally, we require that the set of facts in the node labels satisfies the FDs F.

The following theorem provides a simple characterization of possible answers in terms of proof trees:

THEOREM 2. *For each input instance* I, *a program* P *and a set* F *of FDs,* $A \in poss^{nfat}(I, P, F)$ *if and only if there exists a proof tree for* A *with respect to (I,P,F).*

EXAMPLE 3. *Let* $I = \{A, B\}$, $P = \{C$:- $R(a, 0), R(a, 1), R(a, 0)$:- $A, R(a, 1)$:- $B\}$, *and* $F = \{R : 1 \rightarrow 2\}$. *There are proof trees for both* $R(a, 0)$ *(using* A*) and* $R(a, 1)$ *(using* B*). There is no proof tree for* C *since such a tree would require the presence of two nodes labelled respectively* $R(a, 0)$ *and* $R(a, 1)$, *which the definition forbids.*

Now consider certainty. Proving a fact to be certain involves showing that it is present in all possible worlds, or in other words, that there is no possible world where this fact is absent. This leads us to introduce the notion of a *refuting proof tree* for A. In such a tree, nodes may be labelled both by facts or by negated facts (where we use $\neg A$ to mean "A is absent"). There are two ways to prove a negated fact $\neg A$: either refute all possible derivations of A, or give a proof of a fact B that contradicts A. This intuition can be formalized as follows:

DEFINITION 4. *Given a database instance I, a datalog program P, and a set F of FDs, a* refuting proof tree *for a fact A w.r.t. (I,P,F) is a tree where each node is labelled with a (possibly negated) fact, such that the root is $\neg A$ and each node satisfies one of the following conditions:*

1. *It is a leaf node which is labelled with a fact from I.*
2. *It is labelled B and its children have labels $C_1, ..., C_n$, where B :- $C_1, ..., C_n$ is an instantiation of a rule in P.*
3. *It is labelled $\neg B$ and has children labelled $\neg C_1, \ldots, \neg C_n$, $n \geq 0$, where $B \notin I$ and each instantiation of a rule in P that concludes on B has some C_i in its body.*
4. *It is labelled $\neg B$ and has a unique child labelled C such that $\{B, C\}$ violates some FD in F.*
5. *It is a leaf node labelled $\neg B$, and it has an ancestor which is also labelled $\neg B$.*

Additionally, we require that the set of (positive) facts in the node labels satisfies the FDs F, and that the tree does not contain as labels both a fact and its negation.

EXAMPLE 4. *Let $I = \{A\}$, $P = \{R(a,0)$:- $R(a,1), R(a,1)$:- $R(a,0), R(a,2)$:- $A\}$, and $F = \{R : 1 \rightarrow 2\}$. To show $R(a,1)$ is not certain, we can use the refuting proof tree whose root node $\neg R(a,1)$ has a unique child $\neg R(a,0)$ which in turn has a unique child $\neg R(a,1)$. Alternatively, we could use a refuting proof tree whose root has as a subtree a proof tree for $R(a,2)$. To show that $R(a,2)$ is certain, we observe that the existence of a refuting proof tree for $R(a,2)$ would either imply the existence of a refuting proof tree for A (impossible since $A \in I$) or the existence of a proof tree for $R(a,c)$ for some $c \neq 2$ (also impossible).*

We can show the following theorem.

THEOREM 3. *For each input instance I, program P, and set F of FDs, a fact A is certain w.r.t. (I,P,F) if and only if there exists no refuting proof tree for A w.r.t. (I,P,F).*

While we have established a proof theory, we note that it not surprisingly leads to computationally expensive decision procedures, both for possibility and uncertainty (NP and coNP respectively). Furthermore, one can show that the problems are hard for these classes by reduction from (un)satisfiability. Therefore, we have:

THEOREM 4. *Given an input instance I, a datalog program P, a set F of FDs and a fact A, the problem of deciding if $A \in poss^{nfat}(I, P, F)$ is NP-complete in the size of I, whereas the problem of deciding if $A \in cert^{nfat}(I, P, F)$ is coNP-complete in the size of I.*

3. SET-AT-A-TIME SEMANTICS

In the previous section, we have considered a semantics for datalog with FDs that can be interpreted (in a forward chaining manner) as introducing one fact at a time. One can consider alternatively a semantics based on inferring in one step all the facts that are immediate consequences of the facts and the rules. Note however that some of these facts may contradict known facts and moreover, pairs of these facts may contradict each other. Thus, at each step, we add in a nondeterministic manner a maximal subset of the set of facts that may be inferred without causing an FD violation. We will refer to this semantics as *nondeterministic set-at-a-time* (*nsat*) semantics. The formal definition is as follows:

DEFINITION 5. *Consider a database instance I, a set F of FDs, and a program P. Let Σ be the set of immediate consequences of I and P (with the standard datalog semantics). The* immediate consequence operator *for P w.r.t. F, denoted by $\rightarrow_{nsat}$, is defined by: $I \rightarrow_{nsat} I \bigcup \Sigma'$ if and only if Σ' is a maximal subset of Σ such that $I \bigcup \Sigma'$ satisfies F. Let $\rightarrow^*_{nsat}$ be the reflexive transitive closure of this operator. An nsat* possible world *for I is a maximal instance J such that $I \rightarrow^*_{nsat} J$.*

EXAMPLE 5. *We apply the nsat semantics to Example 1. At each iteration, all possible facts are derived, and then contradictions are settled by adding a maximal subset of the newly derived facts that is consistent with the FDs. In the first iteration, the facts*

$$IsIn(alice, paris, ben), IsIn(carol, london, ben)$$

are derived. In the next iteration, the derivation of IsIn(carol, paris,ben) is blocked since it contradicts the fact IsIn(carol, london,ben). In this particular example, there is only one possible world under nsat semantics.

We use $cert^{nsat}(I, P, F)$ (resp. $poss^{nsat}(I, P, F)$) to denote the certain (resp. possible) facts with respect to the *nsat* semantics. The analogs of the complexity results for *nfat* semantics hold for the *nsat* semantics.

THEOREM 5. *Given an input instance I, a datalog program P, a set F of FDs and a fact A, the problem of deciding if $A \in poss^{nsat}(I, P, F)$ is NP-complete in the size of I, whereas the problem of deciding if $A \in cert^{nsat}(I, P, F)$ is coNP-complete in the size of I.*

Relationship with nfat

Observe that the *nfat* semantics includes two kinds of nondeterminism: (i) data nondeterminism (choosing between different values for $\$Y$ for the same value of $\$X$) and (ii) control nondeterminism (choosing a particular order of rule application). The set-at-a-time semantics removes the second kind of nondeterminism, leaving only the first. As a consequence, not all *nfat* possible worlds are reachable. However, the instances that are reached are always *nfat* possible.

THEOREM 6. *For each input instance I, datalog program P, and set F of FDs, we have that: (i) each nsat possible world is an nfat possible world; and (ii) the converse does not hold in general.*

PROOF. For (i), we simply note that any *nsat* computation can be decomposed into a sequence of additions of a

single fact, yielding an *nfat* computation. For (ii), let $I = \{R(a,b)\}$, $P = \{R(a,b') \text{ :- } R(a,b);\ S(\$X,\$Y) \text{ :- } R(\$X,\$Y)\}$, and $F = \{S : 1 \to 2\}$. Then $\{R(a,b),\ R(a,b'),\ S(a,b')\}$ is possible for the *nfat* but not the *nsat* semantics. □

Note that it follows from the previous result that:

$$poss^{nsat}(I,P,F) \subseteq poss^{nfat}(I,P,F),$$
$$cert^{nfat}(I,P,F) \subseteq cert^{nsat}(I,P,F),$$

and the inclusions may be strict.

Evaluation in inflationary datalog¬

Observe that the proposed semantics is inflationary in the sense that if $I \to_{nsat} J$, then $I \subseteq J$. Indeed, we next highlight connections with inflationary datalog¬ [**?**]. More precisely, we consider the construction of *nsat* possible worlds using inflationary datalog¬.

We recall that the inflationary semantics of a datalog¬ program P on an instance I is the limit, denoted $P_{inflat}(I)$, of the inflationary sequence:

$$\Gamma_P(I) \subseteq \Gamma_P^2(I) \subseteq \Gamma_P^3(I)...$$

where Γ_P is defined as follows:

$$\Gamma_P(K) = K \cup \{A \mid A \text{ immediate consequence of } K \text{ and } P\}$$

Note that inflationary datalog¬ is *deterministic*, whereas we want to use it to simulate the *nsat* semantics that is *non-deterministic*. Thus, we need to introduce some source of data non-determinism. This is achieved in the spirit of languages extending datalog with choice constructs, for instance the *witness* operator defined in [**?**].

