Consistency and Provenance in Rule Processing

Eric Jui-Yi Kao

Computer Science Department
Stanford University
Stanford, CA 94305
United States of America
erickao@cs.stanford.edu

Abstract. Open collections of data and rules on the web are typically characterized by heterogeneous quality and imperfect consistency. In reasoning with data and rules on the web, it is important to know where an answer comes from (provenance) and whether the it is reasonable considering the inconsistencies (inconsistency-tolerance). In this paper, I draw attention to the idea that provenance and inconsistency-tolerance play mutually supporting roles under the theme of reasoning with imperfect information on the web. As a specific example, I make use of basic provenance information to avoid unreasonable answers in reasoning with rules and inconsistent data.

1 Introduction

Data and rules on the web are typically characterized by imperfect data, heterogeneity in quality, distributed authorship, semantic misalignments, and inconsistencies. Furthermore, the highly distributed nature of web data storage calls for the replication of data and the caching of computed (or intermediate) results. In reasoning in such a setting, two important requirements emerge:

1. Computed answers must be associated with information on the sources they rely on.
2. Inconsistencies must be tolerated: logical explosion and unreasonable answers must be avoided.

The first requirement calls for the computation and maintenance of provenance information on computed answers. The second requirement can be fulfilled by following an inconsistency-tolerant semantics in reasoning.

Much work has been done on provenance information in web data (for example [26, 16, 10, 11, 25]). The processing of inconsistent knowledge on the web has also gained attention [18, 23, 28, 29, 3]. In this paper, I draw attention to the idea that provenance and inconsistency-tolerance play mutually supporting roles in the overarching theme of reasoning with imperfect information on the web. As a specific example, I make use of basic provenance information to avoid unreasonable answers in reasoning with rules and inconsistent data.

In section 2, I give a motivating example. In section 3, I formally develop the *existential answers* semantics. In section 4, I show that the problem of computing existential answers is $\mathcal{NP}$-Complete, ruling out a popular rewriting technique which does not rely on provenance computation. Then, in sections 5 and 6, I give a provenance-based method for computing existential answers. In section 7, I give another characterization of existential answers via answer set programming. Finally, I conclude by discussing related work and future directions.

2 Motivating example

The data in table 1 will serve as example data. The BirthYear table associates each person with a birth year. Each person has at most one birth year, but in the example data two conflicting birth years are specified for Cody (and David) because they come from two sources, at least one of which is incorrect. The RecessionYear table gives the recessionary years in the United States.

Table 1. Example data

BirthYear

Person	Year
Adam	1980
Brian	1991
Cody	1984
Cody	1991
David	1980
David	1984

RecessionYear

Year
1980
1981
1982
1990
1991

Table 2. Cached/Materialized Results

RecessionBorn

Person
Adam
Brian
Cody
David

Rule:

```
RecessionBorn(X)  :-  BirthYear(X,Y), RecessionYear (Y)
```

OfAge

Person
Adam
Cody
David

Rule:

```
OfAge(X) :- BirthYear(X,Z),Z<1989
```

Consider the following scenario:
For a study, a researcher would like to send out surveys to those who were born during a recessionary year in the United States and are also legally of age. So adding to the rules in table 2, the researcher writes the rule:
`candidate(X)  :-  RecessionBorn (X) ∧ OfAge(X)`

Standard datalog semantics, which do not consider the constraints, would produce the answers {`Adam`, `Cody`, `David`}, the natural join between the two intensional relations. By digging down, we see that `RecessionBorn(Cody)` is derived from `BirthYear(Cody, 1991)` and `OfAge(Cody)` is derived from `BirthYear(Cody, 1984)`. But at least one of these two pieces of data is incorrect. We see that `Cody` is not a reasonable answer because whether we choose `BirthYear(Cody, 1984)` or `BirthYear(Cody, 1991)`, `Cody` does not satisfy the criteria intended by the query.

Consider the two conflicting tuples `BirthYear(David, 1980)` and `BirthYear(David, 1984)`. If we chose to believe `BirthYear(David, 1980)`, then `David` is an answer. But if we chose to believe `BirthYear(David, 1984)`, then `David` is not an answer. Since we have no information that leads us to believe one of the two conflicting tuples over the other, `David` is not an unreasonable answer.

The notion of *existential answers* (definition 4 in the next section) formalizes this intuition, admitting answers which are reasonable and rejecting answers which are unreasonable. The idea is to reason from consistent subsets of data so that conflicting pieces of information cannot be combined to give an unreasonable answer.

3 Formal Development

In this section, I develop the formal definitions necessary to define *existential answers*. The development follows the definition of datalog in [1].

Let **dom** be a fixed, countably infinite set which serves as the *underlying domain*. For example, **dom** may be the universe of all possible URI's. A *constant* is an element of **dom**. A schema S is a nonempty, finite set of relation names, each with a fixed arity. An instance of a schema S is a finite set of facts (ground atoms) over S and **dom**.

Let **var** be a fixed, countably infinite set of variables that is disjoint from **dom**. A *tuple* is a vector of elements from **dom**. A *free tuple* is a vector of elements from $\mathbf{dom} \cup \mathbf{var}$. I use $X, Y, Z, U, V, x, y, z, u, v$ and all their subscripted variants to denote variables. All other symbols appearing in a tuple are constants.

