Deduction in the Presence of Distribution and Contradictions *

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 dependency* (FD) violation. We propose a simple non-deterministic semantics for datalog with FDs based on inferring facts one at a time, never violating the FDs. We present a novel *proof theory* for this semantics. 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 build upon a distributed datalog idiom, namely *Webdamlog*, to define a semantics for the *distributed setting*. Observe that contradictions naturally arise in such a setting, with different peers having conflicting information or opinions. We study different semantics for this setting.

1. INTRODUCTION

Our goal is to study how peers in a network exchange information and reason together to reach agreements. A difficulty is that the peers may state or infer conflicting facts, where conflicts are captured here by violations of functional dependencies. Peers can settle conflicts by choosing between contradicting base or inferred facts. Such choices add to the uncertainty already inherent in an asynchronous environment. We study the semantics of datalog-like languages and query evaluation in this setting.

We introduce, as a basis for our work, a semantics for *datalog* in the presence of *functional dependencies* (FDs for short). We then extend that semantics to a distributed datalog language, namely Webdamlog.

Datalog with FDs. As a starting point, we consider a centralized setting. We propose a nondeterministic semantics for datalog in presence of FDs. The intuition behind this semantics is that a fact is derived only if its introduction into the database does not result in any FD violation (i.e., if it is not 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 refer to this semantics as non-deterministic fact-at-a-time (*nfat*) semantics, and we also provide a novel *proof theory* for the semabtics. 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 and contrast it with *nfat*.

Note that semantics for datalog with functional dependencies have already been considered, from a somewhat different perspective (see discussion in Sections 2 and 5). We revisit the topic and present new results, such as the proof theory. We then move to considering 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. The semantics we propose is meant to handle such contradictions.

For capturing knowledge inference in a distributed setting, we use a recently proposed distributed datalog idiom called Webdamlog [1]. We "marry" the centralized semantics for datalog with FDs and the semantics of a positive fragment of Webdamlog, and study the resulting semantics.

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. We provide an overview of related works in Section 5, and conclude in Section 6. Due to space limi-

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

tations, proofs are omitted and can be found in [12].

2. DATALOG WITH FDs

In this section, we study the management of contradictions arising from considering datalog together with functional dependencies. We begin by presenting a semantics based on fact inferences (forward chaining), and then present a corresponding proof theory (backward chaining).

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. 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 non-determinism since the result of the process possibly relies on the order of rule activation. We consequently refer to this semantics as *nfat*, standing for *non-deterministic fact-at-a-time*.

We next formally define the semantics, by re-defining the immediate consequence and the consequence operators in the presence of FDs. Note that we assume that the extensional database does not violate the FDs. By construction, the database instances that will be generated will also always be 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 (inflationary)* immediate consequence operator $\rightarrow_{nfat}$ *is defined as follows: $I \rightarrow_{nfat} I \bigcup \{A\}$ if and only if there exists an instantiation A :- $A_1, ..., A_n$ of a rule in P such that $\{A_1, ..., A_n\} \subseteq I$ and $I \bigcup \{A\}$ satisfies the FDs. The* consequence operator $\rightarrow^*_{nfat}$ *is the 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, containing:

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

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. $poss^{nfat}(I, P, F)$). The problem of deciding whether a given fact is certain (possible) with respect to *nfat* is denoted $CERT^{nfat}$ ($POSS^{nfat}$).

Connection with negation. There is a strong connection between FDs and negation (see e.g. [11, 14]). In particular, using a construction similar to the one in [14] we can translate a datalog program P with FDs to a datalog program P' with negation, such that possible worlds for P according to our semantics are exactly the stable models of P'. The exact translation appears in [12].

Proof theory

We next equip the previous semantics with a proof theory.

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* finite *tree labelled with facts, such that the root is labelled A and each node satisfies one of the following:*

1. *It is a leaf node 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 no two facts that label nodes of the tree, violate an FD.

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

THEOREM 1. *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 2. *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 introducing the notion of a *refuting proof tree* for A, defined 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 finite 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 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.*

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 has B for head 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 3. *Let $I = \{A\}$, $P = \{R(a,0)$:- $R(a,1), R(a,1)$:- $R(a,0), R(a,2)$:- $A, R(b,1)$:- $A, R(b,2)$:- $A\}$, 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 $\neg R(a,0)$ (according to rule 3 in the definition) which in turn has a unique child $\neg R(a,1)$ (rule 3). To show that $R(b,1)$ is not certain we can use a refuting proof tree whose root node $\neg R(b,1)$ has a unique child $R(b,2)$ (rule 4) which has a unique child A (rule 2, and the existence of A is due to rule 1). 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 2. *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.