Formally, we use the auxiliary notion of determination.

DEFINITION 6. *Let I be an instance with relations $R_1, ..., R_n$, such that each R_i has arity k_i. Let C be the set of constants in I. A* determination I' of I *is obtained by extending I with, for each i, the relation $Prefer_{R_i}$ of arity $2k_i$ where $I'(Prefer_{R_i})$ contains a total order of C^k.*

Intuitively, the total ordering of the tuples in R_i specifies how to choose between conflicting facts. Now we have:

THEOREM 7. *Let S be a schema. Consider a datalog program P, a set F of FDs, and an instance I over S. Then one can construct a datalog¬ program P' such that J is a possible world for (I,P,F) under the nsat semantics iff J coincides over S with $P'_{inflat}(I')$ for some determination I' of I.*

PROOF. (sketch) Let I be an instance. First consider a possible world J for I. Observe that the run that lead to J specifies some partial order of the tuples for each relation R_i based on the choices in case of conflicts. Consider a total order extending this partial order. This defines a determination I' of I. Consider the FD-fix algorithm of Figure 1. It is easy to verify that FD-fix computes J on that determination of I. Now observe that FD-fix is in PTIME. Note also that one can easily construct a total order of the domain of I in inflationary datalog¬. Thus, since inflationary datalog¬ captures all of PTIME on ordered data [**?**], there exists an inflationary datalog¬ program P' that computes FD-fix. The program P' applied to I' yields J.

Conversely, consider a determination. We can construct the corresponding possible world essentially by using the $Prefer_{R_i}$ relations to resolve the conflicts. □

input: a set F of FDs, a program P,
an instance I over relations R_i,
a determination of I
(also denoted I by abuse of notation)
using relations $Prefer_{R_i}$ (for each i)
output: an *nsat* possible world w.r.t. (I,P,F)

repeat until fixpoint
for each i
compute in $\bar{R}_i$ the facts that are
immediate consequence of I and P
for each i, $\widehat{R_i} := \emptyset$;
for each i, repeat until fixpoint
let t be the largest fact in $\bar{R}_i$
according to $Prefer_{R_i}$
that is compatible with $R_i \cup \widehat{R_i}$;
$\widehat{R_i} := \widehat{R_i} \cup \{t\}$;
for each i, $R_i := R_i \cup \widehat{R_i}$

Figure 1: the FD-fix algorithm

We conclude this section with a remark.

REMARK 1. *Observe that the FD-fix algorithm inserts tuples one at a time. This is to simplify the presentation. One can in fact construct an inflationary datalog¬ program that computes the possible world using a number of steps that is linear in the number of $\to_{nsat}$ steps used to obtain the possible world. Intuitively, one computes in a first step the tuples that could be derived but are blocked by a tuple that can be derived and is preferred. Then in a second step, one computes the tuples that are actually derived.*

4. THE DISTRIBUTED CASE

In this section, we move to a distributed setting, taking as our starting point the distributed datalog dialect Webdamlog recently introduced in [**?**]. We will consider a relevant fragment of this language, referred to as s-Webdamlog (for "simple" Webdamlog). We will study how s-Webdamlog can be equipped with a semantics in the presence of FDs that is the counterpart of the semantics considered for the centralized case. But first, we start with a simple example to illustrate the language.

EXAMPLE 6. *Let us reformulate Example 1 using Webdamlog syntax. Instead of a single relation $IsIn(\$X,\$Y,\$P)$, we have a separate IsIn relation for each peer p, which is denoted by $IsIn@p(\$X,\$Y)$. Each peer p has the FD $IsIn@p : 1 \to 2$ and the following rules:*

$$IsIn@\$P(\$X,\$Y) \text{ :- } Friend@p(\$P), IsIn@p(\$X,\$Y)$$
$$IsIn@p(carol,\$Y) \text{ :- } IsIn@p(alice,\$Y)$$
$$IsIn@p(\$X,\$Y) \text{ :- } baseIsIn@p(\$X,\$Y)$$

Observe the use of variable $\$P$ that matches all the peers that are friends of peer p. Intuitively the first rule says that if you know where someone is, you let your friends know. The initial database includes the following facts:

$$baseIsIn@peter(alice,paris) \quad friend@peter(ben)$$
$$baseIsIn@tom(carol,london) \quad friend@tom(ben)$$

We will see in what follows how to extend the *nsat* semantics to our distributed setting. This will specify possible

worlds for Ben and their intensional facts. In some possible worlds, Ben will think Carol is in Paris, and in others he will think she is in London.

Before considering FDs, we introduce the fragment of Webdamlog that is used in the sequel.

s-Webdamlog

We consider only a subset of the Webdamlog language, referred to as s-Webdamlog (for "simple" Webdamlog). The semantics we propose may be extended to the full language in a straightforward manner. However, the results that we show do not hold for the general language. The extension is briefly discussed at the end of the section.

Alphabet. We assume the existence of two infinite disjoint alphabets of sorted *constants*: *peer* and *relation*. We also consider the alphabet of *data* that includes in addition to *peer* and *relation*, infinitely many other constants of different sorts (*integer*, *string*, *bitstream*, etc.). It is because *data* includes *peer* and *relation* that we may write facts such as the first rule for the *IsIn* relation in Example 6. Similarly we have corresponding alphabets of sorted *variables*. An identifier starting by the symbol \$ implicitly denotes a variable. A *term* is a variable or a constant.

A *schema* is an expression (Π, $\mathcal{E}$, $\mathcal{I}$, σ) where Π is a set of peer IDs; $\mathcal{E}$ and $\mathcal{I}$ are disjoint sets, respectively, of *extensional* and *intensional* names of the form $R@p$ for some relation name R and some peer p; and the typing function σ defines for each $R@p$ in $\mathcal{E} \cup \mathcal{I}$ the arity and sorts of its components.

Facts and rules. A (p-)*fact* is an expression $R@p(\overline{u})$ where $R@p$ is a relation and $\overline{u}$ is a vector of data elements of the proper type, i.e., correct arity and correct sort for each component. We consider that a peer of ID q has a finite set of *s-Webdamlog rules* of the following form:

$$M_{n+1}@Q(\overline{U}) \text{ :- } M_1@q(\overline{U}_1)...M_n@q(\overline{U}_n)$$

where each M_i is a relation term, Q is a peer term, and each $\overline{U}_i$ is a vector of data terms, and subject to the condition that every variable in the head must occur in the body. Observe in particular that if Q is a variable then it must be bound to a peer ID in the body so that we know who the derived fact concerns.

Note that unlike in the general language of [**?**], we do not allow negated atoms in the body, and we assume that all the facts in the body of a rule concern the particular peer that holds this rule. In the terminology of [**?**], the rules we consider are *positive* and *local*. Also, in what follows, we are interested in a setting where each peer has a fixed set of extensional facts, and only the intensional predicates vary. Concretely, this means that we assume the peers' rules are *deductive* [1], i.e. only intensional predicates appear in the heads of rules.

Semantics. The philosophy underlying Webdamlog is that we define a local semantics to be used at each peer, which then induces a global semantics based on moves and runs.

When restricted to positive, local, deductive rules, and the standard datalog semantics is chosen for the local semantics, a *move* of a peer p consists in: (i) computing the fixpoint of p's program (its rules and its extensional facts), and (ii) alerting other peers to the derived facts that concern them via *delegations*. Concretely, when p derives a fact $R@q(\overline{u})$ for some $q \neq p$, it delegates the rule $R@q(\overline{u})$:- to q. This rule is now available to q for use during its computation.

We define a Webdamlog *state* as a 3-tuple (I, Γ, D), where I assigns each peer p a set $I(p)$ of extensional p-facts, Γ assigns each peer p a set Γ of rules, and D assigns to each peer p a set $D(p)$ of rules that have been delegated to p by others peers. In the special case that $\cup_p D(p) = \emptyset$, we call the state a *system* and refer to it by (I, Γ).

A *run* of a system (I, Γ) is a sequence of moves starting from (I, Γ) which satisfies *fairness*, i.e. each peer p is invoked infinitely many times.

Note. While in our restricted setting, delegation essentially reduces to exchange of facts, in general Webdamlog, delegation lets peers install arbitrary rules at other peers. This allows for the definition of a clean semantics for non-local rules, whose bodies involve atoms of multiple peers.

Semantics of s-Webdamlog with FDs

We now extend s-Webdamlog to account for FDs. Intuitively, the logic of a peer contains in addition to its facts and rules, a set of FDs that constrain its local relations. We have to select a particular FD-sensitive semantics to specify the local computation. We choose *nsat* rather than *nfat*, as it is computationally more appealing.