Definition 1 (Datalog syntax). *A rule is an expression of the form* $R_1(u_1) \ :- \ R_2(u_2), \ldots, R_n(u_n), \phi(u_\phi)$*, where* $n \geq 1, R_1, \ldots, R_n$ *are relation names,* $u_1, \ldots, u_n, u_\phi$ *are free tuples of appropriate arities, and* ϕ *is a boolean combination of built-in predicates (*$=, \neq, >$*,etc.). Each variable occurring in* u_1 *or* u_ϕ *must also occur in at least one of* $u_2, \ldots, u_n$*. A datalog program is a finite set of datalog rules.*

To give semantics to datalog programs, one interprets the rules as first-order sentences. Each rule $R_1(u_1) :- R_2(u_2), \ldots, R_n(u_n), \phi(u_\phi)$, is associated with the sentence $\forall x_1 \cdots x_q(\exists x_{q+1} \cdots x_m(R_2(u_2) \wedge \cdots \wedge R_n(u_n) \wedge \phi(u_\phi) \rightarrow R_1(u_1)))$, where $x_1, \ldots, x_q$ are the variables occurring in the head $(R_1(u_1))$ and $x_{q+1}, \ldots, x_m$ are the remaining variables occurring in the rule. For a datalog program P, the set of first-order sentences associated with P is denoted as Σ_P.

Definition 2 (Datalog Semantics). *Let P be a datalog program and D finite set of facts. A model of P in a finite set of facts satisfying Σ_P. The semantics of P on input D, denoted $P(D)$ is the minimum model of P containing D.*[1] *For a fact d, We write $\langle P, D\rangle \vdash_{\mathrm{dl}} d$ to denote that $d \in P(D)$.*

Definition 3 (Denial Constraints). *A denial constraint is a sentence of the form $\forall x_1 \cdots x_m(\neg R_1(u_1) \vee \cdots \neg R_n(u_n) \vee \phi(u_\phi))$, where $n \geq 1, R_1, \ldots, R_n$ are relation names, $u_1, \ldots, u_n, u_\phi$ are free tuples of appropriate arities, ϕ is a boolean combination of built-in predicates ($=, \neq, >$,etc.), and $x_1 \cdots x_m$ are the variables occurring in $u_1, \ldots, u_n, u_\phi$. Each variable occurring in u_ϕ must also occur in at least one of $u_1, \ldots, u_n$. A denial constraint can also be considered a "rule" with an empty head, i.e., $:- R_1(u_1), \ldots, R_n(u_n), \neg\phi(u_\phi))$.*

Definition 4 (Existential Answers). *Given a set of facts D, a set of rules P, and a set of constraints C, a fact d is an existential answers with respect to (D, P, C) if and only if there exists $D' \subseteq D$ such that $P(D')$ is a model of C and $\langle P, D'\rangle \vdash_{\mathrm{dl}} d$. We denote this fact as $\langle P, C, D\rangle \vdash_{\exists\mathrm{dl}} d$.*

Example 1. Here I apply the definition of existential answers to the example in section 2. For brevity, I use the following short-from relation names: `by` for `BirthYear`, `ry` for `RecessionYear`, `oa` for `OfAge`, `rb` for `RecessionBorn`, and `ca` for `candidate`. Let D be the base data (from table 1).

$D = \{$
$\quad \mathtt{by(Adam, 1980)}, \mathtt{by(Brian, 1991)}, \mathtt{by(Cody, 1984)}, \mathtt{by(Cody, 1991)},$
$\quad \mathtt{by(David, 1980)}, \mathtt{by(David, 1984)}, \mathtt{ry(1980)}, \mathtt{ry(1981)}, \mathtt{ry(1982)},$
$\quad \mathtt{ry(1990)}, \mathtt{ry(1991)}$
$\}$

Let P be the datalog program.

$P = \{$
$\quad \mathtt{oa(X)} :- \ \mathtt{by(X, Z)}, \mathtt{Z} < \mathtt{1989},$
$\quad \mathtt{rb(X)} :- \mathtt{by(X, Y)}, \mathtt{ry(Y)},$
$\quad \mathtt{ca(X)} :- \mathtt{rb(X)}, \mathtt{oa(X)}$
$\}$

Let C be the set of constraints.

$C = \{\forall XY(\neg by(X, Y) \vee \neg by(X, Z) \vee Y = Z)\}$

The collection of subsets D' of D such that $P(D')$ is a models of C are the following: (letting $F = \{\mathtt{by(Adam, 1980)}, \mathtt{by(Brian, 1991)}, \mathtt{ry(1980)}, \mathtt{ry(1981)}, \mathtt{ry(1982)}, \mathtt{ry(1990)}, \mathtt{ry(1991)}\}$)

[1] The minimum model always exists [1, pp 281, Theorem 12.2.3].

1. $\{\texttt{by}(\texttt{Cody}, 1984), \texttt{by}(\texttt{David}, 1980)\} \cup F$ and all its subsets
2. $\{\texttt{by}(\texttt{Cody}, 1984), \texttt{by}(\texttt{David}, 1984)\} \cup F$ and all its subsets
3. $\{\texttt{by}(\texttt{Cody}, 1991), \texttt{by}(\texttt{David}, 1980)\} \cup F$ and all its subsets
4. $\{\texttt{by}(\texttt{Cody}, 1991), \texttt{by}(\texttt{David}, 1984)\} \cup F$ and all its subsets

The corresponding sets of datalog answers (in the `ca` relation) are :

1. $\{\texttt{ca}(\texttt{Adam}), \texttt{ca}(\texttt{David})\}$ and all its subsets
2. $\{\texttt{ca}(\texttt{Adam})\}$ and all its subsets
3. $\{\texttt{ca}(\texttt{Adam}), \texttt{ca}(\texttt{David})\}$ and all its subsets
4. $\{\texttt{ca}(\texttt{Adam})\}$ and all its subsets

Hence, $\langle P, C, D\rangle \vdash_{\exists \mathrm{dl}} \texttt{ca}(\texttt{Adam})$ and $\langle P, C, D\rangle \vdash_{\exists \mathrm{dl}} \texttt{ca}(\texttt{David})$, but $\langle P, C, D\rangle \nvdash_{\exists \mathrm{dl}} \texttt{ca}(\texttt{Brian})$ and $\langle P, C, D\rangle \nvdash_{\exists \mathrm{dl}} \texttt{ca}(\texttt{Cody})$. The results agree with the intuition in section 2.