THEOREM 3. *$POSS^{nfat}$ is NP-complete in the size of the input instance. $CERT^{nfat}$ is coNP-complete in that size.*

3. SET-AT-A-TIME SEMANTICS

In the previous section, we have considered a fact-at-a-time semantics for datalog with FDs. We next consider an alternative semantics based on inferring in one step all the facts that are immediate consequences of the facts and the rules, and then non-deterministically choosing a subset of those new facts 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 4. *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. Similarly to the case of the *nfat* semantics, we use $POSS^{nsat}$ and $CERT^{nsat}$ (resp.) to denote the problems of deciding possibility and certainty with respect to the *nsat* semantics.

THEOREM 4. *$POSS^{nsat}$ (resp. $CERT^{nsat}$) is NP-complete (resp. coNP-complete) in the size of the input instance I.*

Comparing with *nfat*, the following theorem holds:

THEOREM 5. *For each input instance I, datalog program P, and set F of FDs: (i) each nsat possible world is an nfat possible world; and (ii) the converse does not hold in general.*

Note that it follows 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. Consequently, the proof theory developed for *nfat* needs to be refined to account for *nsat*; developing a proof theory for *nsat* will be addressed in future research.

REMARK 1. *In the following section, we will focus mainly on the set-at-a-time semantics rather than on the fact-at-a-time one (that is more logically founded). A main reason for considering the set-at-a-time semantics is that it can be implemented using a relational engine. Another reason is that, for the distributed setting, we will use a previously introduced language, namely Webdamlog, that relies on a set-at-a-time semantics. However, for completeness, we revisit this choice towards the end of the section to consider alternative semantics, based on fact-at-a-time derivations.*

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 [1]. We will consider a relevant fragment of this language, referred to as s-Webdamlog (for "simple" Webdamlog). We start with a simple example to illustrate the language.

EXAMPLE 5. *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 : $1 \rightarrow 2$ and the following rules:*

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

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:

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

s-Webdamlog

We consider only a subset of the Webdamlog language, referred to as s-Webdamlog. The semantics we propose may be extended to the full language in a straightforward manner. However, our results do not hold for the general language (see [12] for discussion).

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.). 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. It is because *data* includes *peer* and *relation* that we may write atoms such as $IsIn@\$P(\$X, \$Y)$ (in the head of the first rule in Example 5) where the peer $\$P$ is a variable.

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, resp., 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 which peer the derived fact concerns.

Note that unlike in the general language of [1], 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 [1], 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, 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) [2], and (ii) alerting other peers of 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 to each peer p a set $I(p)$ of extensional p-facts, Γ assigns to each peer a set of rules, and D assigns to each peer p a set $D(p)$ of rules that have been delegated to p. 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) that satisfies *fairness*, i.e. each peer p is invoked infinitely many times.

Semantics of s-Webdamlog with FDs

We now extend s-Webdamlog to account for FDs that constrain local relations. We must select a FD-sensitive semantics to specify the local computation. For the reasons mentioned in Remark 1 we will focus first on *nsat*; we revisit this choice towards the end of the section.

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 [3]. 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 6. *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* $R@q(0,1)$ *:- that is delegated. Note however that no changes are made to* $I(p)$, $\Gamma(p)$, *or its dependencies* $F(p)$ *to record the choice that is made. So, if p is called again immediately, he will make the same choice.*

The semantics we use guarantees that peers always make consistent choices. 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 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

[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.

[2] An alternative is to make a single step of derivations at each move, in the spirit of the fact-at-a-time semantics. We discuss this alternative towards the end of the section.

[3] Note that at the move of a peer p, it may cause contradicting facts to be derived at a peer q. When q moves, it will solve these contradictions

course of a run and that they grow monotonically. We are thus guaranteed that each run converges:

THEOREM 6. *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 $k \geq i$, $\sigma_i = \sigma_k$.*

We thus often identify a run with its finite prefix up to convergence, and we speak of the *final state* of a run.

Possibility and certainty. 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 (Thm. 4) carry to the distributed case.

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 [1] 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 7. *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 [1] 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 different.

Simulation with Webdamlog

We can show that the semantics we have defined for s-Webdamlog with FDs can be "simulated" in "almost" standard Webdamlog with negation (up to some non-deterministic choices). We next provide the intuition for such simulation; details can be found in the full version [12].

For each local intensional relation $R@p$, we use an extensional relation $R_m@p$ satisfying the same FDs as $R@p$, intuitively recording prior choices between contradicting facts, made by the peer. 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. To block the execution of (ii) until (i) is performed, we add a proposition *wait* in the program rules and use a rule *wait* :- that unblocks the rules. 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.

A last difficulty is that the standard Webdamlog with negation we have considered are deterministic whereas the semantics of Webdamlog in presence of FDs, is not. To this end, we non-deterministically choose a total ordering of the tuples in each relation; this total ordering dictates a preference relation, i.e. how to choose among conflicting facts, if they are derived at the same time. See [12] for details.