We require that the initial state $I(p)$ of each peer p satisfies its local constraints. Note that when it is a peer's turn to move, the FDs may force the peer to make choices about which intensional facts to derive. According to the semantics we propose, in a possible run, once a peer chooses between two conflicting facts, the peer will not revise that choice. This is illustrated by the following example.

EXAMPLE 7. *Suppose* $I(p) = \{T(0,0), T(0,1)\}$, $\Gamma(p) =$

$$\{R@p(\$X, \$Y) \text{ :- } T@p(\$X, \$Y)$$
$$R@q(\$X, \$Y) \text{ :- } R@p(\$X, \$Y)\}$$

and $F(p) = \{R@p : 1 \rightarrow 2\}$. *Then when it is p's turn to move, his local nsat computation may generate either* $\{R@p(0,0), R@q(0,0)\}$ *or* $\{R@p(0,1), R@q(0,1)\}$. *In the first case,* $R@q(0,0)$ *:- is delegated to q, whereas in the latter case, it is* $R@q(0,1)$ *:- which is delegated. Note however that no changes are made to* $I(p)$, $\Gamma(p)$, *or its dependencies* $F(p)$ *to record the choice made. So, if p is called again immediately, he has precisely the same two possibilities, but nothing ensures that the same alternative is chosen.*

The semantics we use guarantees peer always makes the same choice. This is achieved by endowing each peer p with an inflationary set of facts $M(p)$ (where M stands for "memory") that accompanies the peer p throughout the run. Formally, a state of a peer p now consists of five components: $(I(p), \Gamma(p), F(p), D(p), M(p))$, i.e., its extensional facts $I(p)$, its original set of rules $\Gamma(p)$, its FDs $F(p)$, the set $D(p)$ of rules that have been delegated to p, and its memory $M(p)$.

Let us now define formally how the peer moves using the *nsat* local semantics. In a given state ($I(p)$, $\Gamma(p)$, $F(p)$, $D(p)$, $M(p)$), we use the *nsat* semantics to nondeterministically choose a fixpoint for the set of rules $\Gamma(p) \cup D(p)$ applied

[1]Technically, Webdamlog requires persistence rules of the form $R@p(\overline{U})$:- $R@p(\overline{U})$ to make extensional facts persist across states. For simplicity, we ignore this detail.

to $I(p) \cup M(p)$ under the constraints $F(p)$. This defines some set H of new facts in the local intensional relations that are added to $M(p)$ and some new delegations that are sent to other peers. Observe that only D and M evolve during the course of a run and that they grow monotonically. We are thus guaranteed that each run converges:

THEOREM 8. *Let (I, Γ, F) be a s-Webdamlog system over a finite set of peers. Then for each fair run $\sigma_0\sigma_1\sigma_2\ldots$ of (I, Γ, F), there exists i such that for all $k \geq i$, $\sigma_i = \sigma_k$.*

Because of this theorem, we often identify a run with its finite prefix up to the point of convergence, and we speak of the *final state* of a run.

Possibility and certainty. Our semantics for s-Webdamlog with FDs is non-deterministic, and we may again (as in the centralized case) consider possibility and certainty of presence of facts in the final state of runs. The definitions for possibility and certainty are as in Section 2, adapted to the distributed setting. It is easy to show that the lower bounds for these problems from Section 2 carry to the distributed case. Designing a distributed algorithm for possibility and certainty is more challenging. Following Section 2, showing possibility of a fact amounts to finding a proof for it, and showing certainty amounts to showing the inexistence of a refuting proof tree. For the former, in some cases it can be done locally at a peer (if the local facts constitute sufficient evidence to prove the fact). However, it may be the case that there is no such local proof, and still the fact is possible via a proof that uses non-local facts. In the worst case, without a priori knowledge about the other peers and their programs, discovering the proof requires broadcast communication. Similarly, uncertainty of a fact F is easy to show in case local facts can be used to refute it (i.e. by proving a fact contradicting F), but in general deciding certainty may still require broadcast communication.

Comparison with the centralized case

We next compare the *nsat* semantics of s-Webdamlog that we have just defined with some "natural" corresponding centralized semantics. In the absence of FDs, for a positive case like the one considered here, it was shown in [**?**] that the distributed semantics is essentially identical to the centralized one. An analogous result does not hold in the presence of FDs for the *nsat* semantics, as demonstrated next.

EXAMPLE 8. *Consider the system (I, Γ, F) with peers p and q, no extensional facts (i.e. $I(p) = I(q) = \emptyset$), a single FD on the unary relation $G@p$, and the following programs:*

$$\Gamma(p) = \{A@p \text{ :- }, B@p \text{ :- } A@p, C@p \text{ :- } B@p,$$
$$G@p(0) \text{ :- } C@p, D@q \text{ :- }, G@p(1) \text{ :- } E@p\}$$
$$\Gamma(q) = \{E@p \text{ :- } D@q\}$$

Note that the first time peer p is called, the delegation $E@p$:- is not present (since it is only produced once q receives $D@q$:- from p), which means that the derivation of $G@p(0)$ is not blocked. There is thus a unique possible world for (I, Γ, F) which contains $G@p(0)$.

The corresponding centralized system intuitively includes all relations, facts and FDs of the peers, keeping the peer identifiers on the relation names to distinguish relations of the same name originally residing at different peers (see Def. 8 in the appendix for the formal definition). In the centralized system $G@p(1)$ can be generated in three steps, whereas $G@p(0)$ requires four steps, and so the unique possible world contains $G@p(1)$. Consequently the sets of possible worlds for the distributed and centralized systems are distinct.

Note that if we were to use *nfat* semantics locally, then the obtained distributed semantics would generate some but not all *nfat* possible worlds of the centralized program. This is because when activating a peer, it runs until fixpoint before releasing control. An exact correspondence between the distributed and the centralized case is obtained if we use a local semantics in which only one step of *nfat* computation is done at each move (i.e. use $\rightarrow_{nfat}$ rather than $\rightarrow^*_{nfat}$). However, this semantics is not satisfying from a practical viewpoint as it requires communication after every inference step.

Simulation with Webdamlog

The Webdamlog language has been studied extensively in [**?**]. The presence of FDs raises novel issues, as previously observed. Interestingly, the semantics we have defined for s-Webdamlog with FDs can be "simulated" in "almost" standard Webdamlog with negation. This is analogous to the centralized case, where we have simulated the evaluation of datalog with FDs programs using inflationary datalog$^\neg$ (in Section 3). In fact, the translation that was introduced for the centralized setting is a key element in the translation that we use here: essentially the same translation is performed at each peer.

In the distributed setting, we need to record the memory relations in M, which means the translation also has to take care of these relations. For each local intensional relation $R@p$, we use an extensional relation $R_m@p$ satisfying the same FDs as $r@p$. (So note that the simulating program is not purely intensional). The *nsat* simulation works as follows: (i) for each $R@p$, a rule of the form $R@p(\vec{u})$:- $R_m@p(\vec{u})$ "loads" R_m in R to block the derivation of conflicting tuples, (ii) we simulate the local rules, and (iii) for each $R@p$, a rule of the form $R_m@p(\vec{u})$:- $R@p(\vec{u})$ records the memory. Note we need to block the execution of (ii) until (i) is performed. This is achieved by adding a proposition *wait* in the program rules and using a rule *wait* :- that unblocks the rules.

Since we are still using an inflationary semantics at each peer, and the set of extensional facts grows monotonically, every run still converges. Note however that to extract the set of facts from the final state, we must take all facts in $\cup_p I(p)$, and then translate every "memory" fact $R_m@p(\bar{u})$ to the fact $R@p(\bar{u})$ it represents. We use the notation $Facts(\sigma)$ to denote the result of applying this procedure to a state σ.

The following theorem demonstrates that determination allows simulating the inferences of a s-Webdamlog system with FDs by a Webdamlog system without FDs:

THEOREM 9. *Consider a s-Webdamlog system with FDs (I, Γ, F). Then one can construct a set of rules Γ' such that W is a possible world for (I, Γ, F) if and only if there exists a determination I' of I, such that $W = Facts(\sigma)$ for a convergence state σ of the system (I', Γ'), with inflationary datalog$^\neg$ as local semantics.*

The proof is similar to the proof of Theorem 7.

Extension to full Webdamlog. Recall that we have defined the semantics for a restricted fragment of Webdamlog called s-Webdamlog, enriched with FDs. To conclude this section, we consider the extension of that semantics to the whole Webdamlog language. Specifically, we have assumed that the rules are all local, positive, and deductive. Extending the semantics and results to account for non-local rules is straightforward; indeed, the restriction to local rules was made only to simplify the presentation.