I present another example. This example contains recursive rules.

Example 2. A fact $\texttt{link}(\texttt{i}, \texttt{j}, \texttt{t})$ represents that there is a direct link of type `t` from `i` to `j`. Let the base facts $D = \{$
$\texttt{link}(\texttt{a}, \texttt{b}, 1), \texttt{link}(\texttt{a}, \texttt{c}, 2), \texttt{link}(\texttt{b}, \texttt{d}, 2), \texttt{link}(\texttt{c}, \texttt{d}, 3), \texttt{link}(\texttt{d}, \texttt{e}, 1), \texttt{link}(\texttt{d}, \texttt{f}, 2)\}$
Let the program $P = \{$
$\texttt{aReaches}(\texttt{Y}) :- \texttt{link}(\texttt{a}, \texttt{Y}, \texttt{U}),$
$\texttt{aReaches}(\texttt{Y}) :- \texttt{aReaches}(\texttt{a}, \texttt{Z}, \texttt{U}), \texttt{link}(\texttt{Z}, \texttt{Y}, \texttt{V})$
$\}$.
It defines which nodes are reachable from `a`.
Furthermore, assume it is known that no two links of the same type can coexist. It can be formalized as the following constraint set $C = \{$
$\forall X_1 Y_1 X_2 Y_2 U(\neg link(X_1, Y_1, U) \vee \neg link(X_2, Y_2, U) \vee X_1 = X_2),$
$\forall X_1 Y_1 X_2 Y_2 U(\neg link(X_1, Y_1, U) \vee \neg link(X_2, Y_2, U) \vee Y_1 = Y_2)$
$\}$
The collection of subsets D' of D that are models of $C \cup \Sigma_P$ are the following:

1. $\{\texttt{link}(\texttt{a}, \texttt{b}, 1), \texttt{link}(\texttt{a}, \texttt{c}, 2), \texttt{link}(\texttt{c}, \texttt{d}, 3)\}$ and all its subsets.
2. $\{\texttt{link}(\texttt{a}, \texttt{b}, 1), \texttt{link}(\texttt{b}, \texttt{d}, 2), \texttt{link}(\texttt{c}, \texttt{d}, 3)\}$ and all its subsets.
3. $\{\texttt{link}(\texttt{a}, \texttt{b}, 1), \texttt{link}(\texttt{d}, \texttt{f}, 2), \texttt{link}(\texttt{c}, \texttt{d}, 3)\}$ and all its subsets.
4. $\{\texttt{link}(\texttt{d}, \texttt{e}, 1), \texttt{link}(\texttt{a}, \texttt{c}, 2), \texttt{link}(\texttt{c}, \texttt{d}, 3)\}$ and all its subsets.
5. $\{\texttt{link}(\texttt{d}, \texttt{e}, 1), \texttt{link}(\texttt{b}, \texttt{d}, 2), \texttt{link}(\texttt{c}, \texttt{d}, 3)\}$ and all its subsets.
6. $\{\texttt{link}(\texttt{d}, \texttt{e}, 1), \texttt{link}(\texttt{d}, \texttt{f}, 2), \texttt{link}(\texttt{c}, \texttt{d}, 3)\}$ and all its subsets.

The corresponding sets of datalog answers (in the `aReaches` relation) are :

1. $\{\texttt{aReaches}(\texttt{n}) : n \in \{b, c, d\}\}$ and all its subsets
2. $\{\texttt{aReaches}(\texttt{n}) : n \in \{b, d\}\}$ and all its subsets
3. $\{\texttt{aReaches}(\texttt{n}) : n \in \{b\}\}$ and all its subsets
4. $\{\texttt{aReaches}(\texttt{n}) : n \in \{c, d, e\}\}$ and all its subsets
5. $\{\}$
6. $\{\}$

Hence, $\langle P, C, D\rangle \vdash_{\exists\mathrm{dl}}$ `aReaches(b)`, `aReaches(c)`, `aReaches(d)`, `aReaches(e)`, but $\langle P, C, D\rangle \not\vdash_{\exists\mathrm{dl}}$ `aReaches(f)`, `aReaches(a)`. Notice that even though $\langle P, D\rangle \vdash_{\mathrm{dl}}$ `aReaches(f)`, it is rejected as an existential answer because it is unreasonable given the constraints.

Based on the definition, a brute-force method for computing existential answers consists of enumerating each set $D' \subseteq D$, checking whether D' is admissible, and finally checking whether $D' \vdash_{\mathrm{dl}} d$. Because the number of subsets of D is $2^{|D|}$, this brute-force method is completely impractical. In section 6, I present a provenance-based method that constructs several special subsets of D and ignore all the other subsets.

4 Complexity and Rewriting

In typical data-oriented applications the set of facts is much larger than the set of rules. It is natural to consider the *data complexity* of rule evaluation. That is, we first fix the program P then consider the complexity of computing the answers given a set of facts as input.

For example, in datalog evaluation, the data complexity for a fixed program P is the complexity of the problem $\Delta_P = \{(D, d) : \langle P, D\rangle \vdash_{\mathrm{dl}} d\}$, where D ranges over the sets of facts and d ranges over the facts.