Using fact-at-a-time semantics

As noted above, we have focused in the distributed case on a set-at-a-time semantics. This choice affected the local semantics of each peer, as well as the semantics of moves. For local semantics we can also use a fact-at-a-time semantics, and for moves we can use a "one step" semantics (i.e. use $\rightarrow_{nfat}$ or $\rightarrow_{nsat}$ rather than $\rightarrow^*_{nfat}$ or $\rightarrow^*_{nsat}$) [4]. This leads to 4 different semantics. We next revisit our development and consider the implications of such semantic choices.

Convergence. Since all proposed semantics are inflationary, we can show that runs are guaranteed to converge, i.e. a counterpart of Thm. 6 continues to hold.

Possibility and certainty. Since possibility and certainty were shown to be NP-hard for both *nfat* and *nsat*, it is easy to show that they remain NP-hard in the distributed case even if we use *nfat* semantics. It is interesting to recall in this context the connection to proof theory. 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. In a distributed setting, showing possibility or uncertainty can sometimes be done using a proof that uses only the local facts of some peer. However, in the worst case, the proof may require facts which are distributed amongst the peers, in which case discovering the proof may require broadcast communication.

Comparison with the centralized case. Recall that for the *nsat* semantics we have shown that the sets of possible worlds for the distributed and centralized systems may be distinct. 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 the "one step" semantics based on *nfat* (although, as mentioned above, the practicality of this semantics is questionable).

Simulation with Webdamlog. Webdamlog (with no FDs) is based on a set-at-a-time semantics. Simulating the *nfat* semantics using Webdamlog is an open problem.

[4]This "one step" semantics is not satisfying from a practical viewpoint as it requires communication after every inference step. Nevertheless, we discuss it for completeness.

5. RELATED WORK

Different semantics have been proposed for datalog extended with negation including inflationary semantics, stable model semantics, and well-founded semantics (cf. Chapters 14-15 of [2] for an introduction). We mentioned a connection between the *nfat* semantics and stable model semantics, similar to the one shown in [14].

Also relevant is the work on datalog extended with nondeterministic witness [3] or choice constructs [11, 14]. At a formal level, there are strong similarities between our *nfat* possible worlds and the stable choice models of [14], and between our *nsat* semantics and the eager dynamic choice semantics of [11]. 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.

There is a large literature on *consistent query answering* (see [9, 10] for surveys) 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 [8]. Recent work on the repair-checking problem [4] 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 and on inconsistency-handling for description logics [17].

Handling contradictions in a multi-agent environment was recently studied in different lines of works [13, 16, 15], however the focus there was on data sharing or corroboration of opinions, rather than on semantics for inference under the presence of FDs. We also note that recent work on distributed declarative systems [6, 5] have taken a different approach, focusing on eventual consistency rather than settling contradictions after every move, and thus have not addressed the semantic issues studied here.

6. 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 and studied 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 our semantics to previously defined semantics for queries with negation.

The semantics introduced in this paper leads to uncertainty. In the spirit of probabilistic databases (e.g. [18]), we are working on "measuring" the uncertainty using probabilities. Note that there are different sources of uncertainty in our setting. In particular, uncertainty arises from choices made to resolve conflicts, but is also inherent to a distributed setting, due to the asynchronicity of the peers.

We also intend to extend the present work to general Webdamlog programs (Recall that in Section 4 we have assumed that the rules are all local, positive, and deductive.) We will further explore other possible semantics for handling contradictions. Finally, we intend to incorporate some of these ideas in a system that is currently being implemented [7].

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

7. REFERENCES

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

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

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

[4] F. N. Afrati and P. G. Kolaitis. Repair checking in inconsistent databases: algorithms and complexity. In *ICDT*, pages 31–41, 2009.

[5] P. Alvaro, N. Conway, J. Hellerstein, and W. Marczak. Consistency analysis in bloom: a calm and collected approach. In *CIDR*, 2011.

[6] T. J. Ameloot, F. Neven, and J. Van den Bussche. Relational transducers for declarative networking. In *PODS*, 2011.

[7] E. Antoine, A. Galland, K. Lyngbaek, A. Marian, and N. Polyzotis. Social networking on top of the webdamexchange system. In *ICDE*, 2011.

[8] M. Arenas, L. E. Bertossi, and J. Chomicki. Answer sets for consistent query answering in inconsistent databases. *TPLP*, 3(4-5):393–424, 2003.

[9] L. Bertossi. Consistent query answering in databases. *SIGMOD Rec.*, 35:68–76, June 2006.

[10] J. Chomicki. Consistent query answering: Five easy pieces. In *ICDT*, pages 1–17, 2007.

[11] L. Corciulo, F. Giannotti, D. Pedreschi, and C. Zaniolo. Expressive power of non-deterministic operators for logic-based languages. In *Workshop on Deductive Databases and Logic Programming*, 1994.

[12] Full version. `http://www.cs.bgu.ac.il/~deutchd/WDLFull.pdf`.

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

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

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

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

[17] R. Rosati. On the complexity of dealing with inconsistency in description logic ontologies. In *IJCAI*, 2011.

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