Allowing negation in deductive rules complicates matters significantly. With negation, one may use the absence of a fact to infer some delegation and then receive evidence of that fact. This leads to revision and convergence is no longer guaranteed. This difficulty was already encountered in [**?**]. The semantics of runs carries to this setting, and the definition of possibility and certainty may be adapted by considering only converging runs. However extending our study to rules with negation requires further investigation.

Finally, the consideration of active rules (i.e., updates) is even more problematic. First, with updates, convergence is again not guaranteed. Furthermore, to analyze different possible runs, one needs to roll back updates, which is complex in this setting. In particular, when there are interactions with the outside world, certain actions cannot be rolled back (e.g., booking a flight). The extension to active rules is an interesting direction for future research.

5. INTRODUCING PROBABILITIES

The semantics we have considered so far are nondeterministic. In this section, we interpret this nondeterminism *probabilistically*, focusing on (a probabilistic variant of) the *nsat* semantics [2]. It is intuitive that for instance a peer p may choose probabilistically between conflicting facts, with the probability of a fact to be chosen being proportional to its support among peers that p considers credible. We will show how to capture such a policy with a very simple program under our semantics.

The psat semantics

There are three different places where uncertainty plays an important role in our approach. This uncertainty is "measured" using probabilities as follows:

Peer ordering For the uncertainty in peer activation ordering we use a uniform distribution for the choice, at each round, of the peer that is activated.

Resolving contradictions To model the uncertainty arising from the different ways of resolving FD violations, we consider *one FD at a time*, and for each FD, we uniformly choose between contradicting facts, at each iteration of the *nsat* semantics. This raises issues when there is more than one maximal FD per relation, as we discuss further.

Uncertainty on base facts To model uncertainty of the base facts, we assume each base fact is associated with some formula over a set of Boolean variables (local to every peer), and the peer randomly picks these variables according to some known distribution. To simplify the presentation, this part of the model is discussed only towards the end of the section. For now, we assume that all base tuples are certain.

[2]We could similarly introduce a probabilistic version of the *nfat* semantics, but this will not be considered here.

When there is more than one FD per relation, a single fact may participate in conflicts for several FDs. The approach we follow is to resolve the conflicts one FD after another according to some (probabilistically generated) order. This yields the algorithm OFD-fix in Figure 2. *All nsat possible worlds are generated by this algorithm.* However, surprisingly, some worlds that are not *nsat* possible may also be generated, as the following example demonstrates.

EXAMPLE 9. *Suppose we have a zero-ary relation U, a unary relation T, a ternary relation R, and the following FDs: $f0 = T : \emptyset \to 1$, $f1 = R : 1 \to 2$, and $f2 = R : 2 \to 3$. Consider the empty instance and the following program:*

$R(a,b,c)$:-	$R(a,b',c)$:-
$R(a',b',c')$:-	$T(1)$:- U
$T(0)$:- $R(a,b',c)$	$T(0)$:- $R(a,b,c), R(a',b',c')$
U :- $R(a,b,c)$	U :- $R(a,b',c)$
U :- $R(a',b',c')$	

Note that there are two nsat possible worlds: $\{R(a,b',c), T(0), U\}$ and $\{R(a,b,c), R(a',b',c'), T(0), U\}$. Now we see what happens when we use OFD-fix to resolve the conflicts at each computation step. In the first step, we derive three immediate consequences: $R(a,b,c), R(a,b',c), R(a',b',c')$. We call OFD-fix with these three facts in Δ, an empty instance I, and some order of the FDs, say $f_2 f_1 f_0$. We first treat f_2, say by selecting $R(a,b',c)$ and discarding $R(a',b',c')$. Then when treating f_1 we may select $R(a,b,c)$ and discard $R(a,b',c)$. The last FD is inapplicable, so OFD-fix terminates with $I = \{R(a,b,c)\}$. In the subsequent computation step, the set of immediate consequences contains the three R-facts and a new fact U. Applying OFD-fix with any FD ordering yields the new instance $I = \{R(a,b,c), R(a',b',c'), U\}$. In the third computation step, there are two new immediate consequences: $T(0)$ and $T(1)$. We may select $T(1)$ and obtain the instance $\{R(a,b,c), R(a',b',c'), U, T(1)\}$. Note that we have reached a fixpoint (all immediate consequences not in the set are blocked by the FDs), and yet this set does not correspond to any nsat possible world.

To avoid generating worlds that are not *nsat* possible, an alternative, in the spirit considered in Section 3, would be to use a (probabilistically generated) order on all the facts over each relations. Note however that the number of probabilistic choices to be made for one step of the computation (corresponding to the number of different orderings of facts) would depend on the data and could possibly be very large. On the other hand, using OFD-fix, the number of choices for one step depends only on the schema.

Furthermore, we stress that in the common case where there is a single maximal FD per relation, the FDs for the different relations can be treated independently, and then the possible worlds generated by OFD-fix are precisely the *nsat* possible worlds, no matter which order is used.

Moves and runs. We next show how to adapt the notions of moves and runs to a probabilistic setting.

DEFINITION 7. *A* probabilistic move *for peer p in state (I, Γ, F, D, M) is defined as follows.*

input: an ordered sequence F of FDs,
an instance I over relations R_i satisfying F
a set Δ of candidate facts to be added
output: a subset of $I \cup \Delta$ satisfying F

$\Delta := \Delta \setminus \{\alpha \in \Delta \mid \alpha$ conflicts with I given $F\}$
for each FD $R_i : X \to A$ in the order of F do
  for each u in $\pi_X(\Delta(R_i))$ do
    choose randomly (and uniformly) $v \in \Delta(R_i)$ such that $\pi_X(v) = u$
    for each $v' \in \Delta(R_i)$ with $\pi_X(v') = u$, $v' \neq v$ do
      delete $R_i(v')$ from Δ
return $I \cup \Delta$

Figure 2: OFD-fix Algorithm

1. *Choose randomly (and uniformly) an order Ω of the FDs in $F(p)$.*
2. *(Incoming information) Apply the OFD-fix algorithm using the order Ω, the set of believed facts $I(p) \cup M(p)$, and the set $\{A \mid A$:- $\in D(p), A \notin M(p)\}$ of candidate facts proposed by delegations. Denote by Θ_1 the resulting set of facts and let $i = 1$.*
3. *(Derived information) Repeat until a fixpoint Θ_n for some n is reached: compute Θ_{i+1} that is the output of the OFD-fix algorithm with input the order Ω, Θ_i, and the set of facts obtained by applying the immediate consequence operator of Γ to Θ_i.*
4. *For each intensional fact $R@q(\bar{u})$ in Θ_n, if $q = p$, add $R@p(\bar{u})$ to $M(p)$, otherwise for $q \neq p$, add $R@q(\bar{u})$:- to $D(q)$.*

A run *for a system (I, Γ, F) is a finite sequence of probabilistic moves such that the last one leads to a* terminal state*, i.e., a state that is kept unchanged by every possible probabilistic move.*

Note that a move involves a sequence of probabilistic choices, first to choose the ordering Ω and then to decide between contradicting facts in OFD-fix. Observe that these choices are Markovian in the sense that each choice is independent of the previous ones. The *probability* of a move is then the product of the probabilities of the choices made in this sequence. The *probability* of a run is the product of the probabilities of the moves made along the run.

REMARK 2. *We have defined runs as* finite *sequences of probabilistic moves ending in a terminal state. Alternatively, one could consider infinite sequences. Intuitively, one can consider that the probability of a finite converging sequence is the sum of the probabilities of the infinite sequences of which it is the prefix. Now a subtlety is that this does not account for all infinite sequences. Indeed, some infinite sequences never reach a terminal state. Consider for instance a sequence that selects always the same peer p and thus never realizes some task that another peer has to perform for termination to be achieved. It turns out that the probability of such an infinite sequence is null. Indeed, a property of s-Webdamlog*[3] *we consider here is that a terminal state is reached with probability 1.*

[3]It should be observed that this property does not hold in general for Webdamlog.

We refer to the probabilistic semantics that we have just defined as the *psat* semantics.

Probability of a Boolean query. Given a Boolean, local, query [4] Q (for instance, a union of conjunctive queries over the relations of some peer), we can define its probability as the sum of the probabilities of all the runs for which Q holds in the final state.