For a finite set of denial constraints C and a nonrecursive datalog program P, the existential answers problem is $\Xi_{P,C} = \{(D, d) : \langle P, C, D\rangle \vdash_{\exists\mathrm{dl}} d\}$.

If P is restricted to nonrecursive datalog programs, then, given the set of constraints C, it is possible to transform P into another program P' in the same language such that for all finite sets of facts D and all facts d, $\langle P, C, D\rangle \vdash_{\exists\mathrm{dl}} d$ if and only if $\langle P', D\rangle \vdash_{\mathrm{dl}} d$ [17, 19][2]. Because datalog evaluation is polynomial time in data complexity, the rewriting result shows that the existential answers problem is also polynomial time in data complexity. That is, given a finite set of denial constraints C and a nonrecursive datalog program P, $\Xi_{P,C}$ is in $\mathcal{P}$.

One may want to apply a similar rewriting approach in the recursive case, rewriting a recursive datalog program P into another datalog program P' such that the standard answers to P' are precisely the existential answers to P. However, we show here that unless $\mathcal{P}$=$\mathcal{NP}$, such a rewriting scheme does not exist for all datalog programs. First, I show that for some datalog program P and finite set of constraints C, $\Xi_{P,C}$ is $\mathcal{NP}$-Hard. If we could rewrite P into a datalog program P' such that for all finite sets of facts D and all facts d, $\langle P, C, D\rangle \vdash_{\exists\mathrm{dl}} d$ if and only if $\langle P', D\rangle \vdash_{\mathrm{dl}} d$, we have reduced the problem $\Xi_{P,C}$ to the problem $\Delta_{P'}$, which is in $\mathcal{P}$. So unless $\mathcal{P}$=$\mathcal{NP}$, such a rewriting does not exist.

[2] The termination of the rewriting algorithm in [17] depends on the finiteness of the resolution closure of $P \cup C$. In the case of nonrecursive datalog P and denial constraints C, the union is essentially a finite set of nonrecursive Horn clauses, which has finite closure under resolution.

4.1 $\mathcal{NP}$-Completeness

First, I show that the existential answer problem is $\mathcal{NP}$-Hard by reduction from `MONOTONE-3SAT`, a well-known $\mathcal{NP}$-complete problem. I first outline the approach:

1. Fix a datalog program P and a finite constraint set C.
2. Define a polynomial-time transformation from any monotone 3CNF formula Φ to a finite set of facts D_Φ and a query fact d_Φ.
3. Show that for any monotone 3CNF formula Φ, Φ is satisfiable if and only if $\langle P, C, D_\Phi\rangle \vdash_{\exists \mathrm{dl}} d_\Phi$.

`MONOTONE-3SAT` is like `3SAT` except that the input 3CNF formula is further restricted to be monotone, that is, each clause consists of either all positive literals or all negative literals, but never a mix of positive and negative literals. More precisely, a monotone 3CNF formula is a formula $(p_{1,1} \vee p_{1,2} \vee p_{1,3}) \wedge (p_{2,1} \vee p_{2,2} \vee p_{2,3}) \wedge \cdots \wedge (p_{n,1} \vee p_{n,2} \vee p_{n,3}) \wedge (q_{1,1} \vee q_{1,2} \vee q_{1,3}) \wedge (q_{2,1} \vee q_{2,2} \vee q_{2,3}) \wedge \cdots \wedge (q_{m,1} \vee q_{m,2} \vee q_{m,3})$, where each $p_{i,j}$ is a propositional variable and each $q_{i,j}$ is the negation of a propositional variable. Without loss of generality, we can assume $m \geq 1$ and $n \geq 1$.

Consider the datalog program $P = \{$
$\mathtt{qp(x,y)} : - \ \mathtt{ep(x,y,u,v)}$,
$\mathtt{qp(x,y)} : - \ \mathtt{qp(x,z), ep(z,y,u,v)}$,
$\mathtt{qn(x,y)} : - \ \mathtt{en(x,y,u,v)}$,
$\mathtt{qn(x,y)} : - \ \mathtt{qn(x,z), en(z,y,u,v)}$,
$\mathtt{Q(x,y,z)} : - \ \mathtt{qp(x,y), qn(x,z)}$
$\}$

Each $ep(x,y,u,v)$ or $en(x,y,u,v)$ is thought of as an edge from x to y, annotated with u and v. $Q(x,y,z)$ means there is an *ep*-path from x to y and also an *en*-path from x to z.

Now consider the constraint
$c = \forall x_1 y_1 u_1 v x_2 y_2 u_2(\neg ep(x_1, y_1, u_1, v) \vee \neg en(x_2, y_2, u_2, v))$. The constraint says that no pair of *ep*-edge and *en*-edge can coexist with the same v value.

For each monotone 3CNF formula $\Phi = (p_{1,1} \vee p_{1,2} \vee p_{1,3}) \wedge (p_{2,1} \vee p_{2,2} \vee p_{2,3}) \wedge \cdots \wedge (p_{n,1} \vee p_{n,2} \vee p_{n,3}) \wedge (q_{1,1} \vee q_{1,2} \vee q_{1,3}) \wedge (q_{2,1} \vee q_{2,2} \vee q_{2,3}) \wedge \cdots \wedge (q_{m,1} \vee q_{m,2} \vee q_{m,3})$, we do the following transformation:
Let $a_1, \ldots, a_p$ be the propositional variables. Define $D_\Phi = \{ep(i, i+1, j, k) : i = 1, \ldots, n; j = 1, 2, 3; a_k = p_{i,j}\} \cup \{en(i, i{+}1, j, k) : i = 1, \ldots, m; j = 1, 2, 3; \neg(a_k) = p_{i,j}\}$

The transformation is clearly polynomial.