EXAMPLE 10. *Consider a set of peers p_i, $1 \leq i \leq N$, where each peer has a relation $baseIsIn@p_i(Person, Location)$ with the FD $baseIsIn@p_i : 1 \to 2$ and the meaning that p_i believes that person is in a particular location. Suppose that another peer, namely Bob, has a relation* [5]

$$IsIn@bob(Person, Location, Peer)$$

with the FD $IsIn@bob : 1 \to 2$ where Bob records the locations of people based on other peers' opinions. Suppose each p_i has the fact $Follower@p_i(bob)$ where Follower is an asymmetric relation (in the style of twitter), and the rule:

$$IsIn@\$P(\$X, \$Y, p_i) \quad :- \quad Follower@p_i(\$P), \; baseIsIn@p_i(\$X, \$Y)$$

Finally suppose we have:

$$\begin{array}{ll} baseIsIn@p_i(alice, london) & i \leq M \\ baseIsIn@p_i(alice, paris) & i > M \end{array}$$

Let $\alpha = M/N$. Observe that the probability that a peer p_i believes that Alice is in London is α (and $1 - \alpha$ for Alice in Paris). Let us compute the belief of Bob that Alice is in Paris, i.e., the probability of the fact $IsIn@bob(alice, london, p_i)$ for some i to appear in the state of Bob. Let us call that event L. Note that there are infinitely many finite runs in which L can happen, since it may take an arbitrarily long sequence of peer activations before Bob is first called. To compute the probability of L, let e_k be the event that k distinct peers that p follows were activated before p's first activation. For each k, let e_{jk} be the event that out of these k peers, exactly j derived $IsIn@bob(alice, london, p')$ for some p'. Now we have: $Pr(L) = \sum_{k=0\ldots N} Pr(L \mid e_k) * Pr(e_k)$. Observe that $Pr(L \mid e_k) = \sum_{j=0\ldots k} Pr(L \mid e_{jk}) * Pr(e_{jk})$ $= \sum_{j=0\ldots k} j/k * Pr(e_{jk}) = 1/k * \sum_{j=0\ldots k} j * Pr(e_{jk})$. Now observe that the sum expression is exactly the expected number of peers believing $baseIsIn@p_i(alice, london)$ for some i, when sampling k peers, which is simply $\alpha * k$, yielding that $Pr(L \mid e_k) = \alpha$. Since this is independent of k and we have $\sum_{k=0\ldots N} Pr(e_k) = 1$ we obtain $Pr(L) = \alpha$. So the probability of Bob believing that Alice is in London is exactly the proportion of the peers Bob follows that hold this belief.

The previous example illustrates that the *psat* semantics can capture the natural intuition of using votes to decide between conflicting opinions. This may seem a complicated manner of obtaining a simple average. However, the interest of the semantics is that it also allows one to capture more complicated settings, e.g., recursive definitions of *IsIn* or complex interactions between uncertain facts.

[4]To simplify the presentation we focus on local queries, but the construction can be extended to queries that use facts of multiple peers.

[5]The third attribute is introduced for technical reasons, and will be used to record the peer that has "passed" the fact to Bob, see further.

Representation using c-tables

Clearly, one would like to be able to represent the possible fixpoints in some (preferably compact) representation. As is standard in the presence of multiple possible worlds, we can consider the use of c-tables [**?**] for such a representation. A *c-table* is a relation in which each tuple is associated with a condition. A *condition* is a Boolean combination of events such as (in)equalities between constants and variables. The idea is to capture all *possible* facts (as defined in the previous section) in a c-table. A possible world is obtained by choosing the values for the events. In a particular world, the facts that appear are exactly those for which the corresponding condition is true. We say that such a c-table *captures* the possible facts for a given system (I, Γ, F). When the events are assigned probabilities, this induces a probability distribution over possible worlds.

Usually, c-table representation can be obtained by keeping the full provenance [**?**, **?**, **?**] of the possible computations. Towards the goal of obtaining such c-tables, we must assume all peers are willing to fully disclose all their information, in particular the provenance of facts. Observe that keeping track of provenance is not trivial, because of the possibly unbounded interaction between the peers.

We start by presenting a construction that "almost" works, in the sense that it indeed constructs a sequence of c-tables that capture more and more precisely the set of possible worlds, but unfortunately the exact representation is never reached. By induction, a c-table representing the set of possible worlds after i steps is constructed. For this, at step i, we introduce probabilistic events for the next move. For instance, some events would correspond to the selection of the peer to move, and others to the choice of an FD ordering, Unfortunately, if the convergence of runs is guaranteed in s-Webdamlog (because we impose that a peer that can be called is eventually called), convergence is not guaranteed for this sequence of c-tables because it is assumed to consider *all* possible runs. In fact, there exist systems for which, for all i, the c-tables at step i and $i+1$ are not equivalent.

To fix this construction, we must not introduce events for moves that do not change the state. If at step i, we can move from state σ to state σ', $\sigma' \neq \sigma$, we introduce an event $e_{i,\sigma,\sigma'}$. Now the computation of the c-table at step i to step $i+1$ is much more complicated, but possible (it is a lengthy case analysis). The difference is that the c-table at step i represents the possible states after i "real" moves, i.e., i changes of states. So we are now guaranteed to converge because of the inflationary nature of the computation. (To check convergence, we have to verify if the tables at step i and $i+1$ are equivalent, which can be performed.) This yields a construction of a c-table that one can use to compute probabilities of queries, either exactly via brute-force computation or using approximation techniques as in [**?**].

Discussion. There are serious drawbacks to such an approach based on c-table representation. First, the size of the representation that is obtained may be prohibitively large. Even more importantly, there are privacy issues. To obtain the c-table representation, all peers must disclose fully their base facts and also their rules, indicating how they derive new facts. Clearly, this provenance information is needed to compute a global c-table. For instance, suppose p_1 tells some fact f to many different peers who tell f to p_2. To compute the probability of the fact at p_2, one needs to know the provenance of the facts to see that they are not independent statements. Revealing all provenance information, including internal logic, is often considered unacceptable. We therefore introduce next an alternative approach for computing query probabilities that is purely based on sampling and does not require a peer to disclose as much information.

Sampling

We want to use sampling to answer Boolean queries. Note that a query q may be answered positively in some possible worlds and negatively in others, and the objective is to compute the probability that it is answered positively. For that, we propose a simple distributed sampling technique.

Distrib-Sampling. We assume that there are finitely many peers and that each peer knows about the existence of all other peers and can contact them. Then, some peer p_q performs one round of sampling as follows:

1. p_q asks the other peers to make a copy of their initial state in temporary relations. Then the peers simulate a run using these temporary relations as follows.

2. p_q asks each peer in the system whether it can move. If some peer can move, then p_q uniformly chooses the next peer to be activated out of those than can move. The peer that is activated makes a probabilistic move and then returns control to p_q for the next simulation step (goto 2.). If no peer can move (a fixpoint has been reached), then p_q records that one more sample has been obtained and whether q is answered positively in this sample and moves on (goto 3.).

3. If the desired number of samples has not been reached, p_q initiates a new sample (goto 1.); and if it has, p_q terminates by giving an estimate of the probability that q holds.

One can show:

THEOREM 10. *(1) The probability computed by Distrib-Sampling converges (as the number of samples grows) to the probability of q according to the* psat *semantics.*

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

(3) The expected time for producing each sample is polynomial in the size of the input instance [6].

PROOF. (sketch) It is easy to observe that we obtain independent samples, since we restart the computation whenever we reach convergence. Then (1) is by the law of large numbers and (2) follows from the Chernoff bound (see e.g. [**?**]). For (3) observe that in the worst case, only one fact can be added by a single peer at each step. But, since we are choosing peers uniformly, the expected waiting time before we activate the peer that actually adds a fact is equal to the number of peers in the system, say N. Since the peer IDs are part of the input instance, N is less than the instance size. Thus the expected total number of moves before convergence is polynomial in the size of the input instance. □

[6] Observe that the peer IDs are part of the instance.

Considering probabilistic base facts. To simplify the presentation we have not considered probabilities assigned to the facts in the initial database. Introducing probabilistic base facts to the framework is useful in a distributed environment, as demonstrated by the next example.

EXAMPLE 11. *Reconsider Example 10, and now assume that the peers p_i that know about the whereabouts of Alice are not totally sure of what they know. Intuitively, we would expect the probability of the presence of Alice in a city to be the average of the probability that the peers p_i believe she is there. Indeed, this is what is obtained with the semantics described next. Note that it would be easy to weigh the opinions of other peers based on the confidence that one has in them, where the confidence in other peers is also captured using additional probabilistic base facts.*

As mentioned above, to capture probabilities on base facts we can use for local states one of the standard models used for probabilistic databases, e.g. probabilistic c-tables [**?**]. In the adapted semantics we then distinguish the first move of a peer from his other moves. In his first move a peer starts by choosing probabilistically a truth value for the variables participating in conditions that appear in the c-table. Consequently its initial state is chosen. Note that we have assumed above that the initial instance of a peer $I(p)$ is consistent with the FDs $F(p)$. This may no longer hold, and consequently after choosing the initial state, the peer's first move proceeds to choose an order on the FDs and fix each of them (in a similar manner to OFD-fix). From this point on, moves and runs are defined as before.