Now $\langle P, \{c\}, D\rangle \vdash_{\exists \mathrm{dl}} Q(1, n+1, m+1)$, if and only if there is an *ep*-path from 1 to $n+1$ and an *en*-path from 1 to $m+1$ such that no pair of *ep*-edge and *en*-edge conflict by sharing the same v-value. Consider choosing each $ep(x,y,u,v)$-edge to be choosing to make $p_{x,u}$ true in the corresponding SAT instance. Also consider choosing each $en(x,y,u,v)$-edge to be choosing to make $q_{x,u}$ true in the corresponding SAT instance. Having an *ep*-path from 1 to $n+1$ and an *en*-path from 1 to $m+1$ is equivalent to picking one of $p_{i,1}, p_{i,2}, p_{i,3}$ to be true for all i from 1 to n and one of $q_{i,1}, q_{i,2}, q_{i,3}$ to be true for all i from 1 to m. Furthermore, the fact that the two paths do not conflict by sharing the same v-value is equivalent to that the set of literals picked to be true to not contain a complementary pair. This is exactly the condition required to choose an assignment of truth values that satisfy the formula Φ. We have a polynomial reduction from `MONOTONE-3SAT` to existential answers.

Hence, the existential answers problem with this fixed query and constraint is $\mathcal{NP}$-Hard.

It is easy to see that $\varXi_{P,C}$ is in $\mathcal{NP}$. For a finite set of facts D and a fact d, $\langle P, C, D\rangle \vdash_{\exists \mathrm{dl}} d$ if and only if there exists an admissible set $D' \subseteq D$ such that $d \in P(D')$. One can nondeterministically guess the appropriate D' and then verify in polynomial time that $d \in P(D')$ and $P(D')$ is a model of C.

Theorem 1 ($\mathcal{NP}$-Completeness). *The existential answer problem $\varXi_{P,C}$ is $\mathcal{NP}$-Complete for some datalog program P and finite set of denial constraints C.*

We conclude that unless $\mathcal{P}$=$\mathcal{NP}$, some existential answer problems over datalog programs and denial constraint cannot be solved by rewriting into datalog (or any other language with polynomial data-complexity, for example, stratified datalog¬).

Theorem 2 (Non-rewritability into dalalog). *Assuming $\mathcal{P} \neq \mathcal{NP}$, for some datalog program P and finite set of denial constraints C, there does not exist a datalog program P' such that for all finite sets of facts D and all facts d, $\langle P, C, D\rangle \vdash_{\exists \mathrm{dl}} d$ if and only if $\langle P', D\rangle \vdash_{\mathrm{dl}} d$.*

5 Provenance

Provenance of data has received considerable attention. Here, we use the method of Agrawal, Benjelloun, Das Sarma, Halevy, Theobald, and Widom [8, 2] to compute and represent data provenance. We briefly review their method here.

Each tuple is associated with a tuple ID. Each computed tuple is associated with a provenance set that gives the set of tuple IDs from which this tuple is derived.

Consider the following example.

Example 3. The base facts D is given by the following table. A fact `link(i, j, t)` represents that there is a direct link from `i` to `j` of type `t`.

link

ID	origin	destination	type	
21	a	b	1	$\lambda(21) = \{\}$
22	a	c	2	$\lambda(22) = \{\}$
23	b	d	2	$\lambda(23) = \{\}$
24	c	d	3	$\lambda(24) = \{\}$
25	d	e	1	$\lambda(25) = \{\}$
26	d	f	2	$\lambda(26) = \{\}$

$P =$
$\{\texttt{aReaches(Y)} :- \texttt{link(a,Y,U)},$
$\texttt{aReaches(Y)} :- \texttt{aReaches(a,Z,U)}, \texttt{link(Z,Y,V)}\}$

The following are the computed facts and their associated provenance information.

aReaches

ID	**destination**	
31	b	$\lambda(31) = \{21\}$
32	c	$\lambda(32) = \{22\}$
33	d	$\lambda(33) = \{31, 23\}$
34	d	$\lambda(34) = \{32, 24\}$
35	e	$\lambda(35) = \{33, 25\}$
36	e	$\lambda(36) = \{34, 25\}$
37	f	$\lambda(37) = \{33, 26\}$
38	f	$\lambda(38) = \{34, 26\}$

If a fact is produced multiple times from different sets of tuples, the fact is presented as multiple tuples, each with its own tuple ID and associated provenance. The computation of provenance can be embedded in a standard bottom-up datalog evaluation procedure. It is important to note that where regular datalog evaluation (set semantics, no duplicate tuples) has polynomial data complexity, datalog evaluation with provenance computation as shown above can be exponential in the number of facts in the input fact set. The increased complexity is due to the fact that even though the number of unique facts that can be generated is polynomially bounded, the number of unique (fact, provenance) pairs is not.

6 Computing existential answers via provenance

The brute-force method of enumerating each set $D' \subseteq D$ is impractical. In this section, I present a method that uses provenance information to construct several special subsets of D called support sets and then consider only those sets. The IsExistentialAnswer procedure calls on the SupportSet procedure defined immediately after.

```
IsExistentialAnswer($P, C, D, id, \lambda$)
  $D'$ := SupportSet($D, id, \lambda$)
  $C^*$ := $\{nogood :- R_1(u_1), \ldots, R_n(u_n), \phi(u_\phi) :$
          $(\forall x_1 \cdots x_m(\neg R_1(u_1) \vee \cdots \neg R_n(u_n) \vee \phi(u_\phi))) \in C\}$
  (Where nogood is a new relation name that is not in the current schema)
  if $nogood \in C^*(P(D'))$ then
    return FALSE
  else
    return TRUE
  end if
```

```
SupportSet($D, id, \lambda$)
  Queue $Q$ := empty queue
  $D' := \{\}$
  $Q$.enqueue($id$)
  while $Q$ is not empty do
    $id' := Q$.dequeue()
    if $\lambda(id') == \emptyset$ then
      $D' := D' \cup \{fact(id')\}$ ($fact(id')$ gives the fact associated with the ID $id'$)
    else
      for $l \in \lambda(id')$ do
        $Q$.enqueue($l$)
      end for
    end if
  end while
  return D'
```

Theorem 3 (Correctness). *For a datalog program P, a set of facts D, a set of denial constraints C, a fact $d \in P(D)$, and a provenance function λ obtained in evaluating $P(D)$, $\langle P, C, D\rangle \vdash_{\exists \mathrm{dl}} d$ if and only if* IsExistentialAnswer(P, C, D, id, λ) *returns TRUE for a tuple ID id of d.*

Example 4. Using the provenance information in example 3 and the datalog program P and constraints C from example 2, I demonstrate the IsExistentialAnswer algorithm.

The tuple ID's associated with the fact $\mathtt{aReaches(f)} \in P(D)$ are 37 and 38. Running IsExistentialAnswer$(P, D, C, 37, \lambda)$ constructs the set $D' = \{\mathtt{link(d,f,2)}, \mathtt{link(b,d,2)}, \mathtt{link(a,b,1)}\}$ from the set of base tuple ID's $\{26, 23, 21\}$. It also constructs $C^* := \{$

$\mathtt{nogood} :- \mathtt{link'(X_1, Y_1, U)}, \mathtt{link'(X_2, Y_2, U)}, \mathtt{X_1} \neq \mathtt{X_2}$,

$\mathtt{nogood} :- \mathtt{link'(X_1, Y_1, U)}, \mathtt{link'(X_2, Y_2, U)}, \mathtt{Y_1} \neq \mathtt{Y_2}$

$\}$. Finally, *nogood* $\in C^*(P(D'))$, so the algorithm rejects the tuple of ID 37 as an existential answer tuple.

Running IsExistentialAnswer$(P, D, C, 38, \lambda)$ constructs the set $D' = \{\mathtt{link(d,f,2)}, \mathtt{link(c,d,3)}, \mathtt{link(a,c,2)}\}$ from the set of base tuple ID's $\{26, 24, 22\}$. It also constructs the same C^* as above.
Finally, *nogood* $\in C^*(P(D'))$, so the algorithm also rejects the tuple of ID 38 as an existential answer tuple.

One can conclude that $\langle P, C, D\rangle \nvdash_{\exists \mathrm{dl}}$ $\mathtt{aReaches(f)}$, in agreement with example 2.

Now I examine whether $\langle P, C, D\rangle \vdash_{\exists \mathrm{dl}}$ $\mathtt{aReaches(e)}$.The tuple ID's associated with the fact $\mathtt{aReaches(e)} \in P(D)$ are 35 and 36.

Running IsExistentialAnswer$(P, D, C, 35, \lambda)$ constructs the set $D' = \{\mathtt{link(d,e,1)}, \mathtt{link(b,d,2)}, \mathtt{link(a,b,1)}\}$ from the set of base tuple ID's $\{25, 23, 21\}$. *nogood* $\in C^*(P(D'))$, so the algorithm rejects the tuple of ID 35 as an existential answer tuple.

Running IsExistentialAnswer$(P, D, C, 36, \lambda)$ constructs the set $D' = \{\texttt{link(d, e, 1)}, \texttt{link(c, d, 3)}, \texttt{link(a, c, 2)}\}$ from the set of base tuple ID's $\{25, 24, 22\}$. $\mathit{nogood} \notin C^*(P(D'))$, so the algorithm accepts the tuple of ID 36 as an existential answer tuple.

One can conclude that $\langle P, C, D\rangle \vdash_{\exists\mathrm{dl}}$ aReaches(e), in agreement with example 2.

7 Answer Set Programming

Answer set programming (ASP) has received much attention in the logic programming community, the database community, and the rules and ontology community because of its ability to declaratively and succinctly express interesting problems from a wide-range of domains. ASP also offers another way to characterize the existential answers. In this section, I characterize existential answers as the solutions to some answer set programs. The use of disjunctive logic programs to specify changes to data was introduced by Arenas and Bertossi and Chomicki [4]. I assume that the reader is familiar with the basics of answer set programming. For an introduction to answer set programming, I recommend Lifschitz's overview [24].

Let the transformation $\mathit{primed}(\phi)$ take a formula (or rule) ϕ and replace within ϕ each relation name R by R'. For a datalog program P, and a finite set of denial constraints C, we define the transformation into an ASP
ExistentialASP$(P) := \{(R'(x)\ :-\ R(x), \mathit{not}\ R'(x)) : R$ a relation name$\}$
$\cup\ \{\mathit{primed}(r) : r \in C \cup P\}$

Example 5.
Let $P = \{$
aReaches(Y) : − link(a, Y, U),
aReaches(Y) : − aReaches(a, Z, U), link(Z, Y, V)$\}$

Let $C = \{$
(false) : − link(X_1, Y_1, U), link(X_2, Y_2, U), $X_1 \neq X_2$,
(false) : − link(X_1, Y_1, U), link(X_2, Y_2, U), $Y_1 \neq Y_2$
$\}$