6. RELATED WORK

We next give an overview of related work in a number of relevant areas.

Datalog with negation or nondeterminism. There is a rich literature devoted to datalog extended with negation, and a number of different semantics have been proposed, among them, inflationary semantics, stable model semantics, and well-founded semantics (cf. Chapters 14-15 of [**?**] for an introduction). We showed a close connection between the *nfat* semantics and stable model semantics.

Also relevant is the work on datalog extended with nondeterministic witness [**?**] or choice constructs [**?**, **?**]. On a formal level, there are strong similarities between our *nfat* possible worlds and the stable choice models of [**?**], and between our *nsat* semantics and the eager dynamic choice semantics of [**?**]. The motivations behind the two lines of research are however quite different: our objective is to model the inherent "don't know nondeterminism" associated with resolving contradictions, whereas work on nondeterministic datalog aims at pruning the space of query answers, and thus corresponds to "don't care nondeterminism". These different motivations lead to the exploration of different issues: certain answers, which are important in our setting, are not considered in the work on nondeterministic datalog, which instead focused on issues related to implementation and expressivity. Also, to the best of our knowledge none of the works in this area have addressed distribution.

Contradictions and repairs. A repair of a database instance w.r.t. to a set of integrity constraints is an instance which satisfies the constraints and differs "minimally" from the original instance (cf. [**?**] for discussion of different minimality criteria). There is a large body of work devoted to the problem of *consistent query answering* (see [**?**, **?**] for surveys) for relational DBs in the presence of integrity constraints (typically FDs or inclusions dependencies). To our knowledge, no study of consistent query answering has been carried out for datalog programs, although logic programming has been used as a tool for performing consistent query answering for relational DBs [**?**]. Recent work on the repair-checking problem [**?**] allows more expressive rule-like integrity constraints (tgds), but these constraints must be satisfied in every repair. By contrast, our datalog rules are used for *inference*, rather than as constraints, and as a result, possible worlds may not contain all tuples which can be inferred given the rules. This also distinguishes our approach from the work on data exchange (cf. [**?**, **?**, **?**]) and on inconsistency-handling for description logics [**?**].

Two other recent works on handling contradictions merit mention. In [**?**], the authors enriched datalog with a repair-key construct (originally introduced in [**?**]) that allows a probabilistic choice between contradicting facts. However, [**?**, **?**] do not account for distribution, and consequently different choices were made in the semantics design, leading to different challenges in query evaluation. Handling contradictions in a multi-agent environment was recently studied in [**?**], but no inference was considered in this context.

Distributed datalog. To our knowledge, the first attempts to distribute datalog on different peers are [**?**, **?**]. More recently, in [**?**], the authors suggest an operational semantics based on a distribution of the program before the execution; an additional model was introduced in [**?**], based on an explicit time constructor. Distributed datalog-style rules have been shown to be extremely useful for implementation of e.g. Web routers [**?**], DHT [**?**] and Map-Reduce [**?**]. These works served as motivation for the introduction of Webdamlog [**?**]. Both Webdamlog and the distributed datalog dialects used in [**?**, **?**] allow negation in rule bodies, but they do not address the problem of resolving contradictions, which is the focus of the present work.

Probabilistic and incomplete databases. Typical work on probabilistic and incomplete databases (e.g. [**?**, **?**, **?**, **?**, **?**]) consider contradictions in the input database, for which (in the probabilistic case) a distribution on their possible solutions is given in advance. In presence of FDs that may be violated in the course of query evaluation in some possible worlds (possible input instances), the 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 here is different, as we propose semantics to settle the contradictions. Moreover, these works typically focus on a centralized rather than a distributed setting.

7. CONCLUSION

We have studied in this paper data management in the presence of contradictions arising from violations of functional dependencies. We have considered the problem first in the centralized setting, where we introduced, studied, and compared two different semantics, namely *nfat* and *nsat*. We have then extended the semantics to the distributed setting, where contradictions play a crucial role. For both the centralized and the distributed case, we have compared and contrasted our semantics to previously defined semantics for

queries with negation. Finally, we have presented a probabilistic approach for settling contradictions and proposed an approximation algorithm for computing probabilities of queries to hold in the fixpoint of the computation.

As always with semantics, we had to make design choices, and alternative choices are of interest. In particular, our semantics is based on considering all possible runs with inflationary semantics. In such a run, in the distributed case, each peer is "stubborn" and never changes his mind. In runs where peers are allowed to change their mind in the presence of new information, convergence is no longer guaranteed. One could also for instance consider runs where peers may change their mind, but only a bounded number of times for each fact. If this bound is reached then they "give up" and decide not to believe any of them. As future work, we intend to study these and other possible semantics.

The implementation of systems based on the languages presented here also raises interesting issues. For instance, given a particular query and a required precision, one may want to optimize the usage of resources by deciding which subquery to ask to which peer to optimize the precision of answers. Also, when there are many answers to a query, one may want to generate the top-k ones, or only those above a certain probability threshold.

Acknowledgments. The authors are grateful to Emilien Antoine, Bruno Marnette, Neoklis Polyzotis, Marie-Christine Rousset, and Jules Testard for discussions on this work.

Appendix

Stable model semantics and well-founded semantics. We briefly recall the definition of stable model and well-founded semantics. Both are defined for ground datalog$^\neg$ programs. For a general datalog$^\neg$ program P and instance I, one simply considers the associated ground datalog$^\neg$ program consisting of all possible instantiations of the rules in P using the constants in P and I together with the bodyless rules corresponding to the facts in I. If (in)equalities appear in the body of rules in P, then after grounding, we remove rules which contain a falsified (in)equality atom, and then we remove from the bodies all remaining (in)equality atoms.

We start with stable model semantics. Given a ground datalog program with negation P and an instance J, the *reduct* of P with respect to J is the negation-free program obtained from P by first dropping from P all rules which have a negated body atom $\neg A$ such that $A \in J$, and then removing all negated body atoms from the remaining rules. An instance J is called a *stable model* of P if J is the minimal model of the reduct of P w.r.t. J.

Now we recall well-founded semantics, which is based on 3-valued instances. Formally, a 3-valued instance is a function $\mathbf{I}$ mapping each fact to either 0 (false), 1/2 (unknown), or 1 (true). The truth value of a Boolean formula α w.r.t. a 3-valued instance $\mathbf{I}$ is denoted $\hat{\mathbf{I}}(\alpha)$ and defined as follows: $\hat{\mathbf{I}}(\beta_1 \wedge \beta_2) = \mathsf{min}(\hat{\mathbf{I}}(\beta_1), \hat{\mathbf{I}}(\beta_2))$, $\hat{\mathbf{I}}(\neg\beta) = 1 - \hat{\mathbf{I}}(\beta)$, and $\hat{\mathbf{I}}(\beta \leftarrow \gamma) = 1$ if $\hat{\mathbf{I}}(\beta) \leq \hat{\mathbf{I}}(\gamma)$, and otherwise 0. Given a ground datalog program with negation P, we say that a 3-valued instance $\mathbf{I}$ is a model of P if it assigns a truth value to all facts in P and makes every rule in P evaluate to true. The immediate consequence operator can be straightforwardly extended to handle so-called *extended datalog programs* which allow truth values in the rule bodies. Repeated applications of the immediate consequence operator yields a least fixpoint, called the minimal model of the extended datalog program. Given a ground datalog program with negation P and a 3-valued instance $\mathbf{J}$, the *(3-valued) reduct* of P with respect to $\mathbf{J}$ is the extended datalog program obtained from P by replacing each negated body atom $\neg A$ by its truth value under $\mathbf{J}$. An instance $\mathbf{J}$ is called a *3-stable model* of P if $\mathbf{J}$ is the minimal model of the reduct of P w.r.t. $\mathbf{J}$. The *well-founded semantics* of P is the 3-valued instance which assigns true (resp. false) to a fact A if A has value true (false) in every 3-stable model of P; all other facts are assigned unknown. This 3-valued instance is itself a 3-stable model (the minimal one). We refer the reader to Chapter 15 of [**?**] for more details.

Translating a distributed system to a centralized datalog program. We provide the formal definition of a translation of a Webdamlog system (I, Γ, F) over a finite set Π of peers into a classical datalog program constrained by FDs.

Definition 8. *Given a system (I, Γ, F), its centralized counterpart is $c(I, \Gamma, F) = (I_c, P_c, F_c)$ where:*

- $I_c = \cup_{p\in\Pi} I(p)$.
- *P_c consists of all possible instantiations of the peer variables in the rules of $\cup_{p\in\Pi}\Gamma_p$ with peers from Π (where only instantiations respecting the typing and arity constraints are allowed).*

- $F_c = \cup_{p \in \Pi} F(p)$.