ExistentialASP$(P, C) = \{$
link′(X, Y, U) : − link(X, Y, U), not link′(X, Y, U),
aReaches′(X, Y, U) : − aReaches(X, Y, U), not aReaches′(X, Y, U),
aReaches′(Y) : − link′(a, Y, U),
aReaches′(Y) : − aReaches′(a, Z, U), link′(Z, Y, V),
(false) : − link′(X_1, Y_1, U), link′(X_2, Y_2, U), $X_1 \neq X_2$,
(false) : − link′(X_1, Y_1, U), link′(X_2, Y_2, U), $Y_1 \neq Y_2$
$\}$

Theorem 4. *Given a datalog program P, a finite set of constraints C (written as headless rules), then for any a finite set of facts D and any fact d,*

$\langle P, C, D\rangle \vdash_{\exists \mathrm{dl}} d$ *if and only if primed*(d) *is in an answer set to the answer set program* `ExistentialASP`(P, C).

The characterization of existential answers as credulous answers to an answer set program gives another method to compute existential answers – reformulate an existential answer problem as an ASP problem, then use an ASP solver to find the credulous answers. However, the cost of this method may be prohibitively high because most general-purpose ASP solvers work by first grounding out the input program[3].

8 Related Work

The importance of adequately processing inconsistencies in reasoning on the web has received wide recognition, but there have been relatively few pieces of work that develop techniques for reasoning with inconsistent knowledge on the web. Notable works in this area include the following [18, 23, 28, 29, 3]. None of them take a provenance-based approach to reason in the presence of inconsistencies.

The idea of finding answers from consistent subsets of a theory is due to Elvang-Gøransson and Hunter[14]. The idea is later refined by Kassoff, Zen, Garg, and Genesereth for application to logical spreadsheets [22, 20].

Kassoff and Genesereth [21] also proposed a method for computing existential answers based on provenance. However, their method is intended for reasoning with first-order theories and use a general resolution-refutation method rather than one based on standard datalog evaluation. As a result, the method is not guaranteed to terminate (even if restricted to horn clauses, which correspond to datalog).

ULDB [8, 2] uses provenance information to compute answers from uncertain databases. The approach is similar to the method presented in this paper. However, the setting they consider is one where all constraints are ground.

As mentioned in section 7, the specification of database repairs using disjunctive logic programs with exceptions first appeared in [4]. Many other works have followed that are based on the same basic idea (some examples include [9, 7, 5, 6, 12, 13, 15]).

9 Future work

The connection between provenance and trust creates a fertile ground for future work in reasoning on the web in the presence of inconsistencies and uncertainties.

As an example, I define the *prioritized existential answers* to account for the fact that some sources are more trusted than others. Existential answers, as defined in section 3, give all facts equal priority. However, in real applications, some sources are known to be much more reliable than others. In the birth year example in section 2, the tuple `BirthYear(David, 1984)` may be more trusted

[3] [27] is a notable exception.

and hence prioritized over the conflicting tuple `BirthYear(David, 1980)`. Then one would reject `David` as a candidate.

With this intuition in mind, I define the *prioritized existential answers.*

Definition 5 (Prioritized Existential Answers). *Assume a (possibly empty) partial order $\succ$ on the set of all facts over the schema. (Intuitively, $d \succ d'$ means that d is prioritized over d'.) Given a set of facts D, a set of rules P, and a set of constraints C a fact d is a prioritized existential answers with respect to $(D, P, C, \succ)$ if and only if there exists $D' \subseteq D$ such that*

1. *$P(D')$ is a model of C,*
2. *$(\forall g \in (D - D'))(\forall g' \in D')(g \not\succ g')$,*
3. *and $\langle D', P \rangle \vdash_{\mathrm{dl}} d$.*

We denote this fact as $\langle P, C, D \succ \rangle \vdash_{\exists \mathrm{dl}} d$.

One can check in example 1 that if
$\mathtt{BirthYear}(\mathtt{David}, 1984) \succ \mathtt{BirthYear}(\mathtt{David}, 1980)$, then `candidate(David)` is not a prioritized existential answer.

A direction for future work is to design a provenance-based method to compute prioritized existential answer.

Acknowledgments

I would like to thank Monica Palmirani, Davide Sottara, Frank Olken and the rest of the RuleML 2011 (America) Program Committee for the invitation to submit this paper. I am grateful to Michael Genesereth, Jeff Ullman, Jennifer Widom, Michael Kassoff, Rada Chirkova, Mary-Anne Williams, Carl Hewitt, Lukasz Golab, and attendees of Stanford Logic Group and Stanford Infolab seminars, for their valuable feedback. I would also like to acknowledge the NSERC[4] and Konica-Minolta (via the MediaX project) for their financial support.

References

1. Abiteboul, S., Hull, R., Vianu, V.: Foundations of databases (1995)
2. Agrawal, P., Benjelloun, O., Sarma, A.D., Hayworth, C., Nabar, S., Sugihara, T., Widom, J.: Trio: A system for data, uncertainty, and lineage. In: 32nd International Conference on Very Large Data Bases. VLDB 2006 (demonstration description) (September 2006), http://ilpubs.stanford.edu:8090/776/
3. Alejandro Gmez, S., Ivn Chesevar, C., Simari, G.R.: Reasoning with inconsistent ontologies through argumentation. Applied Artificial Intelligence 24(1-2), 102–148 (2010), http://www.tandfonline.com/doi/abs/10.1080/08839510903448692
4. Arenas, M., Bertossi, L., Chomicki, J.: Specifying and querying database repairs using logic programs with exceptions. In: In Flexible Query Answering Systems. Recent Developments. pp. 27–41. Springer (2000)