Given a state $\sigma = (I, \Gamma, F, D, M)$, its set of facts is defined as follows:

$$Facts(\sigma) = \cup_p (\, I(p) \cup M(p) \,)$$

Finally, we say that W is a possible world *of a system (I, Γ, F) if there exists a run for (I, Γ, F) with final state σ such that $W = Facts(\sigma)$.*

Proofs. We next provide theorem proofs that were omitted from the paper body due to lack of space.

Proof of Theorem 1. Throughout the proof, we will use $Gr(P, F, I)$ to denote the ground datalog$^\neg$ program associated with (P, F, I). This program is obtained by first considering all full instantiations of rules in $NegProg(P, F)$ using the constants from P and I, as well as all rules of the form A :- for $A \in I$. We then remove all rules which have a body atom $a \neq a$ or a body atom $a = b$ where $a \neq b$, and we drop all (in)equality atoms from the remaining rules.

For the first direction, suppose that $J \in pw^{nfat}(I, P, F)$ is obtained by the sequence $I = I_0 \rightarrow_{nfat} I_1 \rightarrow_{nfat} \ldots \rightarrow_{nfat} I_n = J$ of applications of the immediate consequence operator. Consider the rules of $NegProg(P, F)$ which have a predicate C_R in their heads, and let J' be the set of facts which can be derived from J using these rules. Note that J' coincides with J on the schema of (P, I). Taking the reduct of $Gr(P, F, I)$ w.r.t. J' yields a ground negation-free datalog program *Red*. We aim to show that J' is precisely the minimal model $mm(Red)$ of *Red*.

We begin by showing using induction that $I_j \subseteq mm(Red)$ for all $0 \leq j \leq n$. The base case is trivial: $I_0 = I \subseteq mm(Red)$ since there is a rule A :- in $Gr(P, F, I)$, hence in *Red*, for every $A \in I$. For the induction step, suppose that $I_k \subseteq mm(Red)$. Let $R(c_1, \ldots, c_n)$ be the fact in $I_{k+1} \setminus I_k$, and let $R(\$U_1, \ldots, \$U_n)$:- $Body$ be the rule in P whose instantiation $R(c_1, \ldots, c_n)$:- $Body'$ takes I_k to I_{k+1}. We know that there is a rule

$$R(\$U_1, \ldots, \$U_n) \text{ :- } Body, \neg C_R(\$U_1, \ldots, \$U_n)$$

in $NegProg(P, F)$, and hence the rule

$$R(c_1, \ldots, c_n) \text{ :- } Body', \neg C_R(c_1, \ldots, c_n)$$

will belong to $Gr(P, F, I)$. We next note that $C_R(c_1, \ldots, c_n) \notin J'$ since otherwise there would be an FD $R : X \rightarrow Y$ and some fact $R(d_1, \ldots, d_n) \in J$ such that $(c_1, \ldots, c_n)$ and $(d_1, \ldots, d_n)$ match on X but differ on Y. However, that would mean that J contains facts which violate some FD in F, which is impossible by the definition of possible worlds. Since we know that $C_R(c_1, \ldots, c_n) \notin J'$, it follows that the rule $R(c_1, \ldots, c_n)$:- $Body'$ will belong to *Red*. From the induction hypothesis, we have that $Body' \subseteq mm(Red)$, which means $R(c_1, \ldots, c_n)$ must also belong to $mm(Red)$, so $I_{k+1} \subseteq mm(Red)$. We have thus shown that $J = I_n \subseteq mm(Red)$. To complete our proof that $J' \subseteq mm(Red)$, consider some fact $C_R(c_1, \ldots, c_n) \in J' \setminus J$. We know from the definition of J' and the structure of $Neg(P, F)$ that $C_R(c_1, \ldots, c_n)$ can be derived from J using a single rule in $NegProg(P, F)$ of the form

$$\begin{aligned} C_R(\$U_1, ..., \$U_n) \text{ :- } \quad & R(\$V_1, ..., \$V_n), \$U_{j_1} = \$V_{j_1}, \ldots, \\ & \$U_{j_\ell} = \$V_{j_\ell}, \$U_k \neq \$V_k \end{aligned}$$

It follows that there must be some fact $R(d_1, \ldots, d_n) \in J$ such that $(c_1, \ldots, c_n)$ and $(d_1, \ldots, d_n)$ match on X but differ on Y. As *Red* contains the rule

$$C_R(c_1, ..., c_n) \text{ :- } R(d_1, ..., d_1)$$

and $mm(Red)$ contains $R(d_1, \ldots, d_n)$, the atom $C_R(c_1, \ldots, c_n)$ must also appear in $mm(Red)$.

For the other inclusion, let $K_0 = \emptyset, K_1, \ldots, K_m$ be a derivation of $mm(Red)$ one fact at a time. We show by induction that $K_j \subseteq J'$ for all $0 \leq j \leq m$. The base case is trivial. For the induction step, suppose $K_i \subseteq J'$, and let $R(c_1, \ldots, c_n)$:- $Body$ be the rule taking K_i to K_{i+1}. Note that by the induction hypothesis, we have $Body \subseteq J'$. First consider the case where $R(c_1, \ldots, c_n)$:- $Body$ is the grounding of some rule in P. Then since $Body$ only contains relations from (P, I), we can infer from $Body \subseteq J'$ that $Body \subseteq J$. Suppose for a contradiction that $R(c_1, \ldots, c_n) \notin J'$. This implies $R(c_1, \ldots, c_n) \notin J$, since $J' \setminus J$ only contains facts using the new relation symbols. Because $J \in pw^{nfat}(I, P, F)$, the absence of $R(c_1, \ldots, c_n)$ in J means that there is some $R(d_1, \ldots, d_n) \in J$ such that $\{R(c_1, \ldots, c_n), R(d_1, \ldots, d_n)\}$ violates an FD $R : X \rightarrow Y$ in F. However, this means that $C_R(c_1, \ldots, c_n)$ belongs to J', and so $R(c_1, \ldots, c_n)$:- $Body$ does not appear in *Red*, a contradiction. The only other type of rule which can appear in *Red* takes the form

$$C_R(c_1, \ldots, c_n) \text{ :- } R(d_1, \ldots, d_n)$$

where $(c_1, \ldots, c_n)$ and $(d_1, \ldots, d_n)$ agree on X and disagree on some $y \in Y$ for some FD $R : X \rightarrow Y$ in F. We know that $R(d_1, \ldots, d_n) \in J'$, and that $NegProg(P, F)$ contains the rule

$$\begin{aligned} C_R(\$U_1, ..., \$U_n) \text{ :- } \quad & R(\$V_1, ..., \$V_n), \$U_{j_1} = \$V_{j_1}, \ldots, \\ & \$U_{j_\ell} = \$V_{j_\ell}, \$U_y \neq \$V_y \end{aligned}$$

where $X = \{j_1, \ldots, j_\ell\}$. It follows then from the definition of J' that we must also have $C_R(c_1, \ldots, c_n) \in J'$.

For the other direction, suppose J' is a stable model of $NegProg(P, F)$ w.r.t. to I, or equivalently, that J' is a stable model of the ground datalog$^\neg$ program $Gr(P, F, I)$. We use J to denote the restriction of J' to the schema of (P, I), and we let *Red* denote the reduct of $Gr(P, F, I)$ w.r.t. to J'. We know that J' is the minimal model of *Red*, and that it contains I. Moreover, because of the structure of the rules in *Red*, it is possible to derive all facts in J before producing the remaining C_R-facts. So we can find a sequence $K_0 = I, K_1, \ldots, K_m = J$ which is a fact-at-a-time derivation of J from I using the rules in *Red*. We will show by induction on i that $K_0 = I \rightarrow^*_{nfat} K_i$ w.r.t. P and F for all $0 \leq i \leq m$. The base case ($i = 0$) is trivial since $\rightarrow^*_{nfat}$ is reflexive. For the induction step, suppose that $K_0 \rightarrow^*_{nfat} K_i$ for all $1 \leq i \leq j$, and consider the rule ρ from *Red* which was used to go from K_j to K_{j+1}. As J only contains facts on the schema of (P, I), we know that ρ must be the grounding $R(c_1, \ldots, c_n)$:- $Body$ of some rule in P. As all of the atoms in $Body$ are present in K_j, we must have $K_j \rightarrow_{nfat} K_{j+1}$ except if adding $R(c_1, \ldots, c_n)$ to K_j violates some FD in F. If the latter holds, we can find some $R(d_1, \ldots, R_n) \in K_j$ and some FD $R : X \rightarrow Y$ such that $(c_1, \ldots, c_n)$ and $(d_1, \ldots, d_n)$ agree on X and disagree on some $y \in Y$. Note however that *Red* will contain a rule

$$C_R(c_1, \ldots, c_n) \text{:-} R(d_1, \ldots, R_n)$$

and so $R(d_1, \ldots, R_n) \in J'$ implies that $C_R(c_1, \ldots, c_n)$ must

also belong to J'. However, this is a contradiction since if $C_R(c_1, \ldots, c_n) \in J'$, then the rule ρ cannot appear in *Red*. This means we must have $K_j \rightarrow_{nfat} K_{j+1}$, which yields $K_0 \rightarrow^*_{nfat} K_{j+1}$. We have thus shown $I \rightarrow^*_{nfat} J'$, and all that remains to be shown is that there is no superset J'' of J such that $I \rightarrow^* J''$. Suppose for a contradiction that $J \rightarrow_{nfat} J''$ where $J \subsetneq J''$. Then there is an instantiation of a rule in P such that all body atoms appear in J and the head of the rule is consistent with J given the FDs. Then this instantiated rule will appear in *Red*, and a new fact not in J' can be derived, which contradicts the fact that J' is the minimal model of *Red*. □

Proof of Theorem 2. We note that we can use a proof tree for A w.r.t. (I, P, F) to construct a sequence $I = I_0 \rightarrow I_1 \rightarrow \ldots \rightarrow I_n$ such that $A \in I_n$. Then we can simply extend this to a possible world by applying rules until a fixpoint under $\rightarrow_{nfat}$ is reached. For the other direction, given a possible world generated by the sequence $I = I_0 \rightarrow I_1 \rightarrow \ldots \rightarrow I_n$ which contains A, we use induction to construct proof trees for all facts in I_n as follows. For facts in $I = I_0$, the single-node tree with root labelled with the fact is a proof tree. Otherwise, suppose we have proof trees for all facts in I_{k-1}, and let A be the unique fact in $I_k \setminus I_{k-1}$. Choose some rule A :- $B_1, \ldots, B_m$ whose body holds in I_{k-1}. We already have proof trees for $B_1, \ldots, B_m$, so we simply have to add a new root node labelled A, which we connect to the trees for $B_1, \ldots, B_m$. Note that because I_k is consistent, so the set of labels of nodes in this tree will also satisfy the FDs. □

Proof of Theorem 3. First suppose $A \notin cert^{nfat}(I, P, F)$, and let W be a possible world such that $A \notin W$. There are two possible explanations for the absence of A in W. The first is that there is a fact $B \in W$ such that $\{A, B\}$ violates some FD, and so it is not possible to derive A. The second is that there is no rule in P with head A whose body holds in W. In the former case, we can build a refuting proof tree for A by taking a proof tree for B (which involves only facts from W) and adding a new root node $\neg A$. In the latter case, we can find atoms $A_1, \ldots, A_k$ such that $A_i \notin W$ for all $1 \leq i \leq k$, and each (instantiation of a) rule in P which has head A contains some A_i in its body. If we already had refuting proof trees for $A_1, \ldots, A_k$ (whose positive atoms belong to W), we could simply add them as children to a node $\neg A$, and this would yield a refuting proof tree for A. The idea is thus to apply precisely the same reasoning to each A_i, and then to the children of the A_i, and so on. The important point is that if we ever reach a negated atom $\neg B$ such that the label $\neg B$ appeared higher in the recursive process, we stop and make $\neg B$ a leaf node. As at each step, we ensure that the current node satisfies one of the requirements of the definition, and we only allow positive facts which belong to W. We can thus be sure that the result of this process is indeed a refuting proof tree for A.

For the other direction, suppose there is a refuting proof tree T for fact A w.r.t. (I, P, F). Let Σ be the set of positive facts appearing in the labels of T. As Σ is known to satisfy the FDs in F, it can be extended to a possible world W. We claim that the fact A does not appear in W. To show this, we will prove the following statement: a possible world cannot contain all positive facts in some proof tree for a fact as well as all positive facts for some refuting proof tree for that fact. As we know W contains all positive facts in the refuting proof tree T for A, this allows us to conclude that for every proof tree for A, some positive fact in the tree does not appear in W, and hence A cannot belong to W. It follows that A is not certain.

In order to prove the above statement, we assume for a contradiction that we have a possible world U, a proof tree T^+ for a fact B, and a refuting proof tree T^- for B such that U contains all positive facts appearing in T^+ and all positive facts appearing in T^-. We assume without loss of generality that T^+ has minimal depth given U, i.e. there does not exist a fact B_2, a proof tree T_2^+ for B_2 of depth strictly less than that of T^+, and a refuting proof tree T_2^- for B_2 such that U contains all positive facts appearing in the trees T_2^+ and T_2^-. First we consider the case where T^+ has depth 0, i.e. it consists of a single node labelled B. This yields a contradiction, since this implies that $B \in I$, and hence that there cannot be a node $\neg B$ in T^-. The other possibility is that the root of T^+ is labelled B and has children $C_1, \ldots, C_m$, where B :- $C_1, \ldots, C_m$ is a rule in P. By the definition of a refuting proof tree, there are two possibilities for the root node of T^-. The first is that the root is labelled $\neg B$ and there are no rules in P with head B. This is a contradiction since we know that the rule B :- $C_1, \ldots, C_m$ belongs to P. The second possibility is that the root has children $\neg D_1, \ldots, \neg D_n$, where for every rule in P concluding on B, there is some D_j which appears in its body. It follows that we can find i, j such that $C_i = B_j$. This contradicts the minimality of T^+ given U: simply consider the subtree of T^+ rooted at C_i and the subtree of T^- rooted at $B_j = C_i$. □

Proof of Theorem 4. Membership is straightforward: guess a possible world which contains (or does not contain) the fact of interest. For NP-hardness of possible answers, we give a reduction from SAT. Let φ be a propositional CNF. We will encode $\varphi = c_1 \wedge \ldots \wedge c_n$ over variables $v_1, \ldots, v_k$ using the instance I_φ defined as follows:

$$
\begin{array}{rl}
var(v_j) & 1 \leq j \leq k \\
pos(c_i, v_j) & v_j \in c_i \\
neg(c_i, v_j) & \neg v_j \in c_i \\
next(c_i, c_{i+1}) & 1 \leq i < n \\
start(c_1), end(c_n) &
\end{array}
$$

We also define a datalog program P which contains the rules:

$$
\begin{array}{l}
val(v, 0) \text{ :- } var(v) \\
val(v, 1) \text{ :- } var(v) \\
csat(c) \text{ :- } pos(c, v), val(v, 1) \\
csat(c) \text{ :- } neg(c, v), val(v, 0) \\
chainsat(c) \text{ :- } start(c), csat(c) \\
chainsat(c) \text{ :- } chainsat(d), next(d, c), csat(c) \\
sat \text{ :- } end(c), chainsat(c)
\end{array}
$$

The set F consists of a single FD $val : 1 \rightarrow 2$. Because of the FD on *val*, the set of possible worlds of corresponds precisely to the possible valuations of the variables. The rest of the program checks whether for a given valuation all clauses are satisfied, in which case *sat* holds. It can be verified that φ is satisfiable if and only if *sat* is a possible answer of (I_φ, P, F). As P and F are independent of the input 3CNF φ, this yields NP-hardness in data complexity.

To show co-NP-hardness of deciding certainty, we give a reduction from unsatisfiability of 3CNF. We will encode a

particular 3CNF φ with an instance I_φ defined as follows:

$var(v)$	v is variable of φ
$lit(i, c, v, 1)$	v is the i-th literal of clause c
$lit(i, c, v, 0)$	$\neg v$ is i-th literal of clause c

We also define a datalog program P which contains the rules:

$$unsat \text{ :- } u(c) \quad val(v, 0) \text{ :- } var(v) \quad val(v, 1) \text{ :- } var(v)$$
$$neq(0, 1) \text{ :-} \quad neq(1, 0) \text{ :-}$$

$$u(c) \text{ :- } \quad lit(1, c, v_1, p_1), val(v_1, w_1), neq(p_1, w_1),$$
$$lit(2, c, v_2, p_2), val(v_2, w_2), neq(p_2, w_2),$$
$$lit(3, c, v_3, p_3), val(v_3, w_3), neq(p_3, w_3)$$

The set F consists of a single FD $val : 1 \to 2$. It can be easily verified that φ is unsatisfiable if and only if $unsat$ is a certain answer of (I_φ, P, F). As P and F are independent of the input 3CNF φ, this yields co-NP-hardness in data complexity. $\square$