[4] Natural Sciences and Engineering Research Council of Canada

5. Arenas, M., Bertossi, L., Chomicki, J.: Answer sets for consistent query answering in inconsistent databases. THEORY AND PRACTICE OF LOGIC PROGRAMMING 3(4), 393–424 (2003)
6. Barcel, P., Bertossi, L.: Repairing databases with annotated predicate logic. In: Ninth International Workshop on Non-Monotonic Reasoning (NMR02), Special Session: Changing and Integrating Information: From Theory to Practice. pp. 160–170. Morgan Kaufmann Publishers (2002)
7. Barcel, P., Bertossi, L.: Logic programs for querying inconsistent databases. In: In International Symposium on Practical Aspects of Declarative Languages (PADL. pp. 208–222. Springer-Verlag, LNCS (2003)
8. Benjelloun, O., Das Sarma, A., Halevy, A., Theobald, M., Widom, J.: Databases with uncertainty and lineage. The VLDB Journal 17, 243–264 (March 2008), http://dx.doi.org/10.1007/s00778-007-0080-z
9. Bravo, L., Bertossi, L.: Logic programs for consistently querying data integration systems. In: In International Joint Conference on Artificial Intelligence (IJCAI. pp. 10–15 (2003)
10. Chebotko, A., Lu, S., Fei, X., Fotouhi, F.: Rdfprov: A relational rdf store for querying and managing scientific workflow provenance. Data Knowl. Eng. 69, 836–865 (August 2010), http://dx.doi.org/10.1016/j.datak.2010.03.005
11. Ding, L., Michaelis, J., McCusker, J., McGuinness, D.L.: Linked provenance data: A semantic web-based approach to interoperable workflow traces. Future Gener. Comput. Syst. 27, 797–805 (June 2011), http://dx.doi.org/10.1016/j.future.2010.10.011
12. Eiter, T.: Data integration and answer set programming. In: In Proc. LPNMR05, number 3662 in LNCS. pp. 13–25. Springer (2005)
13. Eiter, T., Fink, M., Greco, G., Lembo, D.: Optimization methods for logic-based query answering from inconsistent data integration systems (2005)
14. Elvang-Gøransson, M., Hunter, A.: Argumentative logics: Reasoning with classically inconsistent information. Data Knowl. Eng. 16(2), 125–145 (1995)
15. Espil, M.M., Vaisman, A.A., Terribile, L.: Revising data cubes with exceptions: A rule-based perspective (2002)
16. Fan, H., Poulovassilis, A.: Tracing data lineage using schema transformation pathways. In: KNOWLEDGE TRANSFORMATION FOR THE SEMANTIC WEB. pp. 64–79. IOS Press (2002)
17. Hinrichs, T.L., Kao, J.Y., Genesereth, M.: Inconsistency-tolerant reasoning with classical logic and large databases. In: Proc. of the eighth Symposium on Abstraction, Reformulation, and Approximation (2009)
18. Huang, Z., van Harmelen, F., ten Teije, A.: Reasoning with inconsistent ontologies. In: Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI'05). pp. 454–459. Edinburgh, Scotland (August 2005)
19. Kao, E.J.Y., Genesereth, M.: Query rewriting with filtering constraints. Tech. Rep. LG-2009-02, Stanford University, Stanford, CA (2009), http://logic.stanford.edu/reports/LG-2009-02.pdf, updated July,2011
20. Kassoff, M., Genesereth, M.: Predicalc: A logical spreadsheet management system. Knowl. Eng. Rev. 22(3), 281–295 (2007)
21. Kassoff, M., Genesereth, M.R.: Paraconsistent inference from data using ω-existential entailment. DALI: Workshop on Data, Logic and Inconsistency (2011)
22. Kassoff, M., Zen, L.M., Garg, A., Genesereth, M.: Predicalc: a logical spreadsheet management system. In: VLDB '05: Proceedings of the 31st international conference on Very large data bases. pp. 1247–1250. VLDB Endowment (2005)

23. Li, D., Lin, Y., Huang, H., Tian, X.: Linear reduction reasoning with inconsistent ontology. In: Computational Sciences and Optimization (CSO), 2011 Fourth International Joint Conference on. pp. 795 –798 (april 2011)
24. Lifschitz, V.: What is answer set programming? In: Proceedings of the 23rd national conference on Artificial intelligence - Volume 3. pp. 1594–1597. AAAI Press (2008), http://portal.acm.org/citation.cfm?id=1620270.1620340
25. Moreau, L.: Provenance-based reproducibility in the semantic web. Journal of Web Semantics (February 2011), http://eprints.ecs.soton.ac.uk/21992/
26. Moreau, L., Clifford, B., Freire, J., Futrelle, J., Gil, Y., Groth, P., Kwasnikowska, N., Miles, S., Missier, P., Myers, J., Plale, B., Simmhan, Y., Stephan, E., den Bussche, J.V.: The open provenance model core specification (v1.1). Future Generation Computer Systems 27(6), 743 – 756 (2011), http://www.sciencedirect.com/science/article/pii/S0167739X10001275
27. Palù, A., Dovier, A., Pontelli, E., Rossi, G.: Answer set programming with constraints using lazy grounding. In: Proceedings of the 25th International Conference on Logic Programming. pp. 115–129. ICLP '09, Springer-Verlag, Berlin, Heidelberg (2009), http://dx.doi.org/10.1007/978-3-642-02846-5_14
28. Schobach, S., Corner, R.: Non-standard reasoning services for the debugging of description logic terminology. In: IJCAI (2003)
29. Zlatareva, N.P.: Supporting uncertainty and inconsistency in semantic web applications. In: FLAIRS Conference (2009)